People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

YAHIA FARES UNIVERSITY OF MEDEA

Faculty of Sciences

Department of Mathematics and Computer Science



Graduation Project

Submitted for Master Degree in Computer Science

Specialty: System Engineering and Web Technology

Enhancing Malware Detection Performance Using Hybrid Machine Learning Based Feature Selection Methods

---

Supervisor:

Dr. Sahmadi Brahim

Presented by:

Fellague Affaf

Hamidia Madina Djamila

2025/2026

Dedication

To the Almighty, for the strength to begin and the grace to finish. To our parents our first teachers and our lifelong anchors. This work is the fruit of your sacrifices, your prayers, and your unconditional love. May Allah grant you the highest rewards for every step you enabled us to take. To our families, for being the quiet strength behind every page.

Acknowledgement

First and foremost, we would like to express our deepest gratitude to our supervisor, Dr. Sahmadi Brahim. Your patient guidance, insightful feedback, and constant encouragement throughout this process have been invaluable. Thank you for pushing us to think critically and for helping us navigate the complexities of this research. We would also like to thank the faculty and staff at Yahia Fares University of Médéa, particularly the members of the Mathematics and Computer Science department, for their support and for providing the academic environment necessary for this work. In recognition of our individual support: Fellag Affaf would like to express her profound gratitude to her beloved parents. To my mother and father: thank you for your endless love, your sacrifices, and for being the foundation upon which my education was built. Your prayers have been my greatest strength throughout my academic journey.

Hamidia Madina Djamila wishes to express heartfelt gratitude to their parents and family for their unwavering support, encouragement, and belief in their potential during every step of this endeavor.

Finally, we thank each other. This thesis is not just a result of academic research, but of a shared journey, mutual support, and a partnership that grew stronger through every challenge. To our friends and colleagues who stood by us thank you for being part of this achievement

Abstract

The rapid proliferation of Android malware poses a persistent threat to mobile security, yet machine learning-based detectors typically operate on high-dimensional feature spaces that introduce significant computational overhead, reduce model interpretability, and limit practical deployment in resource-constrained environments. This thesis addresses these challenges by proposing a hybrid two-stage feature selection framework that combines an ensemble of three filter methods Information Gain, Fisher Score, and Chi-Squared with a Binary Whale Optimisation Algorithm (BWOA) wrapper search guided by Random Forest and XGBoost classifiers, using task-specific fitness functions: F2-score for binary detection to prioritise malware recall, and F1-macro for multiclass categorisation to balance performance across malware families. Evaluated on the CICMalDroid 2020 and TUANDROMD benchmarks, the proposed BWOA-RF configuration achieves 97.93% binary detection accuracy and the highest malware recall (96.05%) among all configurations, using only 87 features an 81.5% reduction from the original 470 while the BWOA-Ensemble configuration achieves 95.00% accuracy and a macro F1 of 0.9405 with 116 features (75.3% reduction), outperforming both full-feature baselines. Cross-dataset validation on the structurally distinct TUANDROMD benchmark confirms cross-representation generalisation with 99.78% accuracy and a 56.8% feature reduction. Feature stability analysis identifies a consistent set of security-meaningful behavioural indicators including getDeviceId, mprotect, and writev selected reliably across independent optimisation runs, providing actionable insight for real-world Android malware detection.

Keywords: Android malware detection, feature selection, Binary Whale Optimisation Algorithm, machine learning, Random Forest, XGBoost, dynamic analysis, syscall features, dimensionality reduction.

ملخص

تُشكّل الزيادة السريعة في برمجيات أندرويد الخبيثة تهديدًا مستمرًا لأمن الأجهزة المحمولة، إلا أن أنظمة الكشف المعتمدة على التعلم الآلي تعمل عادةً ضمن فضاءات سمات عالية الأبعاد، مما يؤدي إلى زيادة كبيرة في التكلفة الحاسوبية، وتقليل قابلية تفسير النماذج، والحد من إمكانية نشرها عملياً في البيئات ذات الموارد المحدودة. تعالج هذه الرسالة هذه التحديات من خلال اقتراح إطار هجين من مرحلتين لاختيار السمات، يجمع بين مجموعة من ثلاث طرق ترشيح هي Gain Information وScore Fisher وChi-Squared، مع خوارزمية Algorithm Optimisation Whale Binary (BWOA) كطريقة تغليف (Wrapper) موجهة بواسطة مصنفي Forest Random وXGBoost، باستخدام دوال ملاءمة مخصصة للمهمة؛ حيث استُخدم مقياس F2-score للكشف الثنائي لإعطاء أولوية أعلى لاسترجاع البرمجيات الخبيثة، بينما استُخدم F1-macro للتصنيف متعدد الفئات لتحقيق توازن في الأداء بين عائلات البرمجيات الخبيثة المختلفة. وقد أُجري التقييم باستخدام مجموعتي البيانات CICMalDroid 2020 وTUANDROMD، حيث حقق إعداد BWOA-RF دقة بلغت 97.93% في الكشف الثنائي، وأعلى معدل استرجاع للبرمجيات الخبيثة بلغ 96.05% بين جميع الإعدادات، مع استخدام 87 سمة فقط، أي بانخفاض قدره 81.5% مقارنةً بالسمات الأصلية البالغ عددها 470. كما حقق إعداد BWOA-Ensemble دقة بلغت 95.00% وقيمة F1-macro مقدارها 0.9405 باستخدام 116 سمة (بانخفاض قدره 75.3%)، متفوقًا على النماذج الأساسية التي تستخدم جميع السمات. وأكد التحقق المتقاطع بين مجموعات البيانات على مجموعة TUANDROMD، ذات البنية المختلفة، قدرة النموذج على التعميم عبر تمثيلات مختلفة للبيانات، محققًا دقة بلغت 99.78% مع خفض عدد السمات بنسبة 56.8%. كما أظهر تحليل استقرار السمات مجموعة متسقة من المؤشرات السلوكية ذات الدلالة الأمنية، مثل getDeviceId وmprotect وwritev، والتي جرى اختيارها باستمرار عبر عمليات التحسين المستقلة، مما يوفر رؤى عملية قابلة للتطبيق في أنظمة الكشف عن برمجيات أندرويد الخبيثة في البيئات الواقعية.

الكلمات المفتاحية: كشف برمجيات أندرويد الخبيثة، اختيار الخصائص، خوارزمية الحوت الثنائية للتحسين، التعلم الآلي، Random Forest، XGBoost، التحليل الديناميكي، خصائص استدعاءات النظام، تقليل الأبعاد.

Résumé

La prolifération rapide des logiciels malveillants Android constitue une menace persistante pour la sécurité des appareils mobiles. Toutefois, les détecteurs fondés sur l'apprentissage automatique exploitent généralement des espaces de caractéristiques de grande dimension, ce qui engendre un coût computationnel élevé, réduit l'interprétabilité des modèles et limite leur déploiement dans des environnements à ressources limitées. Cette thèse répond à ces défis en proposant un cadre hybride de sélection de caractéristiques en deux étapes, combinant trois méthodes de filtrage Information Gain, Fisher Score et Chi-Squared — avec une recherche de type wrapper basée sur le Binary Whale Optimisation Algorithm (BWOA), guidée par les classificateurs Random Forest et XGBoost, en utilisant des fonctions d'évaluation adaptées à chaque tâche : le F2-score pour la détection binaire afin de privilégier le rappel des logiciels malveillants, et le F1-macro pour la classification multiclasse afin d'assurer un équilibre des performances entre les différentes familles de logiciels malveillants. Les expériences menées sur les jeux de données CICMalDroid 2020 et TUANDROMD montrent que la configuration BWOA-RF atteint une précision de 97,93 % en détection binaire ainsi que le meilleur taux de rappel (96,05 %) parmi toutes les configurations, tout en utilisant seulement 87 caractéristiques, soit une réduction de 81,5% par rapport aux 470 caractéristiques initiales, tandis que la configuration BWOA-Ensemble obtient une précision de 95,00% et un F1-macro de 0,9405 avec 116 caractéristiques (réduction de 75,3%), surpassant les modèles de référence utilisant l'ensemble des caractéristiques. En outre, la validation croisée sur le jeu de données TUANDROMD, dont la représentation est structurellement différente, confirme la capacité de généralisation du modèle avec une précision de 99,78 % et une réduction de 56,8 % du nombre de caractéristiques. Enfin, l'analyse de la stabilité des caractéristiques met en évidence un ensemble cohérent d'indicateurs comportementaux pertinents pour la sécurité, notamment getDeviceId, mprotect et writev, sélectionnés de manière fiable au cours d'exécutions indépendantes de l'algorithme d'optimisation, fournissant ainsi des informations exploitables pour les systèmes réels de détection des logiciels malveillants Android

Mots-clés : Détection de malwares Android, sélection de caractéristiques, algorithme d'optimisation de la baleine binaire, apprentissage automatique, Random Forest, XGBoost, analyse dynamique, appels système, réduction de dimensionnalité.

Contents

Dedication **1**

Acknowledgement **2**

Abstract **3**

ملخص **4**

Résumé **5**

General Introduction **15**

Context and Motivation 15

Research Goals 16

Contributions 17

Structure 17

1 Android Malware Detection Background **18**

1.1 Introduction 18

1.2 The Android Operating System 19

1.2.1 Architecture 19

1.2.2 Application structure 20

1.2.3 security model 21

1.2.4 Application Distribution and Installation Mechanisms 22

1.3 Android Malware 23

1.3.1 Definitions and Taxonomy 23

1.3.2 Malware Infection Vectors and Propagation Techniques 24
1.3.3 Evolution and Trends in Android Malware 25
1.4 Android Malware Analysis Techniques 26
1.4.1 Static analysis 27
1.4.2 Dynamic analysis 27
1.4.3 Hybrid analysis 27
1.5 Android Malware Detection Approaches 27
1.5.1 Signature-based detection 28
1.5.2 Heuristic and rule-based detection 28
1.5.3 Machine learning–based detection 29
1.6 Feature Types for Android Malware Detection 30
1.7 ML for Android Malware Detection 31
1.7.1 Machine Learning–Based Detection 31
1.7.2 Limitations of Traditional Machine Learning 32
1.8 Conclusion 33

2 Feature Selection Techniques for Android Malware Detection and Literature Review 34

2.1 Introduction 34
2.2 Feature Selection Fundamentals 35
2.2.1 Definition and Concepts 35
2.2.2 Feature Relevance 36
2.2.3 Feature Redundancy 36
2.2.4 Importance of Feature Selection 36
2.3 Categories of Feature Selection Methods 38
2.3.1 Filter Methods 39
2.3.2 Wrapper Methods 40
2.3.3 Embedded Methods 43
2.4 Metaheuristic Optimization for Feature Selection 44
2.4.1 Overview of Metaheuristics 44
2.4.2 Common Metaheuristic Algorithms 45

2.4.3 Advantages in High-Dimensional Problems 46
2.5 Literature Review 47
2.5.1 Feature Selection-Based Android Malware Detection Approaches 47
2.5.2 Machine Learning-Based Detection Approaches 48
2.5.3 Deep Learning and Advanced Detection Approaches 49
2.5.4 Dynamic and System Call-Based Approaches 49
2.5.5 Systematic Review of the Field 49
2.6 Research Gap and Motivation 50
2.7 Conclusion 52

3 Proposed Approach – Hybrid Tree-Based Wrapper Feature Selection Framework 53
3.1 Introduction 53
3.2 Overview of the Proposed Framework 54
3.3 Data Preprocessing Pipeline 56
3.3.1 Handling Missing and Infinite Values 56
3.3.2 Feature Normalization 57
3.3.3 Feature Encoding 57
3.3.4 Handling Class Imbalance 57
3.3.5 Dataset Splitting Strategy 59
3.4 Binary Whale Optimization Algorithm for Feature Subset Search 60
3.4.1 Overview of the Whale Optimization Algorithm 60
3.4.2 Solution Encoding 61
3.4.3 Population Initialization 61
3.4.4 Transfer Function for Continuous-to-Binary Conversion . . . 62
3.4.5 WOA Operators Adapted for Binary Feature Selection 63
3.4.6 Pseudocode of the Adapted Binary WOA 65
3.5 Fitness Evaluation Using Random Forest and XGBoost 66
3.5.1 Fitness Function Design 66
3.5.2 XGBoost Classifier 68
3.5.3 Iterative Optimization Loop 69

3.6 Final Model Training and Evaluation 69
3.6.1 Consensus Feature Selection 69
3.6.2 Dual-Task Evaluation 69
3.7 Conclusion 70

4 Experiments and Results **71**
4.1 Introduction 71
4.2 Experimental Environment 71
4.2.1 Hardware Configuration 71
4.2.2 Software Environment 72
4.2.3 Parameter Settings 73
4.3 Datasets Description 73
4.3.1 CICMalDroid 2020 Dataset 74
4.3.2 TUANDROMD Dataset 75
4.4 Evaluation Methodology 76
4.4.1 Evaluation Metrics 76
4.4.2 Baseline Methods 77
4.5 Binary Classification Results 78
4.5.1 Five-Fold Cross-Validation Results 78
4.5.2 CICMalDroid 2020 — Holdout Results 79
4.5.3 ROC and Precision Recall Curve Analysis 80
4.5.4 Confusion Matrix Analysis 82
4.5.5 TUANDROMD External Validation 85
4.6 Multiclass Classification Results 86
4.6.1 Cross-Validation Results — CICMalDroid 2020 86
4.6.2 Overall Multiclass Holdout Results 87
4.6.3 Per-Class Performance Analysis 89
4.6.4 Analysis of Difficult Classes 89
4.7 Feature Selection Analysis 90
4.7.1 Number of Selected Features and Reduction Ratio 90
4.7.2 Most Important Features 91

4.7.3 Feature Stability Analysis 93
4.8 Discussion 96
4.8.1 Impact of Hybrid Feature Selection 96
4.8.2 Computational Efficiency 97
4.8.3 Generalisation Capability 97
4.8.4 Comparison with Related Work 97
4.9 Threats to Validity 99
4.9.1 Internal Validity 99
4.9.2 External Validity 100
4.9.3 Construct Validity 100
4.9.4 Reproducibility 100
4.10 Chapter Conclusion 100

5 General Conclusion **102**
5.1 Summary of Contributions 102
5.2 Concluding Remarks 102
5.3 Future Research Directions 103

List of Figures

1.1 Android Architecture 19

1.2 Example of a permission request in an Android application from author (Mirabet Herranz, 2024)[18] 21

1.3 Projected global mobile user growth from 2020 to 2025 22

1.4 Graphical representation of the distinct types of malware. from (Mirabet Herranz, 2024) [18] 26

1.5 : Classification of malware detection techniques based on analysis type 28

1.6 The most used features for malware detection in 20 studies 32

2.1 Dimensionality Reduction Techniques Source: [1] 35

2.2 Feature selection reduction 36

2.3 A view of feature relevance and redundancy. Source: [1] 37

2.4 The general classification of feature selection methods according to their evaluation criteria. Source:[1] 38

2.5 Sequential Forward Selection (SFS) 42

2.6 Sequential Backward Selection (SBS) 42

3.1 Overall architecture of the proposed hybrid filter-BWOA-classifier framework for Android malware detection 54

3.2 Filter voting fusion diagram 56

3.3 Class distribution before and after SMOTE 58

3.4 Data splitting strategy diagram 59

3.5 Binary encoding diagram 61

3.6 *V-shaped vs S-shaped transfer function comparison* 62

3.7 WOA operator diagram 63

4.1 CICMalDroid 2020 class distribution. 75
4.2 TUANDROMD class distribution. 76
4.3 ROC curves for binary malware detection on the CICMalDroid 2020 holdout set. 80
4.4 Precision–Recall curves for binary malware detection on the CICMalDroid 2020 holdout set. 81
4.5 Confusion matrix for C3 (Filter+BWOA+RF). 83
4.6 Confusion matrix for C4 (Filter+BWOA+XGB). 83
4.7 Confusion matrix for C5 (Filter+BWOA+Ensemble). 84
4.8 TUANDROMD dataset. 85
4.9 Confusion matrix for C5 and C4 86
4.10 Confusion matrix for C1 C2 and C3 86
4.11 Confusion matrices for C5 (Filter+BWOA+Ensemble). 88
4.12 Confusion matrices for C3 (Filter+BWOA+RF) and C4 (Filter+BWOA+XGB). 88
4.13 KNN Naive Bayes — full vs BWOA-RF / BWOA-XGB / BWOA-ENS accuracy 89
4.14 Top 20 most important features by Gini impurity decrease from the final Random Forest model trained on the 116 BWOA-selected features. Bar length is proportional to importance score. 91
4.15 Feature stability heatmap showing BWOA selection frequency across five cross-validation folds. Darker cells indicate features selected in more folds. Gold-tier features appear in all five runs; Silver-tier in four; Bronze-tier in three. 95
4.16 BWOA convergence curves across five cross-validation folds for BWOA-RF (α=0.985, left) and BWOA-XGB (α=0.985, right). The bold line represents the mean fitness across folds. Both variants reach stable minima before the iteration budget is exhausted, confirming that the 40-iteration stopping criterion is not artificially constraining the search. 96

List of Tables

1.1 Types of Malware and Their Characteristics 24

2.1 Merits and demerits of search algorithms from A Review of Feature Selection and Its Methods by B. Venkatesh, J. Anuradha 45

2.2 Comparison of Android Malware Detection Studies with Emphasis on Feature Selection Practices 50

3.1 Data leakage prevention summary 60

4.1 Software environment and libraries used in experiments. 72

4.2 Final experimental parameter configuration. 73

4.3 CICMalDroid 2020 class distribution. 74

4.4 Binary classification — 5-fold cross-validation results (mean $\pm$ std). . . 78

4.5 Binary classification holdout results on CICMalDroid 2020. 79

4.6 Binary classification holdout results on TUANDROMD. $\star$ denotes proposed methods. 85

4.7 5-fold CV results on CICMalDroid 2020 development set. 87

4.8 Multiclass holdout results on CICMalDroid 2020 (five classes). $\star$ denotes proposed methods. 87

4.9 Per-class F1-scores for multiclass holdout evaluation on CICMalDroid 2020. 89

4.10 Feature reduction summary across datasets and classifiers. Consensus count = features selected in at least 2 of 5 cross-validation folds. 90

4.11 Top 20 most important features by Gini impurity decrease from the final Random Forest model (116 BWOA-selected features). Feature importance scores sum to 1.0 across all 116 selected features. 92

4.12 Feature stability tiers by cross-fold BWOA selection frequency across five independent optimisation runs. Features classified as Gold, Silver, or Bronze based on the number of folds in which they appear in the consensus subset. 94

4.13 Comparison of binary malware detection results on CICMalDroid 2020 with related work 98

4.14 Comparison of five-class malware categorisation results on CICMalDroid 2020 with related work 98

General Introduction

Context and Motivation

Android remains the most widely used mobile operating system in the world, supporting an ecosystem of applications that spans communication, banking, education, and e-commerce. This dominance, combined with the openness of the Android distribution model, has made the platform a primary target for malicious software. Trojans, spyware, ransomware, adware, worms, and rootkits continue to evolve in sophistication, exploiting both system vulnerabilities and user trust to compromise devices, exfiltrate data, and cause financial harm.

Machine learning has become the dominant paradigm for Android malware detection, replacing purely signature-based approaches that cannot keep pace with the rate at which new malware variants are produced. However, ML-based detectors are typically trained on very large feature spaces extracted from permissions, API calls, intents, network traffic, or system call traces. These high-dimensional representations introduce a number of practical problems: increased computational and memory overhead, reduced model interpretability, and a higher risk of overfitting to noisy or redundant features. In resource-constrained mobile environments, and in security contexts where analysts must be able to justify a detection decision, these problems are not merely theoretical inconveniences they directly limit deployability.

Feature selection has long been proposed as a remedy for the curse of dimensionality, but the existing literature on Android malware detection shows that feature selection practices remain inconsistent and often underutilised. A systematic review of 88 hybrid detection studies found that fewer than half applied any feature selection at all, and that meta-heuristic wrapper methods despite their demonstrated success in other high-dimensional domains were essentially absent from the field. This gap between the maturity of feature selection research in machine learning generally and its limited adoption in Android security specifically constitutes the starting point of this thesis.

Research Goals

This thesis aims to design, implement, and rigorously evaluate a feature selection framework that addresses the dimensionality, redundancy, and interpretability challenges of Android malware detection without sacrificing detection performance. More specifically, the work pursues the following goals:

- To design a hybrid feature selection architecture that combines the computational efficiency of filter methods with the performance benefits of a wrapper-based metaheuristic search, making global feature subset optimisation tractable on high-dimensional malware datasets.
- To adapt the Whale Optimisation Algorithm to the binary feature selection problem, and to evaluate its effectiveness against the dimensionality constraints typical of Android malware datasets.
- To design optimisation objectives that reflect the asymmetric cost structure of malware detection, where a missed malware sample is more harmful than a false alarm.
- To empirically validate the proposed framework on both binary detection and multiclass malware categorisation tasks, across independent datasets, in order to assess its generalisation capability beyond a single benchmark.
- To analyse the stability and security interpretability of the selected feature subsets, so that the contribution extends beyond raw performance metrics toward actionable insight for Android security practitioners.

In pursuit of the goals described above, this thesis is guided by three formal research questions. The first asks whether the Binary Whale Optimisation Algorithm (BWOA), adapted for binary feature selection, can effectively identify a discriminative and compact feature subset from the high-dimensional dynamic syscall feature spaces characteristic of Android malware datasets (RQ1). The second asks whether a hybrid filter-wrapper architecture combining an ensemble of filter methods (Information Gain, Fisher Score, and Chi-Squared) with a BWOA wrapper search outperforms standalone filter-only feature selection methods and full-feature baselines in terms of classification accuracy, malware recall, and feature reduction ratio (RQ2). The third asks whether the feature subsets selected by the proposed hybrid framework generalise across structurally distinct Android malware datasets with different feature representations, confirming that the identified behavioural indicators are dataset-independent (RQ3). These three questions correspond respectively to the evaluation of the BWOA's wrapper search effectiveness (Chapter 4, Sections 4.5–4.6), the comparative evaluation against baselines (Section 4.4.2 and Tables 4.4–4.8), and the cross-dataset generalisation analysis (Sections 4.5.5 and 4.8.3).

Contributions

In pursuit of these goals, this thesis makes the following contributions:

- A hybrid two-stage feature selection framework that fuses three filter methods Information Gain, Fisher Score, and Chi-Squared with a Binary Whale Optimisation Algorithm wrapper search guided by Random Forest and XGBoost fitness evaluators.
- Task-specific fitness functions: an F2-score objective for binary malware detection that explicitly prioritises malware recall, and an F1-macro objective for multiclass categorisation that balances performance across malware families.
- A leakage-free, nested cross-validation evaluation protocol that ensures all feature selection and model training operations are confined strictly to training data.
- Empirical validation on CICMalDroid 2020 and on the structurally distinct TUANDROMD dataset, demonstrating that the proposed BWOA-RF configuration matches or exceeds full-feature baselines while reducing the feature space by 56.8% to 81.5% depending on the dataset and task.
- A feature stability and importance analysis that identifies a compact, security-meaningful set of behavioural indicators including getDeviceId, mprotect, and writev consistently selected across independent optimisation runs.

Structure

The remainder of this thesis is organised as follows. Chapter 1 introduces the Android ecosystem, its security model, and the taxonomy of Android malware, before reviewing the main families of detection approaches. Chapter 2 examines feature selection techniques in depth, covering filter, wrapper, and embedded methods as well as metaheuristic optimisation, and situates the proposed work within the existing literature to identify the research gap. Chapter 3 presents the proposed hybrid filter-BWOA framework, detailing the preprocessing pipeline, the binary adaptation of the Whale Optimisation Algorithm, and the fitness evaluation strategy. Chapter 4 reports the experimental results on both datasets and classification tasks, including comparisons with related work and an analysis of the threats to validity. The General Conclusion closes the thesis with a summary of contributions, concluding remarks, and directions for future research.

Chapter 1

Android Malware Detection Background

1.1 Introduction

Android has come a long way since it was first introduced, and today it is one of the most popular mobile platforms in the world. Its open-source architecture and its ability to support a wide range of devices have contributed to its growth in different sectors such as communication, education, banking, and e-commerce (Chowdhury et al.,2021)[11]. But rapid growth has also increased security risks, and Android devices are now a prime target for cyber threats. Malware is one of the most severe threats to user privacy and system integrity. Malware is malicious software that can be installed on devices without permission. The malware exploits system vulnerabilities and risks sensitive data (Chowdhury et al.,2021)[11]. Earlier versions of the Android operating system were not as secure and were more vulnerable to malicious apps. Many apps had fewer restrictions and access to sensitive resources, increasing the chances of data leakage and illegal activities. Over the years, Android has addressed these problems with a number of security improvements, including application sandboxing and permission-based access control (Jassim Kassir, 2024)[20]. These mechanisms are useful to manage the interaction between applications and to protect the user data through access controls to sensitive resources (e.g., contacts, location, and storage). However, the malware authors are also developing the malware to bypass these security checks (Chowdhury et al.,2021)[11]. Traditional security measures are therefore not sufficient to face the challenge of today. The need for more sophisticated and smart techniques, especially machine learning-based techniques, to improve the detection and prevention of Android malware is increasing. Machine learning models are effective tools for detecting known and unknown threats (Jassim Kassir, 2024) [20] because they can analyze a large amount of data and find hidden patterns.

1.2 The Android Operating System

1.2.1 Architecture

As shown in Figure 1, Android architecture is a software stack that runs apps in a sandboxed environment and is mainly for smartphones. A custom Android embedded Linux system gives access to the phone's hardware, which is the base of the system. (Liu et al., 2020)[23].

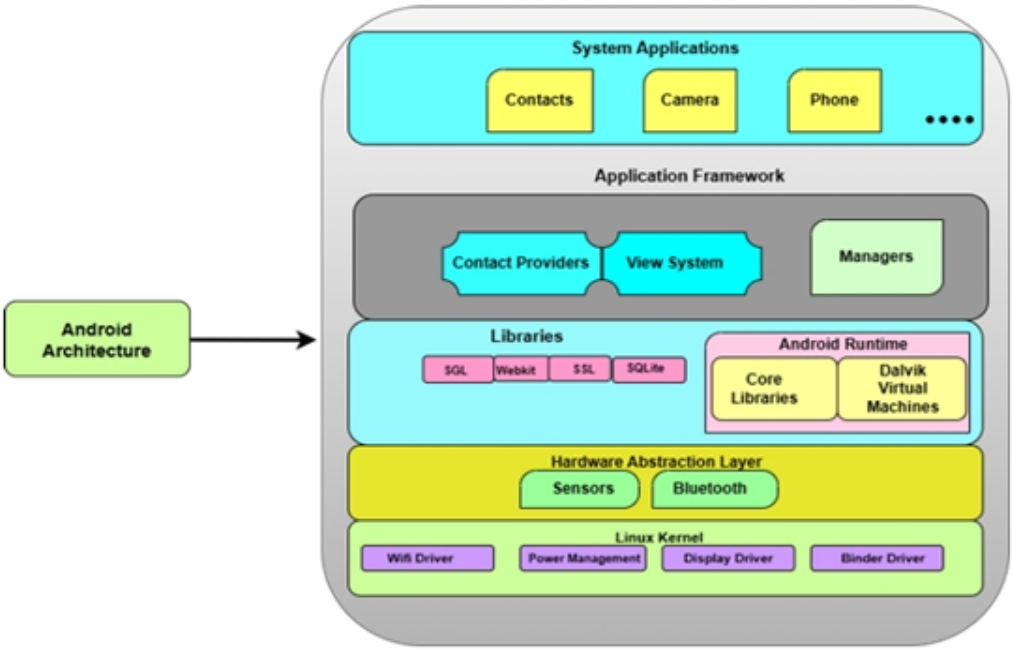


Figure 1.1: Android Architecture

This section gives a quick overview of how an Android app is built. An Android Application Package (APK) contains a file downloaded from the App Store, android manifest.xml, executable code, class resources, and a Dalvik bytecode file (Haq et al.,2022)[16]. The Android Manifest file lists the content providers, services, broadcast receivers, activities, and implementation classes that are necessary parts of an Android app. The Android Manifest file is the main file that sets up the app. The Android manifest file has API authorization permissions in it so that other apps can talk to it too. The manifest file also has intent filters in it.

Linux Kernel

The Android architecture is built on top of the Linux kernel. The kernel is like a link between the hardware and the software. It gives the system services. It takes care of memory, drivers (Wi-Fi, display, power), processes, and system security. Every Android app has

its own user ID (UID) and is separate from other apps. This separation also makes things safer because the apps can't talk to each other directly. But attackers may be able to use holes in the kernel to get more access and break into the system (Jassim Kassir, 2024)[20].

Layer of Abstraction for Hardware (HAL)

The Hardware Abstraction Layer (HAL) is a layer that sits between the operating system and the hardware of the device. This layer gives the hardware an abstract interface so that the system can use the hardware parts without knowing how they work. System calls let applications use services like the GPS or the microphone through the HAL. But mistakes in the HAL can make security holes that let sensitive information leak or let people do things they shouldn't (Chowdhury et al.,2021)[11].

Android Runtime and Libraries We have native libraries and runtime at the top of the HAL: The libraries (SQLite, SSL, and WebKit) provide important features like databases, security, and rendering web pages. The Android Runtime (ART) lets you run apps and manage memory. These parts are very important for the system to work, but they can also be attacked if they are weak (Jassim Kassir, 2024)[20].

Framework for applications

The Application Framework connects the operating system and applications. It has content providers, a view system, and managers that let developers make apps that can safely and well work with the system. Take note that using the APIs incorrectly or giving too many permissions can create a security hole (Chowdhury et al.,2021)[11].

Layer of Applications

You can find the system and user apps, like contacts, the camera, and the phone, in the applications layer. These apps need the framework services to work. Each application runs in its own "sandbox" environment, which limits access to system resources and other applications. This makes the system more secure (Jassim & Kassir, 2024)[20].

1.2.2 Application structure

Android apps are sent out as APKs, which are binary files. An APK has everything you need to run the binary. The executable bytecode (DEX files), resources (like images and layouts), and configuration files that tell the application how to work and what its structure is. The AndroidManifest file is the main configuration file for an APK.xml. This is the application's main configuration file. The specification of application components (activities, broadcast receivers, services), permissions, and system interactions. Here is an example of an Android Manifest file: Figure 2: Example of a permission request in an Android application from author (Mirabet Herranz, 2024)[18].

In this case, the app clearly says that it needs SEND_SMS permission to send SMS mes-

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.awesomeapp">
<uses-permission android:name="android.permission.SEND_SMS"/>
<application ...>
...
</application>
</manifest>
```

Figure 1.2: Example of a permission request in an Android application from author (Mirabet Herranz, 2024)[18]

sages. This is a very important example of how Android apps must say what resources they need access to before they can get to any secure system resources. The Android Manifest also lists intent filters that explain how app parts can work with other apps and services, as well as how apps can work with other apps through clearly defined interfaces.

Android apps run in a safe area. Every app gets its own Linux user ID. Applications can't read each other's data unless they have permission, and they can't see each other. This makes the system much safer because it limits the damage that a bad app can do.

1.2.3 security model

The Android security model protects user data and ensures stable system operation through two main mechanisms: application sandboxing and a permission-based access control model. Each application runs in isolation within its own sandbox, meaning it can only access its own resources unless explicitly allowed otherwise. This limits the impact of malicious or vulnerable apps on the rest of the system.

The permission-based access control model regulates access to sensitive system resources. Applications must declare the permissions they require in the AndroidManifest.xml file. These permissions are classified into normal permissions, which are automatically granted, and dangerous permissions, which require explicit user approval at runtime. Sensitive resources such as contacts, location, and SMS fall under dangerous permissions due to their privacy implications.

During installation and runtime, Android prompts the user to accept or deny requested permissions, and users can later modify these decisions through system settings. In addition to permissions, Android employs other security mechanisms, including application signing to verify the origin of apps, secure inter-process communication (IPC) to control data exchange between applications, and regular security updates to patch vulnerabilities. Together, these mechanisms provide a layered defense against common security threats (Jassim Kassir, 2024; Chowdhury et al., 2021 [20] [11]).

1.2.4 Application Distribution and Installation Mechanisms

Most of the time, Android apps are available through official stores like the Google Play Store and third-party app markets. The process of distributing apps is an important part of the Android security model as a whole.

When an app is installed, the system checks for declared permissions and asks the user to give permission if necessary. In the most recent versions of Android, sensitive permissions are asked for while the app is running. This is meant to give users more control and awareness.

But apps that come from third-party sources are a security risk. Third-party libraries may not use the same security checks as official sources, which means that harmful apps can be spread. Malware developers take advantage of this model by repackaging real apps with bad code and sharing them through third-party sources . When users download these kinds of apps, they get infected because they don't know they're installing harmful apps.

Also, installation methods usually rely on APK verification and application signing, which offers some level of security. But attackers are always coming up with new ways to get around security measures. So, by learning how applications are distributed and installed, we can find possible ways for malware to spread and improve our ability to find it (Chowdhury et al.,2021)[11].

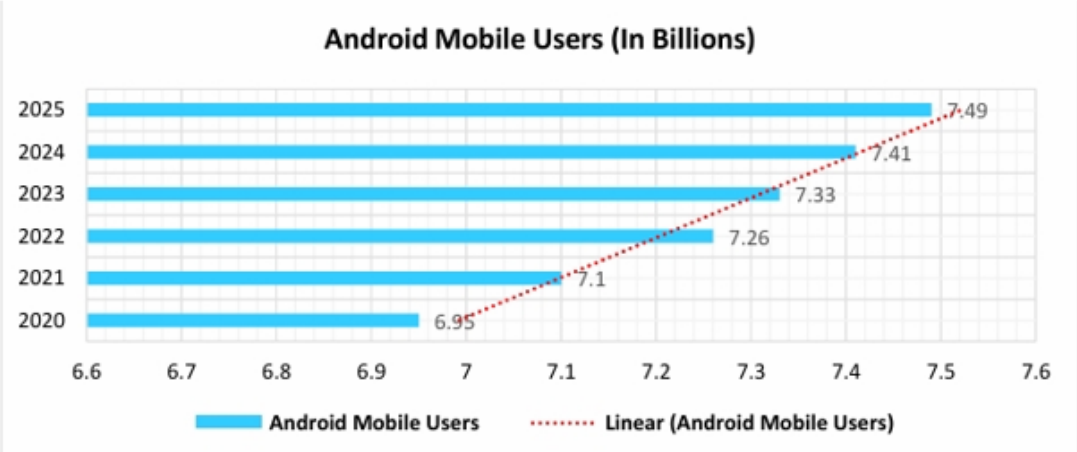


Figure 1.3: Projected global mobile user growth from 2020 to 2025

1.3 Android Malware

1.3.1 Definitions and Taxonomy

Android malware is software that gets into mobile devices, takes advantage of system weaknesses, and uses user data and system features in ways that are not allowed. As the Android ecosystem grows quickly, malware has changed into many different types, each with its own goals and behaviors (Chowdhury et al.,2021)[11]. It is very important to know about the different kinds of Android malware in order to make detection methods, especially when using machine learning models.

Trojans: are the most common type of malware for Android. Trojans look like legitimate apps to users, but they are actually not. The apps are installed and run harmful tasks in the background without the users' knowledge. Malicious tasks could include stealing sensitive information like passwords and leaving a back door open for attacks. Trojan viruses use social engineering a lot to fool people (Chowdhury et al.,2021)[11].

Spyware: is a type of malware that collects private information and tracks what the user does without their permission. It is used to keep track of what the user does, as well as to log keystrokes, record calls, and get messages and location information. Spyware is a privacy threat because it works in secret and sends data to attackers all the time (Jassim Kassir, 2024)[20]. It can be found in apps that seem normal. (Jassim Kassir, 2024)[20].

Ransomware:is a type of malware that locks a device or encrypts user data and stops the device or user from using it until a ransom is paid. You can do this by locking the screen or encrypting the files. This kind of malware is popular because it has a direct effect on finances. Paying the ransom does not mean that the data will be returned (Chowdhury et al.,2021)[11]. Paying the ransom does not guarantee the return of the data (Chowdhury et al.,2021)[11].

Adware: is a kind of malware that can make ads show up on a device that you don't want to see. It isn't always bad, but it can have a big effect on how well a device works and how well a user has a good time. In some cases, it can send the user to bad websites or put strange types of malware on the device, which could be a security risk (Jassim Kassir, 2024)[11].

Worm: is a type of malware that can spread on its own across many devices and networks without any action from the user. Worms usually copy themselves by taking advantage of weaknesses in the system, which can cause the system to crash or the network to slow down. A worm does not need to be installed by a user once it takes advantage of a weakness, unlike a trojan (Chowdhury et al.,2021)[11].

Rootkits: are a different kind of advanced malware that secretly get root access to a

device. Rootkits change the system so they can hide and take over.

Table 1.1: Types of Malware and Their Characteristics

Malware Type	**Main Objective**	**Working Method**	**Impact**
Trojans	Data theft	Trick users into installing malicious software	High
Spyware	User surveillance	Collects data without user consent	High
Ransomware	Financial gain (ransom)	Blocks access until payment is made	Very High
Adware	Advertising profit	Displays intrusive advertisements	Medium
Worms	Fast propagation	Spreads automatically across networks	High
Rootkits	Full system control	Operates stealthily within the system	Very High

Table 1: shows how the main types of Android malware are different from each other in terms of their properties, goals, and effects.

1.3.2 Malware Infection Vectors and Propagation Techniques

Infection vectors are the first places where malware can get into a device. In the context of Android, there are a number of common vectors, such as: malicious apps (repackaging): Attackers frequently repackage existing applications with malicious code and disseminate them through third-party app stores or alternative unofficial distribution channels. Users unknowingly install dirty apps that give them permissions that let bad things happen (Chowdhury et al., 2021)[11]. Phishing and social engineering: Some attack methods use friendly engineering techniques, like fake emails, messages, or websites, to trick people into downloading bad apps or giving away sensitive information. Attackers take advantage of users' trust and use friendly engineering tricks to get around security warnings (Jassim Kassir, 2024)[20].

Downloads that happen while you drive When a user visits a hacked or malicious website, attackers can automatically download malware onto their device. These downloads can use flaws in the browser or system to infect devices without the user having to give their full consent or do anything. Attacks that use SMS and messaging: Malicious links sent through SMS, WhatsApp, or strange messaging apps can send malware. Attackers can automatically install malware on a user's device when they visit a hacked or bad website. These downloads can use weaknesses in the browser or system to spread malware to devices without the user having to do anything or give their permission. Attacks using

SMS and messaging: Malicious links sent through SMS, WhatsApp, or other strange messaging platforms can spread malware. Users who click on these links may be sent to a page where they can download malware or be asked to install malware. Exploitation of System Vulnerabilities: Attackers take advantage of well-known unpatched security holes in the Android operating system or on device applications to inject malicious code and gain unauthorized access or control.

Malware Propagation:

Once the device is infected, the malware may use different methods to spread to other devices and networks. Some malware can make copies of itself and spread to other devices on the same network using WiFi, Bluetooth, or shared files. This is how worms act (Chowdhury et al.,2021)[11]. Some malware can make a device into a bot that is part of a botnet. This is a network of hacked devices that can be used for different types of attacks, such as sending spam or mining cryptocurrency (Mirabet Herranz, 2024)[18]. To obtain more control over the infected device, people use permission abuse and privilege escalation. Some types of malware can get root access by using privilege escalation techniques. Malware can talk to command and control servers to get instructions or send stolen data. This system could also be used to control things from afar.In business settings, malware can then move from one device to another over a local network (lateral movement).

Problems with finding malware: Malware detection becomes harder because there are so many ways for malware to spread and infect computers. Because malware can use encryption, polymorphism, and other more advanced obfuscation methods, these methods need more advanced ways to be found, such as machine learning. Using machine learning to find malware can then help find it better (Jassim Kassir, 2024)[20].

1.3.3 Evolution and Trends in Android Malware

Malware is software that is meant to hurt or disrupt the host system. Malware has its roots in the time when simple viruses were made just to spread and infect other computers without hurting the original host (Chowdhury et al.,2021)[11]. Malware has changed over the years from simple computer viruses to more complex types like worms, trojans, and spyware. Newer types of malware are made to take advantage of weak spots in systems and gather private information that can be used for other purposes. Malware is a constant threat to systems and infrastructure that protect information. It also includes malicious software like ransomware, spyware, and adware (Jassim Kassir, 2024)[20].

Recently, malware has gotten more sophisticated and advanced. The different types of malware are more focused on making money. Since the rise of ransomware, adware, and botnets, malware has become more advanced and focused on making money for the attacker. For instance, ransomware can encrypt the user's data, and the user has to pay

a ransom to get it back. Botnets use a lot of infected devices to launch big attacks, like distributed botnet attacks, and to mine cryptocurrency (Mirabet Herranz, 2024)[18].

Also, the rise in the use of mobile devices, especially those that run on Android, has led to the creation of new types of malware that target the mobile ecosystem. Mobile malware variants exploit application permissions, system vulnerabilities, and user malicious behavior to obtain unauthorized access and control over the device (Mirabet Herranz, 2024)[18].

The complexity and variety of modern malware make signature-based detection methods useless. As a result, there is a growing need for new machine learning-based detection methods that can find new, previously unknown threats more accurately and quickly by looking at behavior patterns (Jassim Kassir, 2024)[20]. Figure 1.4 shows that malware can be divided into different groups based on what they do and why they do it. These groups include viruses, Trojans, spyware,and ransomware.

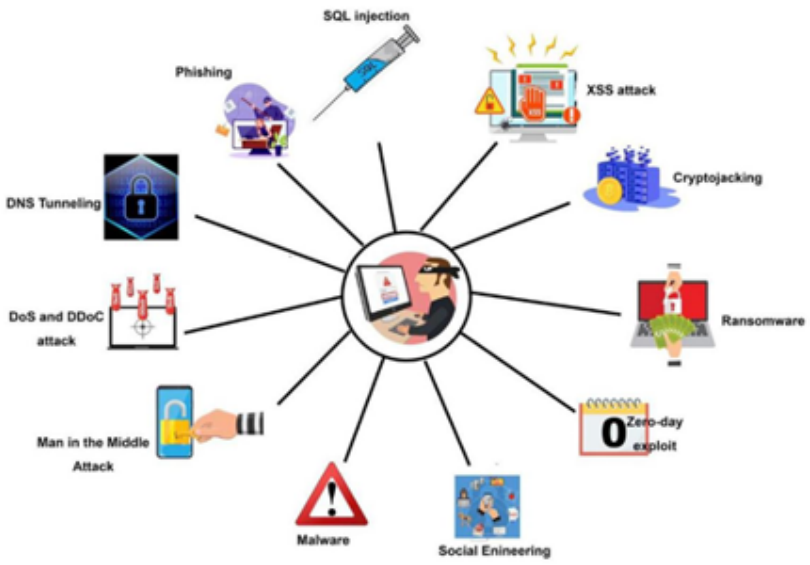


Figure 1.4: Graphical representation of the distinct types of malware. from (Mirabet Herranz, 2024) [18]

1.4 Android Malware Analysis Techniques

Malware analysis is the process of figuring out what kind of malware is on a computer. There are three kinds of malware analysis: static, dynamic, and hybrid. Static analysis is the process of looking through a program's source code without actually running it to see if there is any bad code in it. Dynamic analysis is the process of running an application in a test environment and keeping an eye out for malware. The hybrid analysis is obtained through static and dynamic analysis (Muttoo Badhani, 2021)[29].

1.4.1 Static analysis

Researchers like static analysis the most because it can be set up and used quickly, and it works to some extent. In this method, the application source code is examined but not run on a real or simulated device (Bakour et al., 2019)[5]. When you upload an Android app to the app store, you use a file format called APK. APKtool decompiles APK files, and then it decodes the AndroidManifest.xml file to find out the package names, environment, intent features, components, permissions, and more (Z.Wang et al., 2020)[40].

1.4.2 Dynamic analysis

This method looks at how well the programs on a smartphone work after they are installed to see if they are safe or dangerous. In the literature on dynamic analysis, a variety of challenges have been studied. This method uses anomaly- or behavior-based detection tools to keep an eye on what the smartphone is doing after the app is installed. These methods look at how the smartphone works to see if the app that was installed is harmful or not (Talal et al., 2019)[36].

1.4.3 Hybrid analysis

Researchers started using a method called ”hybrid” to look at both static and dynamic analysis to solve problems with both (Kumar et al., 2019). To get the static attributes, the program's source code is reverse-engineered over and over again. Then, the programs are run on a different system, like an emulator or a real device, to get the dynamic properties. People think that this method of analysis is the most complete and revealing, even though it has problems with implementation and overhead (Bakour et al., 2019)[5].

1.5 Android Malware Detection Approaches

For a long time, researchers have used traditional methods of finding malware as the first line of defense against it. Traditional malware detection methods look for threats in a system by looking for patterns that are already known and following rules that have already been set. These methods helped find early examples of Android malware and keep user data safe on Android (Chowdhury et al., 2021)[11]. Old ways of finding malware don't work as well as new ones because malware is always changing. This review is important because it tells us what these methods do and what their good and bad points are.

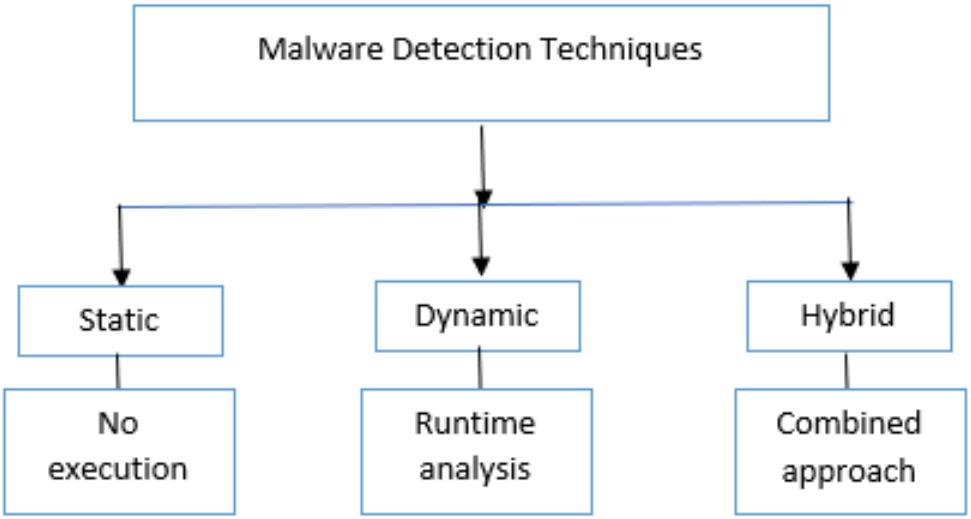


Figure 1.5: : Classification of malware detection techniques based on analysis type

1.5.1 Signature-based detection

One of the oldest and most common ways to find malware is by using detection based on signatures. It works by looking at applications or files and comparing them to a database of malware signatures, which are strange patterns that stand for a certain kind of malware. If the application has a signature that matches, it is malware. It is a fast and easy way to find things, and it works well in real-time detection systems (Jassim Kassir, 2024)[20]. But this method has a lot of problems because it can't find malware that is new or hidden, which is also called a "zero-day" attack. Also, modern malware often hides its signature by using code obfuscation and polymorphism to avoid being found. This means that the signature-based detection method needs to update its database a lot (Chowdhury et al., 2021)[11].

1.5.2 Heuristic and rule-based detection

Heuristic detection compares code logic against a predefined set of "suspicious" criteria, while signature-based techniques are more traditional and compare files for an exact match. It examines the behaviour of an application, such as making unauthorised changes to files, making strange network connections, or making unusual system calls, and looks for characteristics common to malicious software. By focusing on these indicators instead of fixed identities, heuristics can detect "zero-day" threats and polymorphic malware that are not yet in signature databases (Jassim Kassir, 2024 [20]).

However, such a pre-emptive inspection has its technological compromises. Continuous monitoring of system functions can consume significant processing resources, which

may affect system performance and latency. Moreover, heuristic engines can generate false positives by marking safe software as a security risk due to complex or unusual actions performed by a legitimate application (Chowdhury et al., 2021[11]).

1.5.3 Machine learning–based detection

Machine learning–based detection has become one of the most effective approaches for identifying modern malware. Unlike signature-based methods, machine learning techniques do not rely on predefined malware signatures. Instead, they learn patterns and relationships from datasets containing both benign and malicious applications. After training, the model can classify new applications automatically based on extracted features (Ucci et al., 2019 [38]).

In Android malware detection, machine learning models are commonly trained using static, dynamic, or hybrid features. Static features may include permissions, API calls, intents, opcodes, and application metadata extracted from APK files without executing the application. Dynamic features are collected while the application is running and may include system calls, network traffic, CPU usage, and runtime behavior (Mahindru and Sangal, 2021[25]) .

Several machine learning algorithms have been widely applied in malware detection research, including Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Naïve Bayes, and Extreme Gradient Boosting (XGBoost). Ensemble learning models such as Random Forest and XGBoost often achieve strong classification performance due to their ability to manage high-dimensional data and reduce overfitting (Alzaylaee et al., 2020 [3]).

One major advantage of machine learning–based detection is its ability to detect previously unseen malware variants and zero-day attacks by generalizing from learned patterns instead of depending on known signatures. Furthermore, machine learning systems can adapt to evolving malware behaviors by retraining models with updated datasets (Ucci et al., 2019 [38]).

However, machine learning approaches also face several challenges. Their effectiveness depends strongly on the quality of the extracted features and training datasets. Large feature spaces may increase computational complexity and training time, while imbalanced datasets can reduce classification reliability. Feature selection techniques are therefore commonly applied to reduce dimensionality, improve model performance, and lower computational cost (Mahindru and Sangal, 2021 [25]).

1.6 Feature Types for Android Malware Detection

Feature extraction: Feature extraction is a key part of the Android malware detection system, and the features that were taken out affect how well any model works. Features are things about an Android app that can be measured and used by machine learning models to tell the difference between good and bad behavior. When looking for Android malware, features are taken from different parts of the system, such as the code for the apps, how they work with the system, and how they act while they are running. There are many different kinds of these features, like API calls, permissions, system calls, network traffic, and opcodes. (Jassim Kassir, 2024; Chowdhury et al., 2021)[20] [11]. Features That Depend on Permissions

Features that are based on permissions: The most common way to find Android malware is to look at permissions. Each Android app has an AndroidManifest.xml file that lists the permissions it needs. These basically say how much access an app has to a device's sensitive resources, such as the following: contacts, text messages, location, camera, and storage Most of the time, bad apps ask for more permissions than safe apps do. If a simple calculator app can get to the SMS or contacts feature, for example, it might be suspicious. (Chowdhury et al., 2021)[11]

Characteristics Derived from API Calls API calls are function calls that are made at the application level to the Android system and libraries that other people have made. They show you exactly how the app works and what it can do. As an example: Getting to user data Sending text messages Making a connection to a network Reading documents Sometimes, malware will use certain API calls to do bad things. So, you can use the frequency and patterns of these API calls to find machine learning models that are acting strangely. API calls are better for finding malware because they give more information than permissions. But you need to take API calls out of the code, and ways to hide code might change them. (Jassim Kassir, 2024)[20]

Features Based on System Calls: A call to the operating system kernel is a system call. They show exactly how the app works in detail. For instance: Read and write files Making a process Setting aside memory Talking to each other over a network System calls are great for dynamic analysis because they show you how the application is working right now. Malware often makes system calls in ways that are not normal. Machine learning models can use this information to find malware again. But you can only see system call information if you run the app in a sandbox, which takes a lot of processing power. is simple and works well because it doesn't require the application to run. But it has its own problems because some bad apps can get around detection by only asking for a few permissions (Chowdhury et al., 2021)[11].

Features of Network Traffic: Network traffic features show how apps talk to servers that aren't their own. Different types of malware apps connect to servers in other countries to steal data or get instructions from the attacker. Some features of network traffic are: IP addresses, URLs, how often people talk to each other, and patterns of data transmission For example, a bad app can send private information to a command and control server that is far away. Network-based analysis is a good way to find behaviors like data exfiltration and remote control, but it takes a long time to watch the runtime behavior, and encryption can make it harder, which may lead to delays in identifying malicious activities and responding to potential threats. (Jassim Kassir, 2024)[20].

Characteristics of Opcode Sequences Opcode sequences are the basic commands that come from the application's compiled code, which is kept in DEX files. These are the actual steps that the program takes. We can find patterns that aren't easy to see at a higher level of abstraction by taking opcode sequences out of the code. Malicious software often has opcode sequences that aren't found in safe software. ML and DL models can learn these patterns to make the detection more precise. You have to break down the application code to get opcode sequences, which takes a long time and is hard to do. Obfuscation methods can also change the opcode sequences so that they can't be found. (Chowdhury et al., 2021)[11] Talk about why quality is important for features. The features you choose affect how well malware detection systems work. The characteristics should be: Important for the job of finding Not extra helpful Strong against efforts to get around it The models might be wrong, the false positive rate might be high, and the detection performance might be bad if the features aren't right. A common way to improve performance is to mix and match different types of features, such as permissions, API calls, and dynamic features. This is called hybrid feature extraction. Also, feature selection methods are used to pick the most important features and cut down on the number of dimensions. This results in models that exhibit enhanced accuracy and functionality (Jassim Kassir, 2024; Chowdhury et al., 2021)[20] [11].

1.7 ML for Android Malware Detection

1.7.1 Machine Learning–Based Detection

ML techniques have been widely used for Android malware detection because they can learn patterns automatically from data, instead of manually creating signatures. In these methods, applications are usually represented by feature vectors obtained from static or dynamic analysis, such as permissions, API calls, intents, and system calls. 22 Classifiers like Support Vector Machines (SVM), Random Forest (RF), Decision Trees (DT), and k-Nearest Neighbors (KNN) are widely used for benign and malicious application classi-

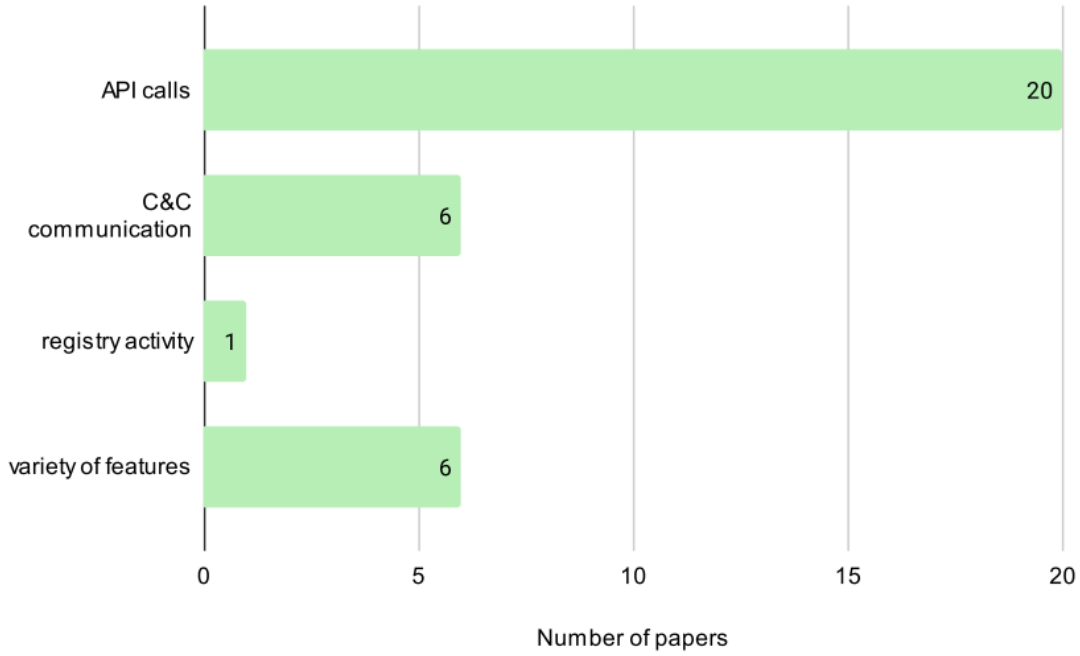


Figure 1.6: The most used features for malware detection in 20 studies

fication. The DREBIN system proposed by Arp et al. [4] is one of the most influential works in this area. DREBIN uses a large set of static features and a linear SVM classifier to achieve efficient and explainable malware detection. The effectiveness of ML models for Android malware detection using carefully engineered features has been demonstrated also in other studies features [35][12]).

1.7.2 Limitations of Traditional Machine Learning

Traditional ML-based approaches are effective but have some limitations. First, they heavily depend on manual feature engineering, a process that needs domain expertise and may fail to capture intricate patterns within the data. Second, these methods are often limited in high-dimensional feature spaces, where redundant and irrelevant features can degrade performance and cause overfitting [15].

Moreover, many classical algorithms assume linear relationships between features and class labels, hence limiting their ability to model the complex and non-linear behavior of modern malware. Moreover, their performance is highly sensitive to feature selection and preprocessing steps, making them less robust in real-world scenarios [39]

1.8 Conclusion

This chapter presented the fundamentals of the Android ecosystem, Android malware, and the main malware detection approaches used . It also reviewed the most common feature types and machine learning techniques applied in Android malware detection, while highlighting the challenges of high-dimensional and redundant feature spaces. These challenges motivate the study of feature selection methods, which are discussed in the next chapter.

Chapter 2

Feature Selection Techniques for Android Malware Detection and Literature Review

2.1 Introduction

When several studies on the importance of feature selection were published in 1997 by several researchers, such as Blum and Langley, and Kohavi and John, the number of features used in most applications at that time was relatively small and did not exceed 40 . [15] . However, in the following years, data increases exponentially in both complexity and size. This rapid growth of data presents challenges for effective data management and processing.

The application of high-dimensional data in machine learning and data mining algorithms gives rise to a critical issue known as the curse of dimensionality. It refers that data become sparser in high-dimensional space, adversely affecting algorithms designed for low-dimensional space [17]. Furthermore, the high dimensionality of data is frequently associated with redundant, noisy, irrelevant features, which increase the risk of overfitting and degrade the overall performance of machine learning models. [39]

These challenges affect numerous domains that handle large-scale data, including malware detection, where applications can be represented by a large number of features, including API calls, permissions, and intents. It is not necessary to utilize all these features for malware detection, as a significant portion of them introduces noise into the learning process.

To efficiently manage a very high-dimensionality datasets for the sake of making it available and automatically extract insights, knowledge and hidden patterns, it is a necessity to apply dimensionality reduction tools and techniques(see Figure 3.7). Among these,The

feature selection process has been instrumental in mitigating these challenges.

feature selection enhances classification accuracy, reduces computational costs, and improves model interpretability. For instance, it enables the identification of critical features, such as sensitive permissions ,which are strongly correlated with malicious behavior.

In this chapter, we examine feature selection techniques as a pivotal step in machine learning. We outline the primary categories of feature selection methods and analyze their respective advantages and limitations. Moreover, this chapter provides an overview of existing work in the field and highlights the relevance of feature selection.

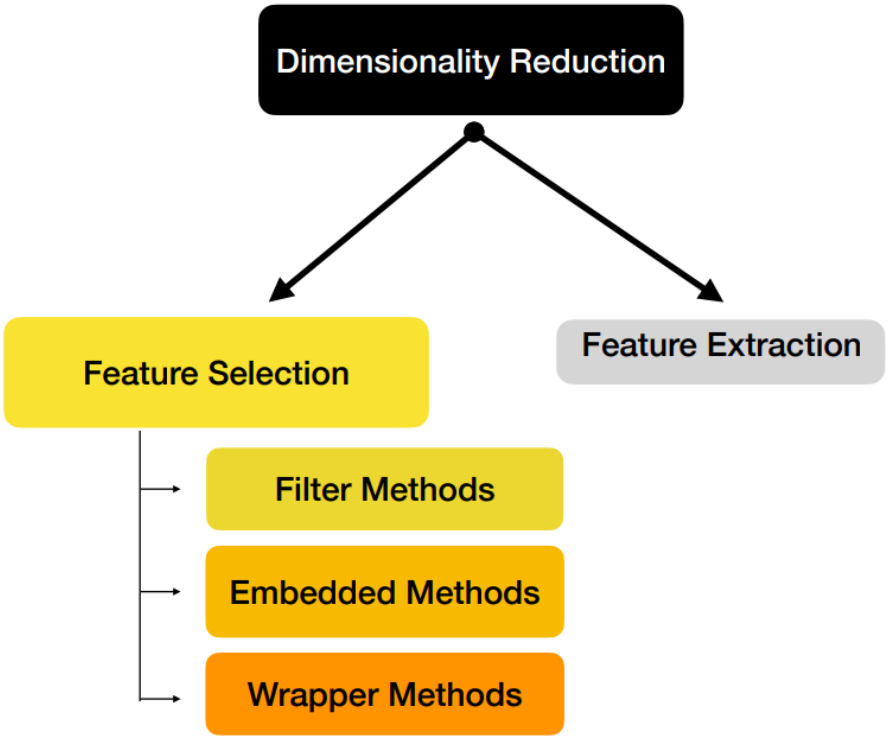


Figure 2.1: Dimensionality Reduction Techniques Source: [1]

2.2 Feature Selection Fundamentals

2.2.1 Definition and Concepts

Features selection or the process of selecting a subset of relevant features and removing the irrelevant ,noisy and redundant ones(see Figure 1.1) 2.2.is a preprocessing step in machine learning meant to deal with large-scale data problems, intending to obtain the most predictive subset without any changes in the features[1].

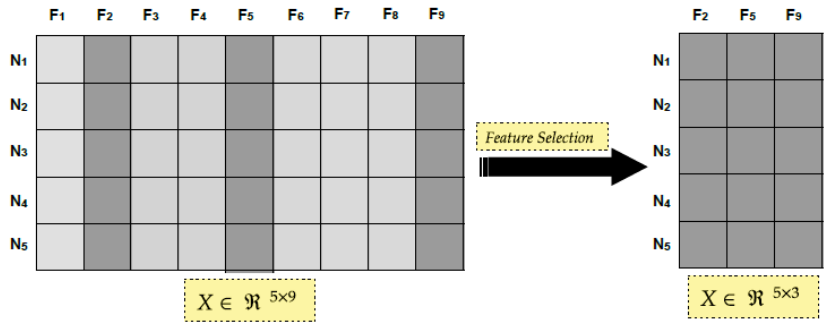


Figure 2.2: Reducing the feature space $X \in R^{5\times9}$ by selecting the most informative features while maintaining the physical meaning of the original features. The new feature space is $X \in R^{5\times3}$. Source: [1]

2.2.2 Feature Relevance

The main concept of feature selection is selecting only the most relevant features, which are considered relevant if they provide information about the target (weak relevance)[22] or if changing the value of a relevant feature affects the classification given by the target concept[7].(strong relevance)[22] In contrast to relevant features, irrelevant features do not provide any useful information. For a feature xi from the dataset, if the correlation between xi and the class label y is close to zero, then xi is an irrelevant feature and can be safely removed from the dataset.

- Strongly relevant: directly influence the output
- Weakly relevant: provide useful information in combination with other features

2.2.3 Feature Redundancy

The concept of redundant features helps distinguish weakly relevant features, identifying which can be removed and which may be important in the classification process according to the Markov blanket. A feature becomes unnecessary if its information is fully captured by a subset of other features (Figure 2). In conclusion, a feature becomes redundant if it is weakly relevant and its information is completely captured by a subset of other features (Yu and Liu, 2004a[42]).

2.2.4 Importance of Feature Selection

Feature selection plays a very important role by serving several purposes: **The facilitation of model interpretation.** In numerous domains, the interpretability of ML models has become an essential component, as the model's efficacy is insufficient to establish confi-

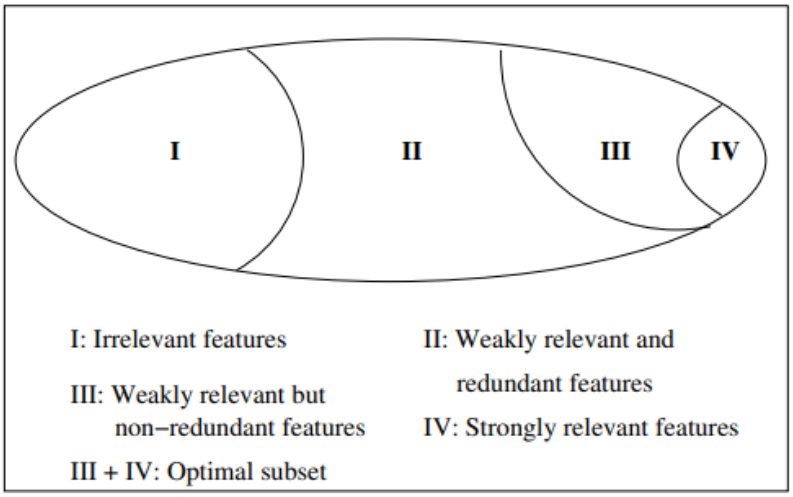


Figure 2.3: A view of feature relevance and redundancy. Source: [1]

dence in its decisions. In the majority of real-world applications, it is also essential to understand the rationale behind a specific decision or prediction, particularly in healthcare and financial services,malware detection, as well as in other strictly regulated domains, to utilize ML systems for high-stakes decisions that have a profound impact on both human lives and society (Rudin, 2018)[34]. This requirement for model interpretability exacerbates the demand for model interpretability(Rozi, 2025)[33]. Feature selection can significantly improve the interpretability of a model by reducing the feature space. This reduction in the feature space can assist in the visualization, comprehension, and identification of potential factors that may influence a model's ability to make specific decisions.

The reduction of resource requirements (short training time, small storagecapacity, etc.). In general, algorithms that are fed with reduced feature subsets result in a faster execution and a reduction in storage capacity.

Avoiding the curse of dimensionality. The curse of dimensionality is a fundamental motivation for feature selection, particularly when dealing with a large number of feature situations and a limited sample size. Reducing the amount of feature space significantly contributes to increase the accuracy of classification as a result of the effects of a finite sample size Jain and Chandrasekaran (1982).[10].

Avoiding the over-fitting problem, automatically leading to a better model generalization. The opportunity to make decisions based on noise is automatically reduced by reducing the quantity of redundant/noisy data. Consequently, the overfitting issue is mitigated.

Improving accuracy: less noise in data, which means an improved modeling accuracy. Machine learning algorithms may not always benefit from an abundance of data. The efficacy of learning algorithms can be negatively impacted by the presence of uninformative and noisy features. Consequently, the accuracy can be enhanced by first eliminat-

ing the unnecessary feature.

2.3 Categories of Feature Selection Methods

Feature selection algorithms are classified into supervised, semi-supervised, and unsupervised categories based on the presence of label information.Supervised feature selection methods typically fall into three categories: Filter, Wrapper, and Embedded approaches.

Filter: methods perform feature selection independently of any learning algorithm. They depend on the statistical relationship between each feature and the target class, measuring this relationship using mathematical forms like correlation and chi-square. This approach is advantageous in terms of speed and generalization ability.

Wrappers: evaluate features collectively rather than individually, like filters. They depend on learning algorithms to assess the performance of each generated subset. Consequently, this makes the learning process more complex and computationally costly. Nonetheless, this interaction with the learning model often results in better performance compared to filters.

•**Embedded:** This method can identify dependencies with lower computational complexity than wrapper methods, as feature selection is integrated and optimized during classifications.

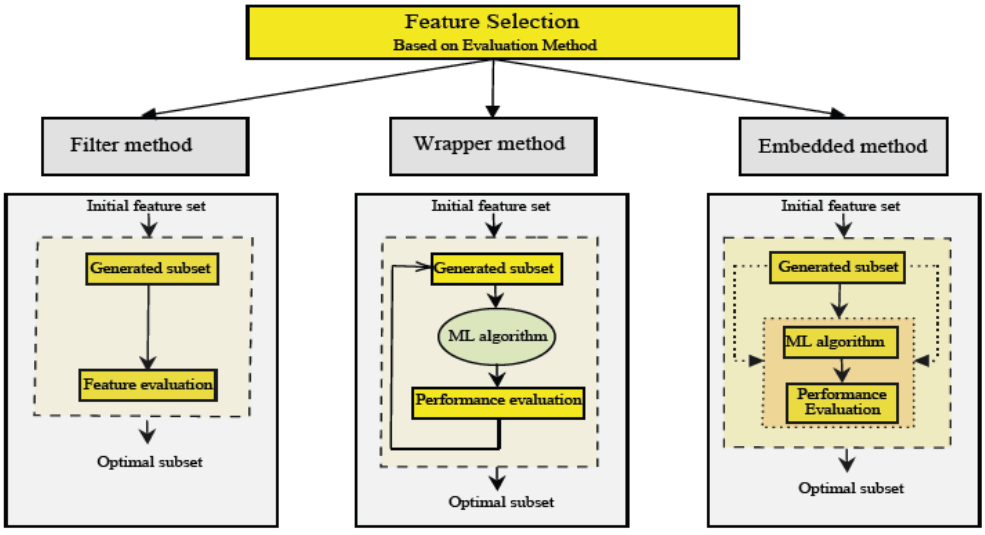


Figure 2.4: The general classification of feature selection methods according to their evaluation criteria. Source:[1]

2.3.1 Filter Methods

In Filter methods, a ranking criterion is applied to score variables, and those below a certain threshold are removed. The remaining features are then used as input to the classification algorithm. Filter methods assess the relevance of features using measures like distance, information, correlation, and consistency .In the filter method, feature selection needs to be performed only once, and then different classifiers can be evaluated due to its advantages of being fast, scalable, and independent of a learning algorithm.(Mwadulo, 2016[31]).

A disadvantage of filter techniques is their absence of interaction with the classifier, resulting in more generalized outcomes and reduced classification accuracy . Filter methods can be categorized into univariate and multivariate. Univariate filter methods ignore feature dependencies which can lead to selection of redundant features and worst classification performance when compared to other feature selection techniques . On the other hand, multivariate filter methods model feature dependencies independent of the classifier (Mwadulo, 2016[31]). Some of the most popular filter methods will be described in the following sub-subsections.

Information Gain

Information Gain (IG) is both an entropy-based evaluation method and a univariate feature selection method. IG allows one to understand how much information Each feature carries the target concept. However, information gain (IG) can also be defined using mutual information. In particular, information gain IG (F) is the reduction in the entropy that is achieved by learning a feature (F):

$$IG(F) = H(S) - \sum_i \frac{S_i}{S} H(S_i) \quad (2.1)$$

Where $H(S) = -\sum_{i=0}^{n}[p(i)\log p(i)]$ is the entropy of a given dataset S and $H(S_i)$

Is the entropy of the $i-ths$ subset produced by splitting S based on feature F. In feature selection context, Features can be ranked based on their Information Gain scores, where higher values indicate stronger discriminative power between classes (Quinlan, 1986 [32]).

Chi-Square

Chi-square (Chi-2) is a statistical test used to determine the independence of variables. (Jin et al. 2006[21]). For feature selection, the Chi-square method evaluates the dependency between each feature and its class label using the following formula:

$$x^2 = \sum \frac{(Y_0 - Y_E)^2}{Y_E}$$

where: Where Y_0 is the true value and Y_E is the expected value. When $Y_0$and Y_E are very close, it indx^2 value. The higher x^2value indicates that the feature is more dependent to the class label which means that it is useful at discriminating instances from different classes. Chi-square has limitations such as its sample size requirements and the difficulty of interpretation when there are many categories in the independent or dependent variables. McHugh(2013[26])

Mutual Information

Mutual information is the measure of the amount of one random variable has about the other [1] or the shared information between a feature and the class label . The most effective feature is the one that shares significant information with the target class, enabling it to efficiently differentiate members of one class from another. Mathematically, MI is defined as follows:

$$I(X,Y) = \sum_{i=1}^{n} \sum_{j=1}^{n} p(X(i), Y(j)) \cdot \log \left(\frac{p(X(i), Y(j))}{p(X(i)) \cdot p(Y(j))} \right) \tag{2.5}$$

MI is zero when X and Y are statistically independent ($p(X(i), Y(j)) = p(X(i)) \cdot p(Y(j))$).

2.3.2 Wrapper Methods

Rather than ranking each feature individually as in filters, the Wrapper approach assesses feature subsets using the classifier's prediction performance as a black box. It employs prediction performance as a criterion function to guide the search for the optimal feature subset and to identify features that optimize it.

In contrast to filters, this method considers the interactions between features, which significantly enhances performance. Additionally, wrappers necessitate a vast amount of computations due to the necessity of constructing numerous models from start during the search process. Exhaustive search methods are particularly computationally impractical for datasets that are significantly larger.

As a result, a variety of methods and algorithms, including sequential search and evolutionary algorithms, are further developed and introduced to accomplish good experimental performance with practical computational costs. Still, wrapper methods are more

computationally complex than Filter and Embedded methods due to their reliance on the induction algorithm [1].

Crucially, the efficacy of a Wrapper approach is fundamentally dependent on its underlying induction algorithm. Because the feature subset is evaluated based on the performance of this 'black box' classifier.

Sequential Forward Selection

Sequential forward selection (SFS) is a simple greedy algorithm used for feature selection. It starts with an empty set and adds features one by one, stopping when no further improvement is possible. SFS works effectively when the optimal subset of features is small. However, a limitation is that once a feature is added, it cannot be removed, which can sometimes lead to sub-optimal outcomes. This limitation stems from the nesting effect where the greedy nature of SFS prevents it from re-evaluating features once they are selected. Unlike these local search methods, Metaheuristic algorithms detailed later in this chapter overcome this by facilitating a global search of the feature space, allowing for the discovery of more complex feature combinations that greedy algorithms might overlook Below is a detailed explanation of the SFS process ([1]).

Sequential Backward Elimination

Unlike forward selection, Sequential Backward Elimination (SBS) begins with all features and progressively removes them one by one. In each step, the feature whose removal causes the greatest increase (or the smallest decrease) in the evaluation function is eliminated [1].

Recursive Feature Elimination

The RFE selection method involves a recursive backward elimination of predictors using a greedy optimization algorithm. This process ranks features based on their importance measures. It begins with the entire feature set, and at each step, the importance of each feature is evaluated. The least important feature is then removed. To improve efficiency, an alternative approach removes a group of features simultaneously. The recursive nature is essential because the importance of features can change significantly when evaluated within different subsets, especially for highly correlated features. The sequence in which features are eliminated during this process determines their ranking Akhiat [1].

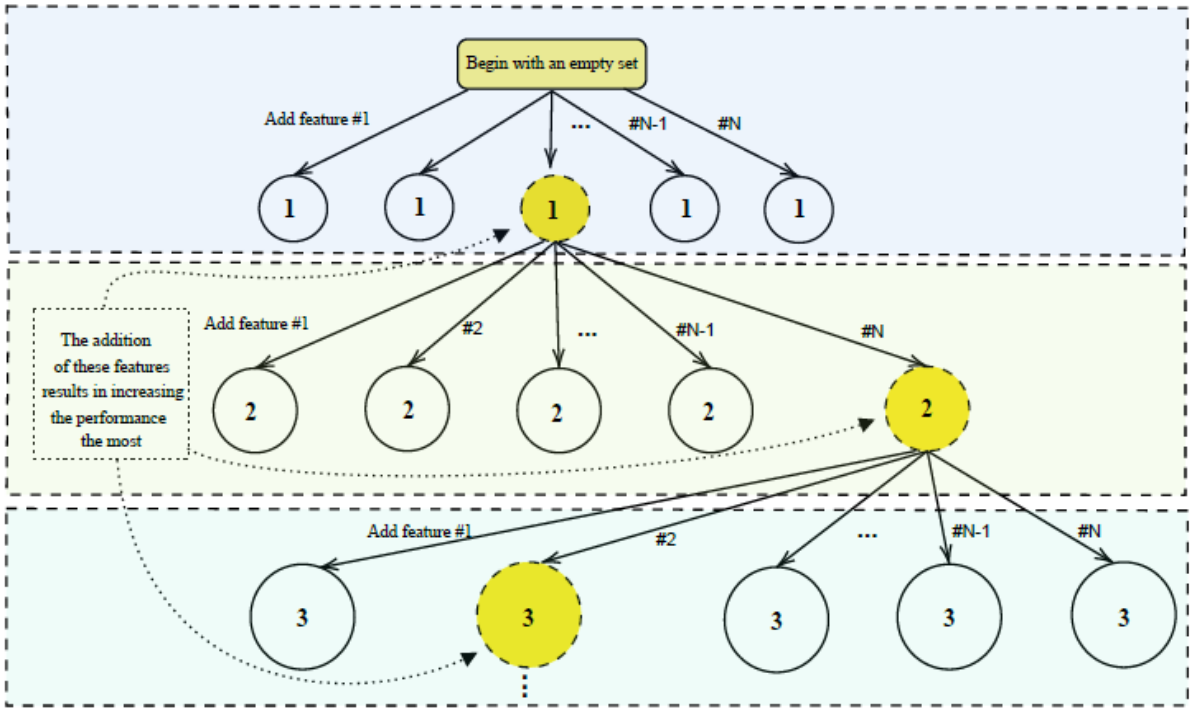


Figure 2.5: *The illustration shows the three steps of the SFS subset selection algorithm. It begins with an empty feature set and sequentially adds one feature at a time from the available pool. After evaluating classifier performance with each added feature, the algorithm selects the feature that results in the greatest increase in accuracy. This chosen feature is then added to the set and included in subsequent iterations. The process continues by trying to add another feature from the remaining $N - 1$ options, continuing until a stopping criterion is met Source:[1]*

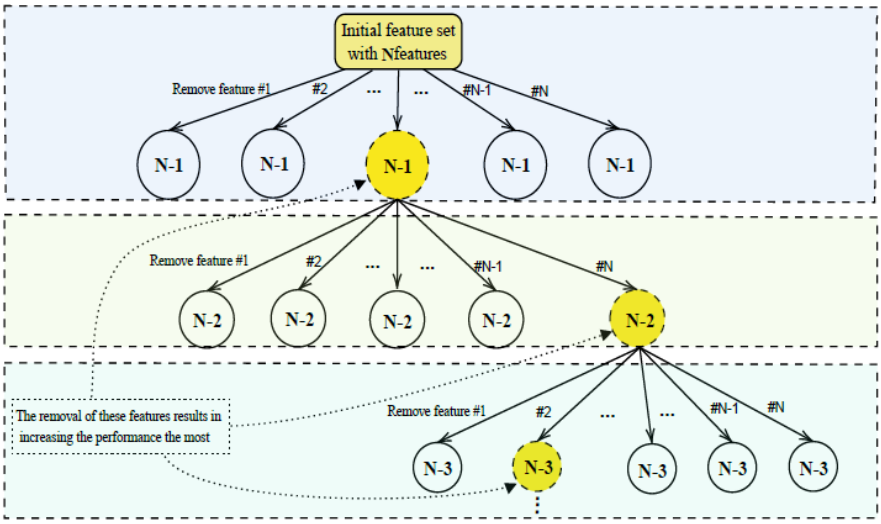


Figure 2.6: *This illustration depicts the three steps of the SBS subset selection algorithm. It begins with all N features, then sequentially removes one feature at a time, assessing performance with a wrapped classifier. The feature whose removal leads to the greatest accuracy improvement is selected. The process repeats with the remaining $N - 1$ features. Source:[1]*

2.3.3 Embedded Methods

Unlike the wrapper strategy, which uses a heuristic search guided by the classifier's performance. embedded strategy offers a compromise option between filter and wrapper methods. It uses the learning process itself to both perform feature selection and build an optimized classifier. Embedded approach, which serves as an intermediate method between filters and wrappers, selecting features. These methods evolve during the learning process based on the classifier's evaluation criterion, typically resulting in lower computational costs compared to wrapper methods. The most common embedded techniques are regularization methods [1].

LASSO

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a linear regression technique that employs L1 regularization to improve model generalization by applying a penalty. LASSO offers several advantages, as it is highly effective at reducing variance because the coefficients of the misleading features are pe-nalized and removed ,consequently ,it helps avoiding over-fiting problems , thus ,better generalization ability . Moreover ,LASSO is very useful at interpreptability enchacement by canceling irrelevant features.

There are different mathematical forms to introduce LASSO method; According to Robert Tibshirani, The lasso estimate is defined by the solution to the L_1 optimization problem.

$$Minimize\left(\frac{\|Y - X\beta\|_2^2}{n}\right) \text{ subject to } \sum_{j=1}^{k} \|\beta\|_1 < t \tag{2.9}$$

Where t is the upper bound for the sum of the coefficients. This optimization problem is equivalent to the parameter estimation that follows:

$$\hat{\beta}(\lambda) = argmin_\beta \left(\frac{\|Y - X\beta\|_2^2}{n} + \lambda\|\beta\|_1\right) \tag{2.10}$$

Where $\|Y - X\beta\|_2^2 = \sum_{i=0}^{n}(\|Y_i - (X\beta)_i\|^2)$ and $\lambda \geq 0$ is the regularization parameter that control the amount of shrinkage. The lasso method has some limitations:

- From each group of correlated features, LASSO tends to select one feature from each group and neglects other features.
- In datasets where m (number of instances) is small, and n (number of features) is

large, LASSO selects at most m features before it saturates.

Random forest for feature selection

Random forest is one of the machine learning algorithms known for its excellent prediction accuracy and its ability to select informative variables, making it the most widely used machine learning algorithm. This is due to its associated variable importance measures, the Gini index and mean decrease accuracy.

Gini index: measures the effectiveness of each variable's split in separating samples of the two classes within a given node, averaged across all trees.

Mean decrease accuracy is calculated by permuting data in out-of-bag (OOB) samples. It measures importance by comparing the prediction error on OOB samples with the error after permuting data values, averaged across all trees in the forest[1].

2.4 Metaheuristic Optimization for Feature Selection

2.4.1 Overview of Metaheuristics

Metaheuristic optimization algorithms are advanced search methods that are meant to quickly search through large and complicated search spaces to find solutions that are close to optimal. These techniques are especially effective for addressing combinatorial optimization problems when conventional exact methods become computationally impractical (Blum and Roli, 2003[8]).

The goal of feature selection is to find the best subset of features from a set of n features. This issue is fundamentally combinatorial, given that the quantity of potential feature subsets increases exponentially 2^n. So, exhaustive search methods don't work properly, especially for high-dimensional datasets that are often used in real-world applications like detecting Android malware (Guyon and Elisseeff, 2003[15]).

Metaheuristic algorithms deal with this problem by using smart search methods that find a balance between two basic strategies: exploration and exploitation. Exploration is the ability to look into different parts of the search space to avoid local optima. Exploitation, on the other hand, is the process of improving the current best solutions to get better results. This balance lets metaheuristics move through large search spaces and find high-quality subsets of features.

Table 2.1: Merits and demerits of search algorithms from A Review of Feature Selection and Its Methods by B. Venkatesh, J. Anuradha

Search algorithm	Category	Merits	Demerits
SFS & SBS	Suboptimal FS	• Computational time is very less • Simple to implement • Robust to multi-colinearity	• Redundant features are selected • Nesting effect • Fails to provide optimal results
SFFS & SFBS	Suboptimal FS	• Reduce redundant features • Overcomes Nesting effect • Provide optimal results	• NP-hard to search all subsets • A monotonic behavior of the FS
Plus-l-minus-$r(l-r)$	Suboptimal FS	Overcomes the nesting effect	No hypothetical method for predicting the values of l and r
Exhaustive search	Optimal FS	Reduce the computational time	• Not suitable for large dataset • NP-hard to search all subsets
Branch and bound	Optimal FS	Reduce the computational time	• Not suitable for large dataset • NP-hard to search all subsets

2.4.2 Common Metaheuristic Algorithms

Several metaheuristic algorithms have been the widely adopted for feature selection due to their flexibility and effectiveness in handling high-dimensional problems .

Genetic Algorithm (GA) is one of the earliest and most popular metaheuristic methods inspired by the process of natural evolution (Holland, 1992[19]). In GA, each possible solution is shown as a chromosome that holds a part of the features. The algorithm uses genetic operators like selection, crossover, and mutation to adjust the population over time, with the goal of producing better solutions.

Another widely utilized method is Particle Swarm Optimization (PSO), which is based on how fish schools and bird flocks behave (Kennedy and Eberhart, 1995[**?**]). In PSO, each particle is a possible feature subset that moves through the search space by modifying its position based on what it knows and the best solution the swarm has found due this collaborative behavior convergence to optimal or near-optimal solutions is efficient .

The Whale Optimization Algorithm (WOA) is a newer metaheuristic based on how humpback whales hunt (Mirjalili and Lewis, 2016)[28]. It uses methods like encircling prey, spiral updating, and random search to mimic the bubble-net feeding strategy. Because it can explore and exploit effectively, WOA has shown promise in solving hard optimization problems .While GA and PSO are established benchmarks, WOA was specifically selected for this research due to its superior mathematical framework for balancing exploration and exploitation. Its unique bubble-net feeding mechanism provides a more dynamic transition between searching for new feature subsets and refining existing ones, which is critical for navigating the high-dimensional datasets characteristic of Android malware.

2.4.3 Advantages in High-Dimensional Problems

Metaheuristic optimization techniques offer several advantages when applied to feature selection in high-dimensional domains.

First, they are capable of efficiently exploring extremely large search spaces without requiring exhaustive evaluation of all possible feature subsets. This makes them particularly suitable for NP-hard problems, where traditional methods become computationally infeasible.

Second, metaheuristic algorithms are able to capture complex interactions between features. Unlike filter methods, which evaluate features independently, metaheuristics evaluate subsets of features as a whole, leading to improved predictive performance.

Third, these methods provide a good trade-off between solution quality and computational cost. Although they do not guarantee finding the global optimum, they are capable of identifying near-optimal solutions within a reasonable time frame.

Finally, metaheuristic-based feature selection is highly suitable for real-world applications such as Android malware detection, where datasets are characterized by high dimensionality, feature redundancy, and non-linear relationships. In such contexts, metaheuristic algorithms can significantly improve model performance while maintaining computational efficiency. Based on these advantages, metaheuristic optimization represents a powerful approach for feature selection, motivating its adoption in this work to enhance Android malware detection performance.

2.5 Literature Review

The preceding sections established the theoretical foundations of feature selection, including filter, wrapper, embedded, and metaheuristic optimisation approaches. This section reviews existing Android malware detection studies through the lens of feature engineering and feature selection practices. Beyond reporting detection performance, particular attention is given to how features are selected, reduced, or utilised, since feature selection directly influences computational efficiency, model complexity, interpretability, and deployment feasibility in resource-constrained Android environments.

2.5.1 Feature Selection-Based Android Malware Detection Approaches

Feature selection has increasingly been recognised as a critical component of Android malware detection systems due to the high dimensionality of features extracted from permissions, API calls, network traffic, and system call traces. Unlike classification algorithms, which focus on learning decision boundaries, feature selection aims to eliminate irrelevant and redundant information, thereby reducing computational complexity while preserving discriminative capability.

Alhogail and Alharbi (2025) [2] proposed a two-stage machine learning framework for Android malware detection using the CICMalDroid 2020 dataset. Their approach combined Chi-square statistical ranking with Select-From-Model (SFM) feature selection before applying Random Forest, K-Nearest Neighbours, Naïve Bayes, and Decision Tree classifiers. The framework reduced the original 470-feature space to 120 features, corresponding to a dimensionality reduction of 74.5%, while achieving 97.82% binary classification accuracy and 96.09% multiclass accuracy. Although effective, the feature selection process relies exclusively on filter-based techniques and does not exploit feature interactions through wrapper optimisation. Furthermore, although Random Forest inherently provides feature importance measures, the authors still required an external feature selection stage to achieve substantial dimensionality reduction, suggesting that classifier-intrinsic mechanisms alone may be insufficient for obtaining compact and computationally efficient feature subsets

Several studies have also investigated dimensionality reduction techniques such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) for Android malware classification. These approaches transform the original feature space into a smaller set of latent variables before classification. Experimental results indicate that while PCA substantially reduces dimensionality, it may lead to performance degradation due to information loss. LDA generally preserves classification performance more effectively; however, both techniques generate transformed features that lack interpretability and do not

explicitly identify the most informative original attributes.

Metaheuristic optimisation algorithms have been explored in broader cybersecurity and intrusion detection domains for feature subset selection. Techniques such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) have demonstrated the ability to identify compact feature subsets while maintaining strong predictive performance. However, their application within Android malware detection remains limited. The systematic review conducted by Yunmar et al. (2024) [43] found no evidence of Whale Optimization Algorithm (WOA) or Binary Whale Optimization Algorithm (BWOA) being applied to Android malware feature selection among the 88 studies reviewed.

Collectively, these studies demonstrate that existing Android malware detection research primarily relies on filter-based ranking methods or dimensionality reduction techniques. The integration of statistical filtering with a global metaheuristic wrapper search remains largely unexplored, particularly for dynamic syscall-based feature spaces such as those found in CICMalDroid 2020. This limitation directly motivates the hybrid Filter-BWOA framework proposed in this work.

2.5.2 Machine Learning-Based Detection Approaches

Tyagi and Gautam (2024) [37] evaluated five machine learning classifiers on the DREBIN dataset, comprising 123,453 applications and 5,560 malware samples. Using Android permissions and application metadata as features, their Sequential Neural Network achieved 99% accuracy, outperforming Decision Tree, Support Vector Machine, K-Nearest Neighbour, and Logistic Regression classifiers. Although the results demonstrate the effectiveness of metadata-based static analysis, the study performs no explicit feature selection and operates on the complete feature space.

Bhat et al. (2023) [6] proposed a stacking ensemble approach based on behavioural feature analysis. A redundancy elimination strategy was applied prior to classification, resulting in an accuracy of 98.08%. Their findings demonstrate that even simple feature reduction techniques can improve detection performance. Nevertheless, no systematic wrapper-based or metaheuristic feature selection strategy was employed.

Muzaffar et al. (2023) [30] conducted a comprehensive benchmarking study by reimplementing sixteen Android malware detection methods on a standardised dataset containing approximately 124,000 applications. Their evaluation showed that static features alone can achieve detection accuracies of up to 96.8%. More importantly, they highlighted significant inconsistencies in feature engineering and preprocessing practices across the literature, emphasising the need for reproducible and systematic feature selection frameworks.

Gómez and Muñoz (2023) [13] trained several machine learning models, including a

Multilayer Perceptron (MLP), using static features extracted from CICMalDroid 2020, CICMalDroid 2017, and CICAndMal 2017 datasets. Their MLP achieved an F1-score of 98.9%. However, the selected feature subset was manually curated from prior literature rather than generated through a systematic feature selection process.

2.5.3 Deep Learning and Advanced Detection Approaches

Millar et al. (2021) [27] proposed a multi-view deep learning architecture for zero-day Android malware detection. Separate Convolutional Neural Networks were trained on opcode sequences and API calls, while a fully connected neural network processed permission features. These outputs were subsequently fused to perform final classification. The model achieved F1-scores of 0.9927 on the Drebin dataset and 0.9963 on the AMD dataset. However, the feature representation consisted of 23,226 dimensions and no feature selection was applied. The absence of dimensionality reduction in such a large feature space represents a significant computational burden and illustrates the scalability challenge addressed in this work.

Yizheng et al. (2023) [41] introduced a hierarchical contrastive learning framework combined with a sample selection strategy for continuously updating Android malware classifiers. Their approach reduced false positive rates from 0.86% to 0.48% and false negative rates from 14% to 9%. While effective in improving classification reliability, the framework applies no feature dimensionality reduction and therefore does not address feature-space optimisation.

2.5.4 Dynamic and System Call-Based Approaches

Guerra-Manzanares et al. (2022) [14] investigated the challenge of concept drift in Android malware detection using dynamic system call analysis. Their Multi-Machine Learning Pool approach was evaluated on a seven-year test set derived from the KronoDroid dataset. Results demonstrated that syscall frequency features remain highly discriminative for malware detection across extended time periods. Notably, no feature selection was applied to the syscall feature space, leaving open the question of whether a compact subset of system calls could preserve detection performance while reducing computational overhead.

2.5.5 Systematic Review of the Field

Yunmar et al. (2024) [43] conducted a systematic literature review of 88 hybrid Android malware detection studies published between 2012 and 2023. Their review provides the

most comprehensive analysis of feature utilisation and feature selection practices within the field. The authors report that system calls constitute the most frequently used dynamic feature type, appearing in 43 of the reviewed studies, followed by network activity features in 32 studies. Critically, only 48 of the 88 studies (54.5%) applied any form of feature selection, while 13 failed to clearly describe the method employed. Among studies that implemented feature selection, Information Gain and Chi-square filter methods were the most common approaches. Metaheuristic wrapper methods were entirely absent, including Whale Optimization Algorithm and its binary variant. Furthermore, 95% of reviewed studies relied on non-integrated detection pipelines requiring manual intervention. These findings confirm that globally optimised feature selection remains an underexplored area in Android malware detection and provide strong justification for investigating the proposed Filter-BWOA framework.

Table 2.2: Comparison of Android Malware Detection Studies with Emphasis on Feature Selection Practices

Author (Year)	**Core Methodology**	**Feature Types**	**Dataset Size**	**Accuracy / Performance**	**Feature Selection (FS)**
Millar et al. (2021)	Multi-view Deep Learning (CNN + FNN)	Opcodes, Permissions, API Calls	Drebin & AMD	0.9927 (F1)	None (Vector size: 23,226)
Muzaffar et al. (2023)	Benchmarking 16 Studies	Static Features	124,000 Apps	96.8%	Static Analysis
Bhat et al. (2023)	Stacking Ensemble	Behavioral Features	Not specified	98.08%	Redundancy Elimination
Gómez & Muñoz (2023)	Multilayer Perceptron (MLP)	Static Features	18,000+ (CIC)	98.9% (F1)	Recognized Static Subset
Yizheng et al. (2023)	Hierarchical Contrastive Learning	Sample Selection	Not specified	FPR: 0.48% / FNR: 9%	Sample Selection Strategy
Tyagi & Gautam (2024)	Sequential Neural Network	Permissions & Metadata	123,453 Apps	99%	None (Focus on Metadata)
Yunmar et al. (2024)	SLR of 88 hybrid studies	Static + Dynamic	88 studies reviewed	N/A (survey)	Only 54.5% of studies use any FS; IG/Chi^2 dominate; no WOA applied
Alhogail Alharbi (2025)	RF, KNN, NB, DT	Dynamic Syscalls	11,598	97.82%	Chi^2 + SFM (120/470)

2.6 Research Gap and Motivation

Based on the review of feature selection paradigms and recent advances in Android malware detection, several critical research gaps can be identified.First, the scalability challenge in high-dimensional malware datasets remains a fundamental issue. Modern Android malware detection systems rely on thousands of features extracted from permissions, API calls, and system behaviors. While several studies, such as Millar et al. (2021)[27], achieve high detection performance using feature vectors of 23,226 dimensions without any reduction,

this pattern is not isolated. Yunmar et al. (2024) [43] confirm through systematic review that 45.5% of hybrid Android malware detection studies apply no feature selection whatsoever, defaulting to full feature spaces that increase computational complexity and limit deployment in resource-constrained mobile environments.

Second, despite the success of tree-based machine learning models, such as Random Forest and XGBoost, in handling tabular malware data, these models are typically applied on the full feature space. Although they exhibit robustness to irrelevant features, they do not explicitly address the problem of feature redundancy and may still incur unnecessary computational overhead. Therefore, relying solely on model-intrinsic feature importance is insufficient for achieving optimal feature compactness and efficiency.

Third, traditional feature selection techniques exhibit inherent limitations when applied to high-dimensional security datasets. Filter-based methods such as Information Gain and Chi-square dominate the limited feature selection work that does exist — Yunmar et al. (2024)[43] identify these as the most frequently applied techniques across the 35 studies that clearly describe their selection method. However, these filters evaluate features independently and fail to capture complex inter-feature relationships. Classical wrapper methods address this limitation but suffer from high computational cost and local optima trapping due to their sequential greedy search. This creates a methodological gap: the field predominantly uses either no selection or the simplest filters, leaving the design space of global metaheuristic wrapper methods substantially unexplored for dynamic syscall feature spaces.

Furthermore, Although metaheuristic algorithms such as Genetic Algorithm and Particle Swarm Optimization have been applied to feature selection in Android malware detection, Yunmar et al. (2024) [43] identify metaheuristic wrapper methods as absent from the feature selection practices of the 88 studies they reviewed — none employ WOA or its binary variant. This confirms that Binary Whale Optimization Algorithm (BWOA), despite its demonstrated effectiveness in other high-dimensional optimisation problems, has not been applied to Android dynamic syscall feature selection, representing a concrete and documented research gap.

Motivated by these limitations, this research aims to develop an efficient and scalable feature selection framework for Android malware detection. Specifically, this work proposes a Binary Whale Optimization Algorithm (BWOA) as a wrapper-based feature selection method, integrated with high-performance tree-based classifiers (Random Forest and XGBoost).

The main objective is to significantly reduce the dimensionality of the feature space while maintaining—or potentially improving—classification performance. Unlike traditional approaches that rely solely on classifier-internal mechanisms, the proposed method

explicitly searches for an optimal subset of features by balancing classification accuracy and subset size.

By combining the global search capability of BWOA with the robustness of tree-based models, this research aims to achieve a better trade-off between accuracy, computational efficiency, and model interpretability, thereby addressing the practical challenges of deploying malware detection systems in real-world environments

2.7 Conclusion

This chapter has provided a comprehensive theoretical foundation for feature selection, moving from its historical roots in 1997 to the advanced metaheuristic techniques of today. We have established that for modern, high-dimensional datasets especially in Android malware detection the Curse of Dimensionality makes dimensionality reduction a necessity rather than an option.

Through analysis of the Filter, Wrapper, and Embedded paradigms, we concluded that while filters offer speed, the Wrapper approach (specifically when powered by metaheuristics like WOA) offers the best opportunity to capture complex feature dependencies. The review of existing work highlighted that the "Gap" in current research is not a lack of accuracy but a lack of optimized feature-model synergy, as many high-performing models still rely on thousands of redundant and noisy attributes.

Therefore, this chapter lays the groundwork for Chapter 3, where we introduce our Proposed Methodology. It will describe the implementation of a Hybrid tree-based classifiers (Random Forest and XGBoost) aimed at identifying the best feature subset globally, resulting in a detection system that is both malware-resistant and performance-optimized.

Chapter 3

Proposed Approach – Hybrid Tree-Based Wrapper Feature Selection Framework

3.1 Introduction

This chapter presents the proposed hybrid framework, which integrates both paradigms in a sequential two-stage architecture. In the first stage, an ensemble of three complementary filter methods Information Gain, Fisher Score, and Chi-Squared eliminates obviously irrelevant features through a computationally cheap voting fusion process, reducing the search space from hundreds of features to a manageable candidate set. In the second stage, the Binary Whale Optimization Algorithm (BWOA) performs fine-grained wrapper-based selection within this reduced candidate set, guided by Random Forest and XGBoost classifiers as fitness evaluators. The resulting feature subset is then used to train the final classification models.

Three design principles guided the framework's development. First, strict leakage prevention: all feature selection operations are performed exclusively on training data within each cross-validation fold, ensuring that no information from validation or test sets contaminates the selection process. Second, empirical parameter validation: all hyperparameters including the BWOA regularisation coefficient α, the filter voting threshold, and the early stopping patience were determined through systematic ablation experiments rather than assumed from prior work. Third, dual-task evaluation: the framework was validated on both binary classification (malware versus benign) and multiclass classification (malware family detection) to demonstrate task-generality of the selected feature subsets.

The remainder of this chapter is organised as follows. Section 3.2 presents an overview of the complete framework architecture. Section 3.3 describes the data preprocessing pipeline.

Section 3.4 details the Binary WOA adaptation for binary feature selection. Section 3.5 presents the fitness evaluation strategy. Section 3.6 describes the final model training and evaluation protocol. Section 3.7 concludes the chapter.

3.2 Overview of the Proposed Framework

The proposed framework follows a five-stage sequential pipeline illustrated in Figure 3.1. Each stage is designed to address a specific aspect of the malware detection problem, and together they form a coherent methodology that balances detection performance, feature compactness, computational efficiency, and evaluation rigour.

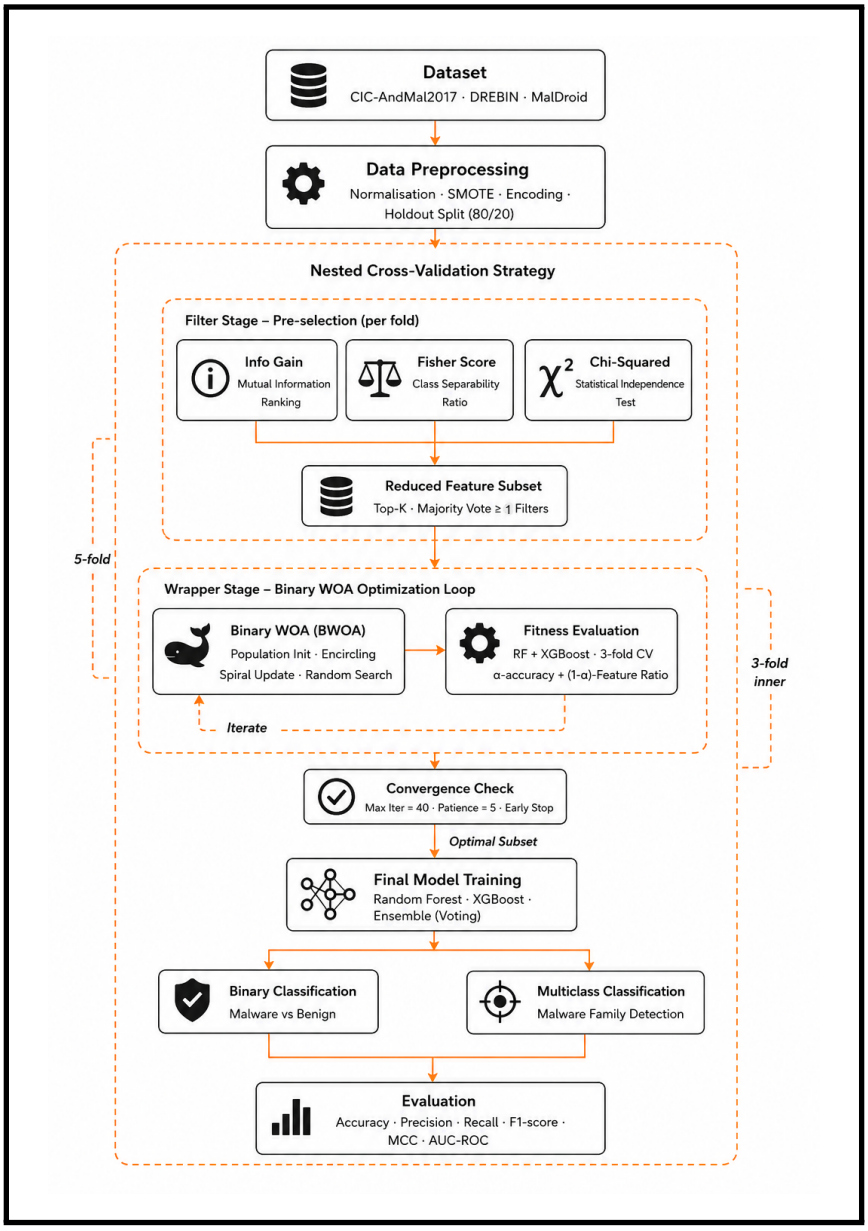


Figure 3.1: Overall architecture of the proposed hybrid filter-BWOA-classifier framework for Android malware detection

Stage 1 — Dataset acquisition and preprocessing. Raw Android malware datasets are loaded and subjected to a preprocessing pipeline that handles missing values, normalises feature scales, encodes class labels, and addresses class imbalance. The full dataset is partitioned into a development set (80%) and a permanently locked holdout test set (20%).

Stage 2 — Outer cross-validation fold split. A 5-fold stratified cross-validation loop is applied to the development set. Within each fold, the training partition is used exclusively for all feature selection and model training operations. The test partition is used only for performance evaluation it is never accessed during feature selection.

Stage 3 — Filter ensemble preselection. Within each training fold, three filter methods Information Gain, Fisher Score, and Chi-Squared independently rank all features by relevance. A voting fusion mechanism retains only features ranked in the top-K by at least one out of three methods, producing a high-quality reduced candidate set of approximately 150 features from the original 470 (Figure 3.2).

Stage 4 — Binary WOA wrapper optimisation. The BWOA performs fine-grained feature subset search within the filter-reduced candidate set. Each whale encodes a candidate binary feature mask. Fitness is evaluated using a lightweight classifier (50-tree RF or XGBoost) via 3-fold inner cross-validation. The algorithm iterates for up to 40 iterations with early stopping patience of 5, returning the feature mask with the lowest fitness value.

Stage 5 — Final model training and evaluation. The optimised feature subsets are used to train three final classifiers: Random Forest, XGBoost, and a soft-voting ensemble. Models are evaluated on both the cross-validation test folds and the locked holdout set using six metrics: accuracy, precision, recall, F1-macro, MCC, and AUC-ROC.

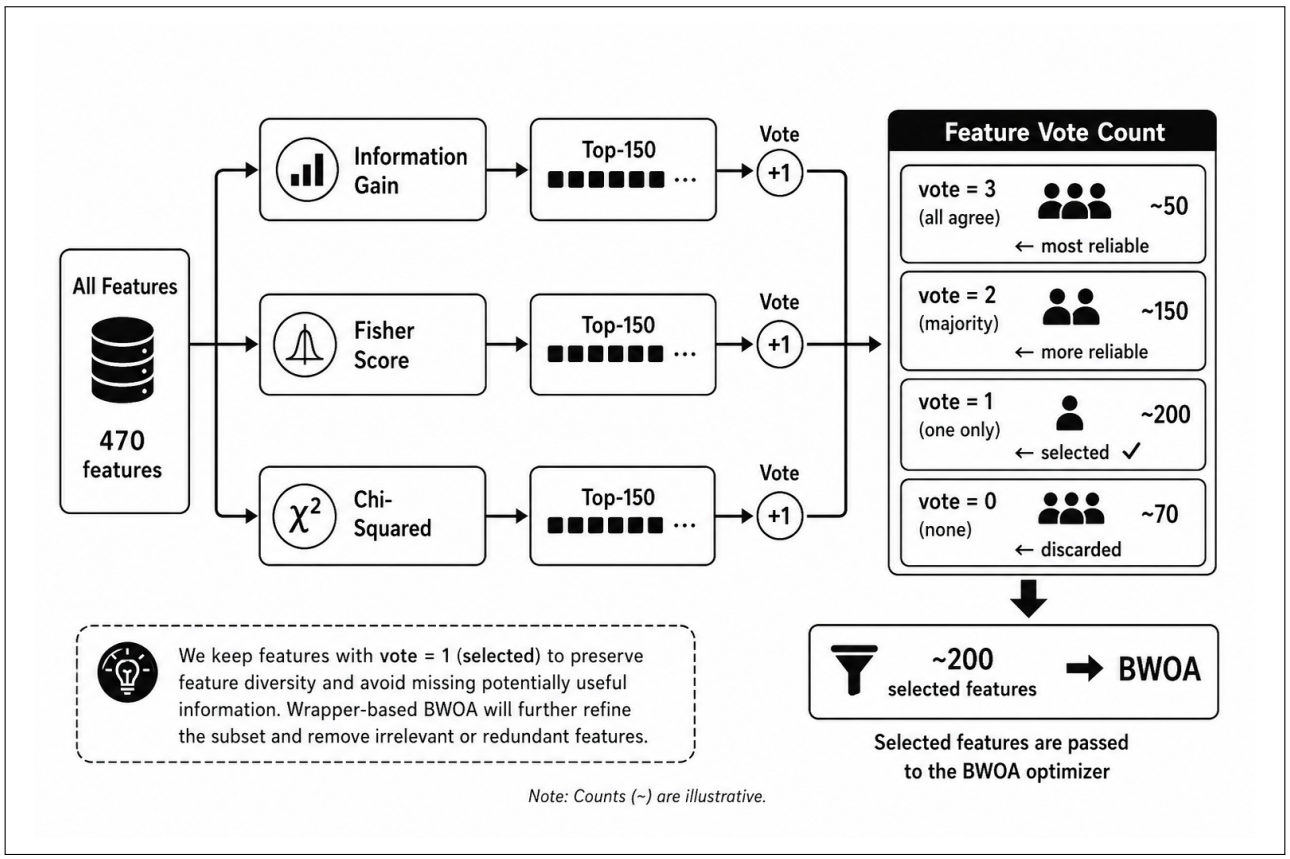


Figure 3.2: Filter voting fusion diagram

3.3 Data Preprocessing Pipeline

Preprocessing transforms raw dataset files into clean, normalised, balanced feature matrices suitable for machine learning. All preprocessing operations that involve fitting specifically the scaler and SMOTE are applied exclusively within training folds to prevent data leakage.

3.3.1 Handling Missing and Infinite Values

Feature extraction from Android applications can produce undefined values in two scenarios: division by zero during ratio-based feature computation produces infinite values, and incomplete dynamic analysis executions produce missing (NaN) values. Both must be resolved before normalisation. Infinite values are first converted to NaN to unify the treatment of undefined entries. Missing values are then imputed using the per-feature median. Median imputation was selected over mean imputation because syscall frequency distributions are right-skewed a small number of applications execute certain calls at very high frequency, inflating the mean and making it an unrepresentative imputation value. The median is robust to such outliers and better represents the typical feature value for

imputation purposes.

3.3.2 Feature Normalization

Feature normalization was applied using Min–Max scaling, which transforms each feature into the interval $[0, 1]$ according to

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

where x_{min} and x_{max} are computed exclusively from the training partition of each fold. The fitted scaler is then applied to the corresponding test partition using only the training statistics. This ensures that the normalization process does not incorporate any distributional information from the test data.

Normalization serves two purposes in this pipeline. First, the Chi-Squared filter requires non-negative feature values, which are guaranteed by Min–Max scaling. Second, for BWOA fitness evaluation, normalized features ensure that the classifier's performance estimate is not distorted by features with different numeric scales, resulting in more reliable fitness comparisons between candidate subsets.

3.3.3 Feature Encoding

Class labels representing malware families and the benign category were encoded as consecutive integers using label encoding. For binary classification, malware samples were assigned label 1 and benign samples label 0. For multiclass classification, each malware family was assigned a unique integer label. The original string labels are preserved in a class name mapping to enable interpretable reporting of per-class metrics.

3.3.4 Handling Class Imbalance

Android malware datasets typically exhibit class imbalance, where certain malware families are underrepresented relative to others. Training classifiers on imbalanced data without correction tends to produce models biased toward the majority class, resulting in poor recall on minority malware families which is the most safety-critical failure mode in a security application.

The Synthetic Minority Oversampling Technique (SMOTE) was employed to address this imbalance. SMOTE generates synthetic minority-class samples by interpolating between existing minority samples and their k nearest neighbours in feature space(figure 3.3).

In this work, k=3 neighbours were used. A smaller neighbourhood value was selected because the syscall frequency feature space is high-dimensional and sparse; larger k values risk interpolating across class boundaries, generating synthetic samples that do not represent valid malware behaviour patterns.

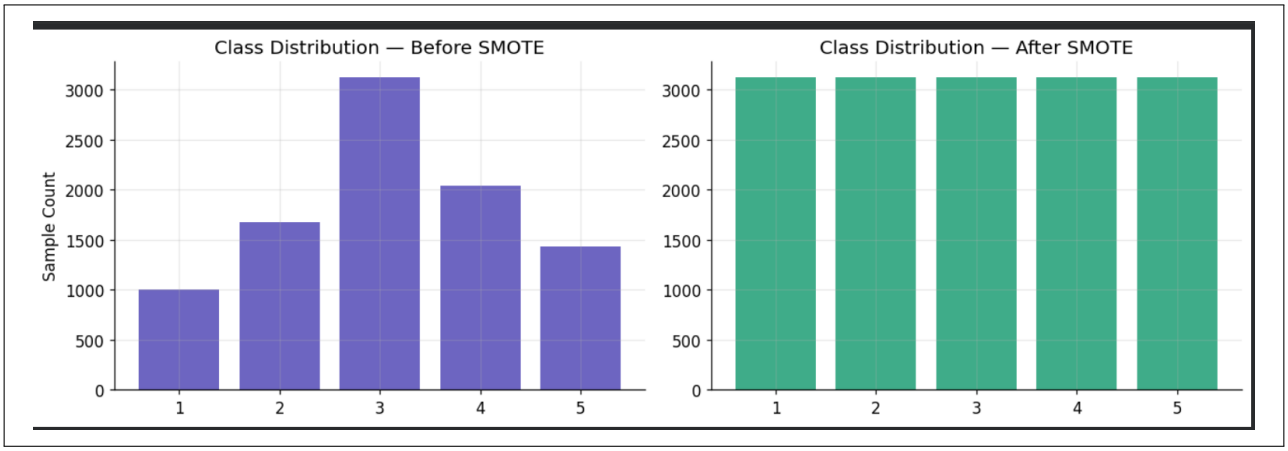


Figure 3.3: Class distribution before and after SMOTE

Critically, SMOTE is applied only to the training partition of each fold, after the scaler has been fitted and applied. Applying SMOTE before splitting, or fitting the scaler on SMOTE-augmented data, would constitute data leakage. The test partition always consists of real, unaugmented samples.

3.3.5 Dataset Splitting Strategy

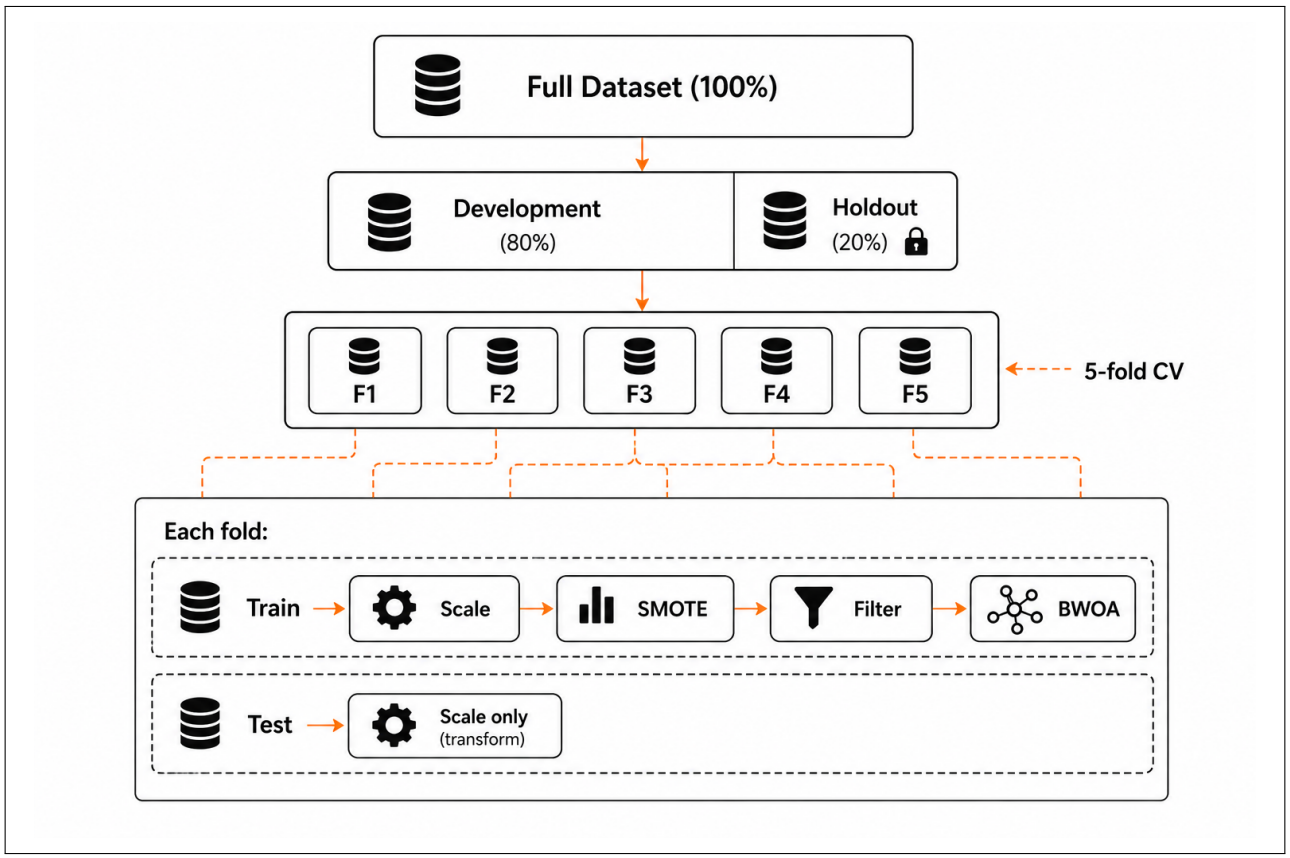


Figure 3.4: Data splitting strategy diagram

The evaluation strategy follows a two-level nested structure designed to provide both an unbiased cross-validation estimate and a true out-of-sample performance measurement.

At the outer level, the full dataset is split into a development set (80%) and a permanently locked holdout test set (20%) using stratified random sampling. The holdout set is isolated at this point and is never accessed during any training, feature selection, or hyperparameter tuning operation. It is used exclusively for the final performance evaluation reported in Chapter 4.

At the inner level, the development set is subjected to 5-fold stratified cross-validation. Stratification preserves the class distribution across all folds, preventing evaluation bias that could arise if one fold contained a disproportionate number of easy or difficult samples. Within each fold, all operations — scaling, SMOTE, filter feature selection, and BWOA optimisation are performed on the training partition and applied to the test partition without refitting. Table 3.1 summarises the leakage prevention strategy applied at each pipeline stage.

Table 3.1: Data leakage prevention summary

Pipeline stage	Fit on	Applied to	Leakage risk if reversed
Min-Max scaler	Train fold only	Train + test fold	Test statistics contaminate training normalisation
SMOTE	Train fold only	Train fold only	Synthetic test samples inflate evaluation metrics
Filter methods (IG, Fisher, Chi^2)	Train fold only	Train fold only	Test feature rankings bias subset selection
BWOA optimisation	Train fold only	Train fold only	Test performance guides feature selection
Final holdout evaluation	—	Holdout set only	Any earlier access invalidates the holdout estimate

3.4 Binary Whale Optimization Algorithm for Feature Subset Search

3.4.1 Overview of the Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) is a swarm intelligence metaheuristic introduced by Mirjalili and Lewis (2016), inspired by the bubble-net hunting behaviour of humpback whales. During a hunt, humpback whales swim in a shrinking spiral pattern around their prey while releasing bubbles, creating a net that traps fish near the surface. WOA mathematically models this behaviour through three position-update mechanisms that balance exploitation of the current best solution with exploration of the search space.as mentioned in chapter 2.

WOA was selected for this work for three reasons. First, it requires fewer hyperparameters than competing metaheuristics such as Genetic Algorithms or Particle Swarm Optimization, reducing the tuning burden. Second, it has demonstrated competitive performance on binary feature selection benchmarks in the intrusion detection and malware detection literature. Third, its exploration-exploitation balance is controlled by a single linearly decreasing parameter a, which provides a natural and interpretable convergence

schedule.

3.4.2 Solution Encoding

Each whale in the population represents a candidate feature subset encoded as a binary vector of length equals the number of features passing the filter stage. A value of 1 at position i indicates that feature i is included in the subset, whereas a value of 0 indicates that it is excluded.

Formally, for a filter-reduced feature space of dimension d (approximately 150 features in this work), each whale X_i is represented as:

$$X_i = [b_1, b_2, \ldots, b_d], \quad b_j \in \{0, 1\}$$

Feature indices:	f_1	f_2	f_3	f_4	f_5	f_6	f_7	$\cdots$	f_{150}
Binary mask:	1	0	1	1	0	0	1	$\cdots$	1
	$\downarrow$		$\downarrow$	$\downarrow$			$\downarrow$		$\downarrow$
Selected:	f_1		f_3	f_4			f_7		f_{150}

Figure 3.5: Binary encoding diagram

With 20 agents and approximately $d \approx 150$ features, the population explores 20 points in a search space containing 2^{150} possible feature subsets. Exhaustive search over such a space is computationally intractable, which justifies the use of a metaheuristic optimization approach.

To guarantee solution validity, at least one feature must always remain selected. Therefore, if all bits of an agent become zero after a position update, one randomly selected bit is forcibly set to 1.

3.4.3 Population Initialization

The initial population is generated randomly, where each bit is independently assigned a value of 1 with probability 0.5. Consequently, the expected number of selected features per agent is approximately $d/2 \approx 75$.

This initialization strategy promotes diversity in the population by distributing candidate solutions across the search space rather than concentrating them near empty or fully selected subsets.

3.4.4 Transfer Function for Continuous-to-Binary Conversion

WOA position update equations operate in continuous space. Therefore, converting the resulting continuous values into binary form requires the use of a transfer function. In this work, the V-shaped hyperbolic tangent transfer function was adopted:

$$T(x) = |\tanh(x)|$$

This function maps any real-valued input to the interval $[0, 1]$. A bit flip from its current value b_j^t occurs with probability $T(x_j^{t+1})$ according to:

$$b_j^{t+1} = \begin{cases} 1 - b_j^t & \text{if } r < T(x_j^{t+1}) \\ b_j^t & \text{otherwise} \end{cases}$$

where:

$$r \sim \text{Uniform}(0, 1)$$

The V-shaped family of transfer functions was preferred over S-shaped (sigmoid) alternatives because V-shaped functions assign non-negligible flip probabilities to both large positive and large negative continuous values as shown in the Figure 3.5. This allows bits to flip in both directions regardless of the sign of the continuous update, improving the algorithm's ability to escape local optima.

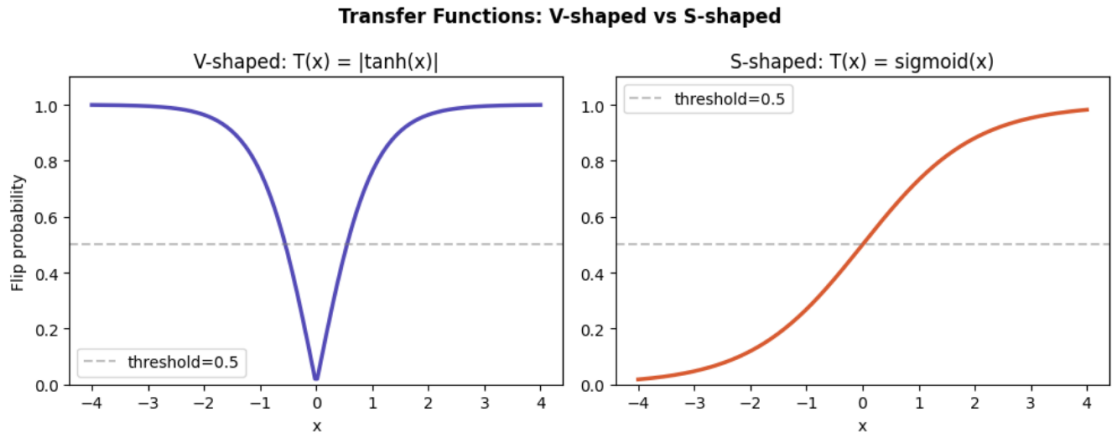


Figure 3.6: *V-shaped vs S-shaped transfer function comparison*

3.4.5 WOA Operators Adapted for Binary Feature Selection

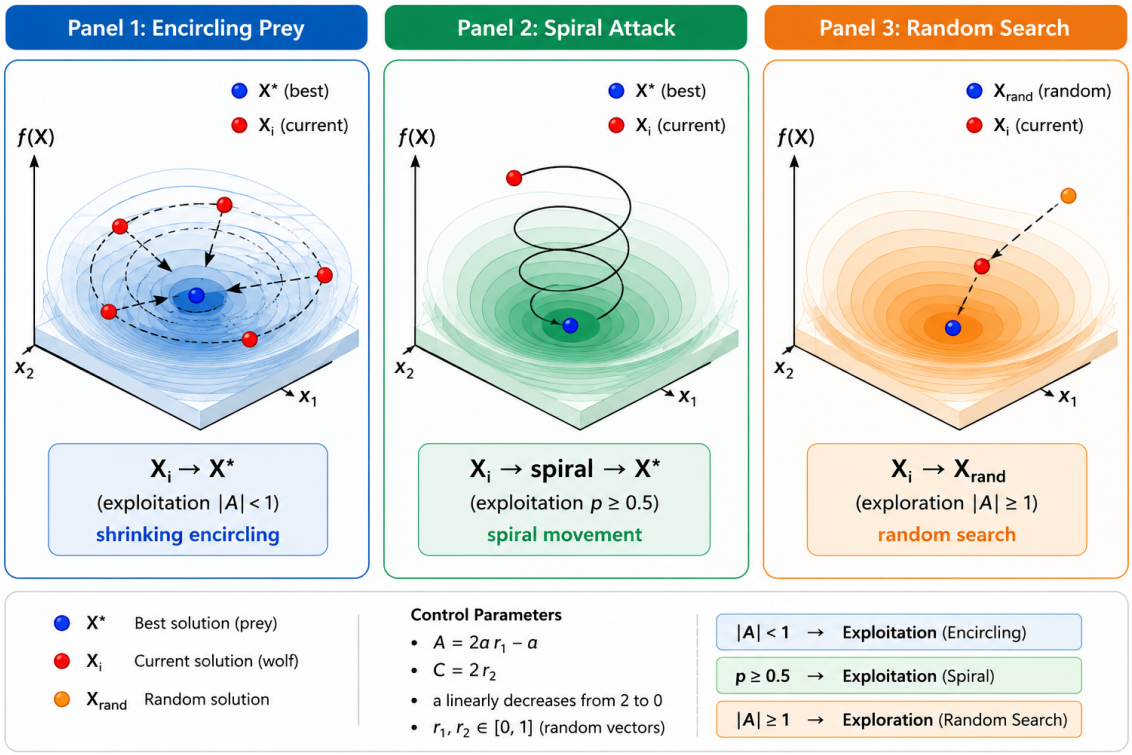


Figure 3.7: WOA operator diagram

Encircling Prey (Exploitation, $|A| < 1$) When the random coefficient A has magnitude less than 1, the whale updates its position toward the current global best solution X^*:

$$\vec{D} = |C \cdot X^*(t) - X(t)|$$

$$X(t+1) = X^*(t) - A \cdot \vec{D}$$

where:

$$A = 2a \cdot r_1 - a$$

$$C = 2r_2$$

and:

$$r_1, r_2 \sim \text{Uniform}(0, 1)$$

The parameter a decreases linearly from 2 to 0 during the optimization process, progressively shifting the algorithm from exploration to exploitation.

Spiral Updating (Exploitation, $p \geq 0.5$) With probability 0.5, the logarithmic spiral update mechanism is applied instead of the encircling update:

$$X(t+1) = \tilde{D} \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t)$$

where:

$$\tilde{D} = |X^*(t) - X(t)|$$

$$b = 1$$

and:

$$l \sim \text{Uniform}(-1, 1)$$

This mechanism models the circular attack trajectory of humpback whales around trapped prey.

Random Search (Exploration, $|A| \geq 1$) When $|A| \geq 1$, a randomly selected whale X_{rand} replaces the best solution as the reference point:

$$\vec{D} = |C \cdot X_{\text{rand}}(t) - X(t)|$$

$$X(t+1) = X_{\text{rand}}(t) - A \cdot \vec{D}$$

This mechanism encourages exploration of regions far from the current best solution and helps prevent premature convergence to local optima.

The balance between exploitation ($|A| < 1$) and exploration ($|A| \geq 1$) is governed implicitly by the parameter a. Early in the search, large values of a make large $|A|$ values more probable, favouring exploration. As a approaches zero, smaller $|A|$ values dominate, causing the search to focus on refining solutions near the best solution found.

After each continuous position update, the transfer function converts the solution into binary form and the validity constraint is subsequently applied.

3.4.6 Pseudocode of the Adapted Binary WOA

Algorithm 1 Binary Whale Optimization Algorithm (BWOA) for Feature Selection

Require: Filtered training set X_{train}, labels y_{train}

Ensure: Best feature subset mask `best_mask`

Initialise population: $P \leftarrow \{X_i \in \{0,1\}^d \mid i = 1, \ldots, N\}$
Enforce that each X_i contains at least one selected feature
Evaluate fitness $f(X_i)$ for all agents using Equation (3.2)
`best` $\leftarrow \arg\min f(X_i)$
`no_improve` $\leftarrow 0$
for $t = 1$ to `max_iter` **do**
  $a \leftarrow 2 - 2\,t/$`max_iter`
  for each agent i **do**
    $r_1, r_2 \sim \text{Uniform}(0, 1)$
    $A \leftarrow 2ar_1 - a; \;\; C \leftarrow 2r_2$
    $p \sim \text{Uniform}(0, 1); \;\; l \sim \text{Uniform}(-1, 1)$
    if $p < 0.5$ **then**
      if $|A| < 1$ **then**
        $D \leftarrow |C \cdot$ `best` $- X_i|$
        $X_{\text{new}} \leftarrow$ `best` $- A \cdot D$
      else
        Randomly select X_{rand}
        $D \leftarrow |C \cdot X_{\text{rand}} - X_i|$
        $X_{\text{new}} \leftarrow X_{\text{rand}} - A \cdot D$
      end if
    else
      $D \leftarrow |$`best` $- X_i|$
      $X_{\text{new}} \leftarrow D\, e^l \cos(2\pi l) +$ `best`
    end if

Algorithm 1 Binary Whale Optimization Algorithm (BWOA) – continued

$X_i \leftarrow \texttt{Binarize}(X_{\text{new}})$
if $\sum X_i = 0$ **then**
Randomly flip one bit to 1
end if
end for
Update fitness values for all agents
if $\min(\texttt{fitness}) < f(\texttt{best})$ **then**
$\texttt{best} \leftarrow \arg\min(\texttt{fitness})$
$\texttt{no_improve} \leftarrow 0$
else
$\texttt{no_improve} \leftarrow \texttt{no_improve} + 1$
end if
if $\texttt{no_improve} \geq \texttt{patience}$ **then**
break
end if
end for
return `best_mask`

3.5 Fitness Evaluation Using Random Forest and XGBoost

3.5.1 Fitness Function Design

The fitness function quantifies the quality of a candidate feature subset by jointly measuring classification performance and subset compactness. A subset that achieves high predictive performance using many features is less desirable than one achieving comparable performance with fewer features. This trade-off is formalised as:

$$f(\text{mask}) = \alpha \cdot (1 - M) + (1 - \alpha) \cdot \frac{|S|}{|F|}$$

where M is the classification metric used by the optimisation process, $|S|$ is the number of selected features, $|F|$ is the total number of candidate features after the filter pre-

screening stage, and $\alpha \in [0, 1]$ controls the balance between predictive performance and compactness. The fitness function is minimised: a lower value corresponds to a better feature subset. The function has two interpretable limits when $\alpha = 1$ the function reduces to pure error minimisation with no compactness incentive, and when $\alpha = 0$ it minimises feature count regardless of performance. The value $\alpha = 0.985$ was selected to place 98.5% weight on classification performance while maintaining a modest but consistent pressure toward dimensionality reduction, ensuring that BWOA eliminates redundant features without sacrificing discriminative power.

Fitness Metrics for Binary and Multiclass Tasks

Different optimisation objectives were employed for the binary and multiclass experiments to reflect their distinct evaluation goals.

Binary malware detection. For the binary task, the fitness metric M is the F_2-score, computed as a special case of the generalised F_β-score:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

with $\beta = 2$. Setting $\beta = 2$ places twice the weight on recall relative to precision, directly encoding the asymmetric cost structure of Android malware detection: a false negative a malicious application classified as benign allows a threat to execute on the user's device, potentially causing data exfiltration, financial loss, or system compromise. A false positive a benign application flagged as malware causes user inconvenience but no security breach. These two error types are not equally costly, and a fitness metric that treats them as equivalent does not reflect the real-world deployment objective. By using F_2-score, BWOA is guided to select feature subsets that maximise malware detection recall rather than merely maximising overall classification accuracy. The F_2-score was computed using macro-averaging across all binary classes, ensuring that the minority malware class contributes equally to the fitness signal regardless of its smaller sample count relative to benign applications.

Multiclass malware categorisation. For the multiclass task, the fitness metric M is the macro-averaged F_1-score (F_1-macro):

$$F_{1\text{-macro}} = \frac{1}{K} \sum_{i=1}^{K} F_{1_i}$$

where K is the number of classes and F_{1_i} is the F_1-score of class i. Unlike the binary detection task, the multiclass objective is not to maximise malware recall alone but to achieve

balanced discrimination across all five malware family categories Adware, Banking malware, SMS malware, Riskware, and Benign. F_1-macro assigns equal importance to each class regardless of its frequency, preventing the optimisation from becoming biased toward majority classes and encouraging the selection of feature subsets that perform consistently across all malware families. This distinction in fitness metric between the binary and multiclass tasks reflects a principled design choice: the two tasks have different performance objectives, and the feature subsets selected under each objective are consequently different a finding confirmed empirically in Section 4.6.2, where the multiclass-optimised subset (116 features, CIDX-MC) achieves higher macro-F_1 than the binary-optimised subset when evaluated on the five-class task.

Computational Efficiency of the Fitness Proxy

The classifier used within the fitness function is a lightweight proxy a 50-tree Random Forest or 50-round XGBoost rather than the full final model used for holdout evaluation. This proxy captures the relative quality differences between candidate feature subsets accurately while reducing per-evaluation runtime by approximately 75% compared to the final 200-tree configuration. Since BWOA evaluates up to $n_{\text{agents}} \times n_{\text{iterations}}$ candidate subsets per fold ($20 \times 40 = 800$ evaluations in the configuration used here), this reduction is essential for making the wrapper search computationally tractable within the nested cross-validation protocol.

3.5.2 XGBoost Classifier

XGBoost implements gradient boosting, constructing an additive ensemble of decision trees by iteratively fitting new trees to the residuals of the current ensemble. The objective function combines a differentiable loss term with explicit regularisation:

$$L = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k} \Omega(f_k)$$

where

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

The objective function penalises tree complexity through the number of leaves T and the leaf weight magnitude $\|w\|^2$. This regularisation prevents individual trees from memorising training noise, which is especially important when the classifier operates on a small BWOA-selected feature subset where individual features carry high information density.

The XGBoost parameters used in this work (depth $= 4$, learning rate $= 0.05$, subsample $=$

0.8, colsample $= 0.8$, $\gamma = 0.1$) were confirmed through the ablation study described in Chapter 4.

3.5.3 Iterative Optimization Loop

During each BWOA iteration, all N agents are evaluated by the fitness function. The fitness computation for a single agent involves: extracting the feature subset defined by the agent's binary mask, training the proxy classifier on the SMOTE-augmented training fold restricted to those features, evaluating via 3 fold inner cross-validation, and computing the fitness score. With 20 agents, 3 inner folds, and up to 40 outer iterations, the worst-case evaluation count is $20{\times}40{\times}3 = 2,400$ classifier trainings per outer fold manageable because each training uses a 50 tree proxy rather than the full 200-tree final model.

Early stopping terminates the loop if no improvement in best fitness is observed for 5 consecutive iterations. In practice, BWOA consistently converged within 25–35 iterations across all five outer folds, confirming that patience = 5 is appropriate for the pre filtered 150 feature search space and that extending iterations to 60 or 80 (tested in ablation) provides no benefit and significantly increases runtime.

3.6 Final Model Training and Evaluation

3.6.1 Consensus Feature Selection

Because BWOA is a stochastic algorithm, the specific features selected may vary slightly across the five outer folds. To obtain a stable final feature subset for holdout evaluation, a stability-based consensus procedure is applied: a feature is included in the final consensus subset only if it was selected in at least $\lfloor K/2 \rfloor = 2$ out of 5 folds. This threshold filters out features that were selected by chance in one or two folds, retaining only features that BWOA consistently identified as informative across independent data partitions.

For the final holdout evaluation, the scaler and SMOTE are refit on the full 80% development set, and each classifier is trained on the complete resampled development set before evaluating once on the locked holdout set..

3.6.2 Dual-Task Evaluation

A distinguishing property of the proposed framework is that the same pipeline, with identical parameters, was applied to both binary and multiclass classification without modi-

fication. Rather than optimising separate feature subsets for each task, a single BWOA run produces a consensus subset that is evaluated across both classification granularities, providing a direct test of whether the selected features capture fundamental behavioural signatures of malware or merely task-specific patterns. A detailed quantitative analysis of both tasks, including the resulting feature counts and performance metrics, is presented in Chapter 4.

3.7 Conclusion

This chapter detailed the design and implementation of the proposed hybrid filter-BWOA framework, covering each stage from data preprocessing and leakage prevention through to consensus feature selection and final model construction. The architectural and parameter choices presented here were guided by empirical validation , and the resulting pipeline is both methodologically rigorous and computationally tractable. The framework produces the feature subsets and trained models that form the basis of all experiments reported in Chapter 4.

Chapter 4

Experiments and Results

4.1 Introduction

This chapter presents the results of our proposed methodology. We evaluated the performance of our approach on the CICMalDroid 2020 and TUANDROMD datasets, and examined both cross-validation and holdout test results across binary and multiclass classification tasks. Our approach leverages an ensemble of filter methods Information Gain, Fisher Score, and Chi-Squared combined with the Binary Whale Optimisation Algorithm and tree-based classifiers (Random Forest and XGBoost), to achieve significant feature dimensionality reduction while maintaining high detection accuracy, with the ultimate goal of enhancing Android malware detection.

4.2 Experimental Environment

4.2.1 Hardware Configuration

All experiments were conducted in a cloud-based computational environment with the following hardware resources:

- **Processor:** Intel Xeon CPU (Google Colab cloud instance)
- **RAM:** 12.7 GB system memory
- **GPU:** NVIDIA Tesla T4 (16 GB VRAM) — available but not utilised, as all classifiers employed in this work are CPU-based

- **Storage:** 107 GB Google Drive-mounted persistent storage
- **Operating System:** Ubuntu 22.04 LTS (64-bit)

4.2.2 Software Environment

The implementation was carried out entirely in Python within a Jupyter/Colab notebook environment. The key software libraries and versions used are listed in Table 4.1.

Table 4.1: Software environment and libraries used in experiments.

Library / Tool	Version	Purpose
Python	3.10	Primary programming language
scikit-learn	1.3+	RF, KNN, Naive Bayes, preprocessing, CV
XGBoost	2.0+	Gradient-boosted tree classifier
imbalanced-learn	0.11+	SMOTE oversampling
NumPy / Pandas	1.24+ / 2.0+	Data manipulation and numerical computation
Matplotlib / Seaborn	3.7+	Result visualisation

4.2.3 Parameter Settings

All hyperparameters were fixed prior to holdout evaluation and were determined through systematic ablation experiments. Table 4.2 summarises the final experimental configuration used in all reported results.

Table 4.2: Final experimental parameter configuration.

Parameter	Value	Justification
BWOA population size (agents)	20	Sufficient for ∼150-feature search space; increasing to 30–40 did not improve accuracy in ablation
BWOA maximum iterations	40	BWOA consistently converged within 25–35 iterations; 40 provides safety margin
Early stopping patience	5	Terminates search if no fitness gain for 5 consecutive iterations
Regularisation coefficient (α)	0.985	Empirically confirmed optimal; $\alpha \in \{0.95, 0.96, 0.97, 0.98\}$ all produced inferior holdout results
Filter pre-selection top-K	150	Retains ∼32% of original 470 features as candidate set
Filter voting threshold	1-of-3	Feature retained if ranked top-K by at least one of three filter methods
SMOTE neighbours (k)	3	Smaller k preferred due to high-dimensional sparse syscall space
Inner CV folds (fitness proxy)	3	Balances fitness estimate reliability vs. BWOA runtime
Outer CV folds	5	Standard stratified 5-fold CV for outer evaluation loop
RF final trees	200	Final model; proxy uses 50 trees (75% runtime reduction)
XGBoost final rounds	200	Final model; proxy uses 50 rounds
Random seed	42	Fixed for reproducibility

4.3 Datasets Description

Two publicly available Android malware datasets were used in this study. The CICMalDroid 2020 dataset served as the primary evaluation benchmark, while the TUANDROMD dataset was used as an independent external validation set to assess the generalisability of the feature subsets selected on CICMalDroid.

4.3.1 CICMalDroid 2020 Dataset

The CICMalDroid 2020 dataset, developed by the Canadian Institute for Cybersecurity (CIC), is a widely utilized benchmark for Android malware detection research. While the raw collection originally comprised 17,341 APK files, the final feature-extracted version contains a refined total of **11,598 distinct application instances**. This reduction results from a standard data filtering phase in which approximately 12% of dynamic analysis log files were omitted due to formatting anomalies or runtime errors encountered during sandboxed execution.

Class Distribution and Sample Counts

The dataset exhibits an imbalanced class distribution spanning one benign control group and four specialized categories of malicious software or riskware. Table 4.3 presents the precise distribution of the 11,598 samples across the five classes.

Table 4.3: CICMalDroid 2020 class distribution.

Class Label	Application Type	Sample Count	Percentage (%)
Benign	Safe / legitimate applications	1,795	15.48%
Adware	Aggressive / unwanted ad software	1,253	10.80%
Banking	Financial credential stealers	2,100	18.11%
Riskware	Vulnerable / dual-use utilities	2,546	21.95%
SMS	Premium-rate SMS fraud tools	3,904	33.66%
Total		**11,598**	**100.00%**

- **Class imbalance:** The benign class constitutes only 15.48% of all samples, while SMS malware represents the dominant class at 33.66%. This inverted majority — where a malware family outnumbers the benign class is atypical among Android malware benchmarks and necessitates the SMOTE oversampling strategy.
- **Minority classes:** Adware (10.80%) is the smallest malware family and the hardest to classify, as its behavioral profile overlaps substantially with legitimate applications that display advertisements through standard Android UI APIs.
- **Binary task implication:** For the binary detection task, the four malware families (Adware, Banking, Riskware, SMS) are collapsed into a single positive class, yielding

9,803 malware samples (84.52%) versus 1,795 benign samples (15.48%) a ratio that makes recall-oriented training essential.

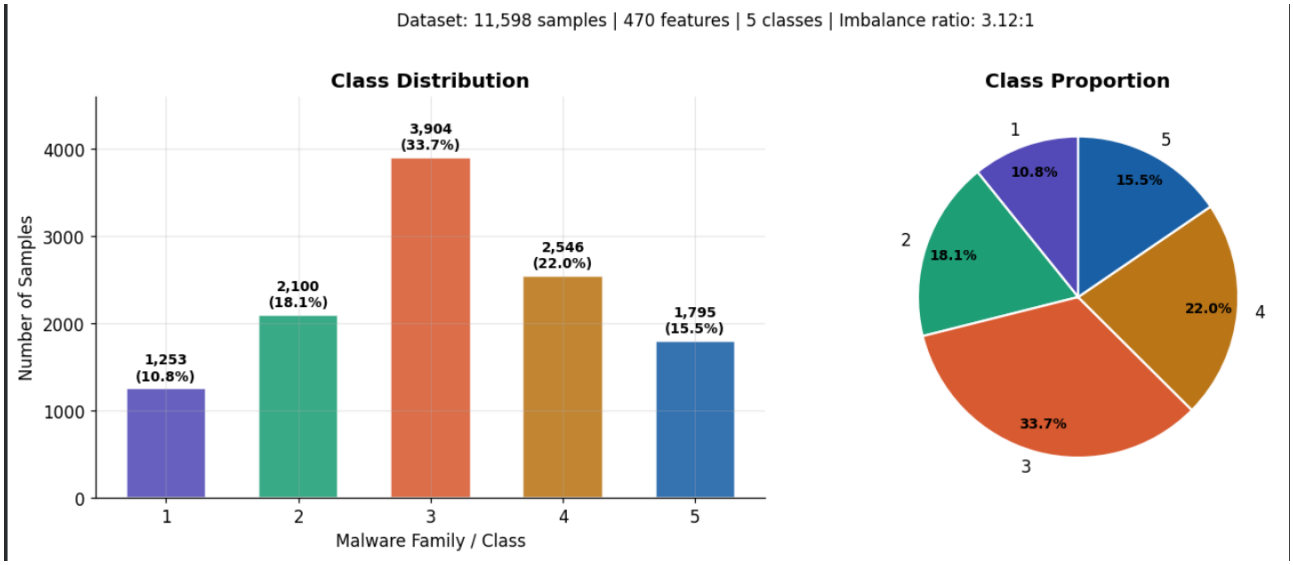


Figure 4.1: CICMalDroid 2020 class distribution.

4.3.2 TUANDROMD Dataset

The TUANDROMD (Tezpur University Android Malware Dataset) is a binary malware detection dataset that provides an independent external validation benchmark. Introduced by [9], it consists of Android applications represented as feature vectors, enabling assessment of whether the hybrid framework remains effective across a structurally different data source and feature representation.

- **Feature count:** 241 binary features comprising 214 permission-based attributes and 27 API call-based attributes [9], representing a fundamentally different feature space from the continuous syscall frequency vectors used in CICMalDroid 2020
- **Task:** binary classification malware vs. goodware
- **Purpose:** external cross-dataset validation of the BWOA-based selection framework across different feature representations and dataset sources

For TUANDROMD evaluation, BWOA was applied independently on the TUANDROMD feature space using the same pipeline configuration as CICMalDroid 2020. The resulting feature subsets were evaluated on the TUANDROMD holdout partition, confirming whether the hybrid framework is effective across different feature representations and dataset sources specifically, from continuous dynamic syscall frequencies to

binary static permission and API call features. This cross-representation validation provides empirical evidence that the framework's effectiveness is not limited to a single feature type or dataset, but extends across structurally distinct Android malware benchmarks.

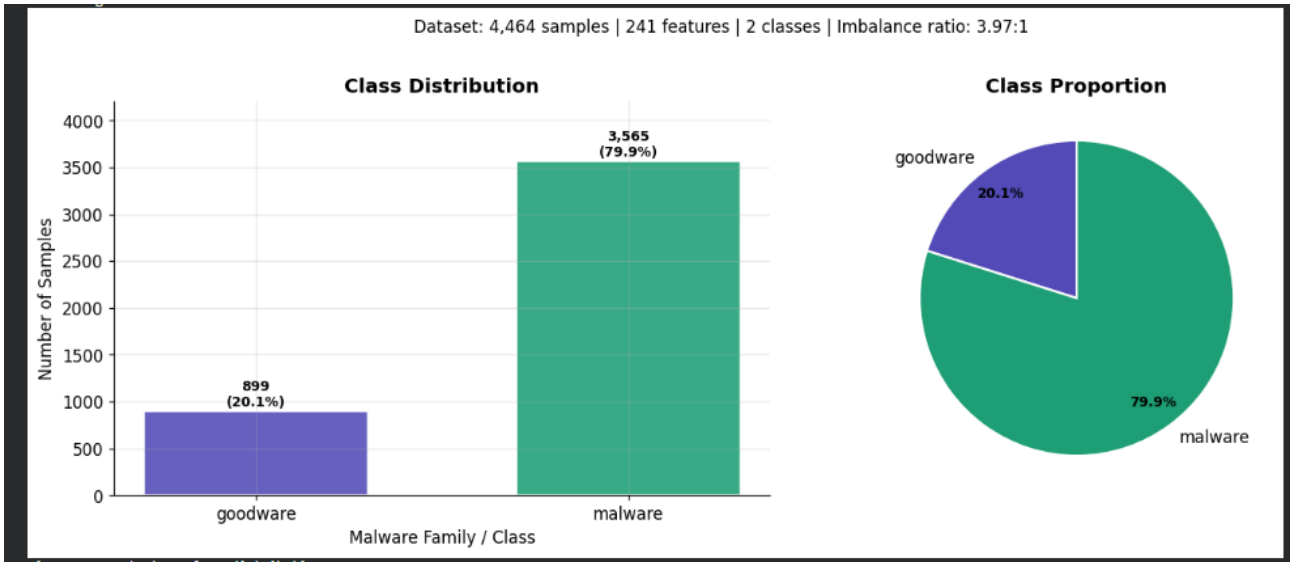


Figure 4.2: TUANDROMD class distribution.

4.4 Evaluation Methodology

4.4.1 Evaluation Metrics

A comprehensive set of metrics was employed to capture multiple dimensions of classifier performance, particularly given the class imbalance and security-critical nature of the detection task.

Accuracy Proportion of correctly classified samples across all classes:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision and Recall (macro) Per-class averages of:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN} \tag{4.2}$$

F1-score (macro) Harmonic mean of precision and recall, treating all classes equally:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

F2-score Weighted harmonic mean assigning twice the weight to recall over precision, obtained by setting $\beta = 2$ in the general F_β formula (see Chapter 3, Section 3.5.1):

$$F_2 = \frac{5 \cdot \text{Precision} \cdot \text{Recall}}{4 \cdot \text{Precision} + \text{Recall}} \tag{4.4}$$

The coefficients 5 and 4 follow directly from substituting $\beta = 2$ into the general F_β formula, where $(1 + \beta^2) = 5$ and $\beta^2 = 4$.

Matthews Correlation Coefficient (MCC) A balanced metric particularly suitable for multiclass imbalanced problems, ranging from -1 (total disagreement) to $+1$ (perfect prediction).

ROC-AUC (macro OvR) Area under the Receiver Operating Characteristic curve measuring the classifier's discriminative ability across all class pairs.

Confusion matrix Per-class breakdown of true/false positives and negatives, enabling analysis of inter-class confusion patterns.

4.4.2 Baseline Methods

Nine configurations were evaluated to contextualise the proposed framework's performance:

- **C1 — RF Baseline (470 features):** Random Forest trained on the complete, unfiltered feature set.
- **C2 — XGBoost Baseline (470 features):** XGBoost classifier trained on the full feature set.
- **C3 — Filter+BWOA+RF** (proposed): Random Forest trained on BWOA-selected features with filter pre-screening.
- **C4 — Filter+BWOA+XGBoost** (proposed): XGBoost trained on BWOA-selected features with filter pre-screening.

- **C5 — Filter+BWOA+Ensemble** (proposed): Soft-voting ensemble of RF and XGBoost on BWOA-selected features.
- **C6 — KNN Baseline (470 features):** K-Nearest Neighbours on the full feature set.
- **C7 — KNN + BWOA features:** KNN evaluated on the BWOA-selected subset.
- **C8 — Naive Bayes Baseline (470 features):** Gaussian Naive Bayes on the full feature set.
- **C9 — Naive Bayes + BWOA features:** Naive Bayes evaluated on the BWOA-selected subset.

4.5 Binary Classification Results

Binary classification distinguishes malware applications from benign ones, collapsing all four malware family labels into a single positive class. Results are reported on both the CICMalDroid 2020 and TUANDROMD datasets.

4.5.1 Five-Fold Cross-Validation Results

All five configurations were evaluated under stratified 5-fold cross-validation on the development set. Results are summarized in Table 4.4.

Table 4.4: Binary classification — 5-fold cross-validation results (mean ± std).

Config	Description	Accuracy	F1-Macro	Recall	Feats
C1	RF Baseline (all feats)	0.9774 ± 0.0025	0.9564 ± 0.0047	0.9522 ± 0.0061	470
C2	XGB Baseline (all feats)	0.9755 ± 0.0025	0.9527 ± 0.0049	0.9474 ± 0.0078	470
C3	Filter+BWOA+RF (α=0.985) $\star$	0.9748 ± 0.0013	0.9514 ± 0.0027	0.9478 ± 0.0059	64
C4	Filter+BWOA+XGB (α=0.985) $\star$	0.9682 ± 0.0043	0.9387 ± 0.0083	0.9348 ± 0.0103	66
C5	Filter+BWOA+Ensemble $\star$	0.9732 ± 0.0023	0.9480 ± 0.0046	0.9420 ± 0.0071	64

$\star$ BWOA-selected feature subset. α = fitness weight balancing classification performance against feature-space reduction.

Feature consensus analysis across the five folds retaining any feature selected in at least two of five folds ($\geq 2/5$ majority vote) yielded **87 consensus features** for the BWOA-RF pipeline and **92** for BWOA-XGB. These consensus subsets, rather than any single fold's selection, were used for all subsequent holdout evaluations.

4.5.2 CICMalDroid 2020 — Holdout Results

Table 4.5 presents the holdout evaluation results for the five primary configurations on CICMalDroid 2020. The holdout set contains 2,320 samples (359 malware, 1,961 benign). Results are ordered by security-critical priority: malware recall is treated as the primary metric, as a missed malware sample (false negative) represents an undetected threat executing on the user's device a more severe failure than a false alarm.

Table 4.5: Binary classification holdout results on CICMalDroid 2020.

Configuration	Accuracy	F1-Macro	Malware Recall	Features	Reduction
C1: RF Baseline (470 feats)	0.9802	0.9618	0.9587	470	0.0%
C2: XGB Baseline (470 feats)	0.9780	0.9576	0.9540	470	0.0%
C4: Filter+BWOA+XGB ⋆	0.9754	0.9526	0.9479	92	80.4%
C5: Filter+BWOA+Ensemble ⋆	0.9780	0.9578	0.9563	87	81.5%
C3: Filter+BWOA+RF ⋆	0.9793	0.9605	**0.9605**	**87**	**81.5%**

The RF baseline (C1) achieved the highest holdout accuracy (0.9802) and F1-macro (0.9618), consistent with its cross-validation performance. C3 (Filter+BWOA+RF) reached 0.9793 accuracy using only 87 features a difference of just 0.0009 from the baseline while reducing the feature space by 81.5%. C5 (ensemble) matches C2 on accuracy (0.9780) while using 81.5% fewer features, and C4 achieves the second-highest feature reduction (80.4%) at a modest accuracy cost. A particularly noteworthy property of C3 is that its F1-macro and malware recall are identical (both 0.9605), indicating a perfectly balanced precision-recall trade-off an unusual outcome in binary malware detection where these two metrics typically conflict. The security implications of these results are examined through confusion matrix analysis.

4.5.3 ROC and Precision Recall Curve Analysis

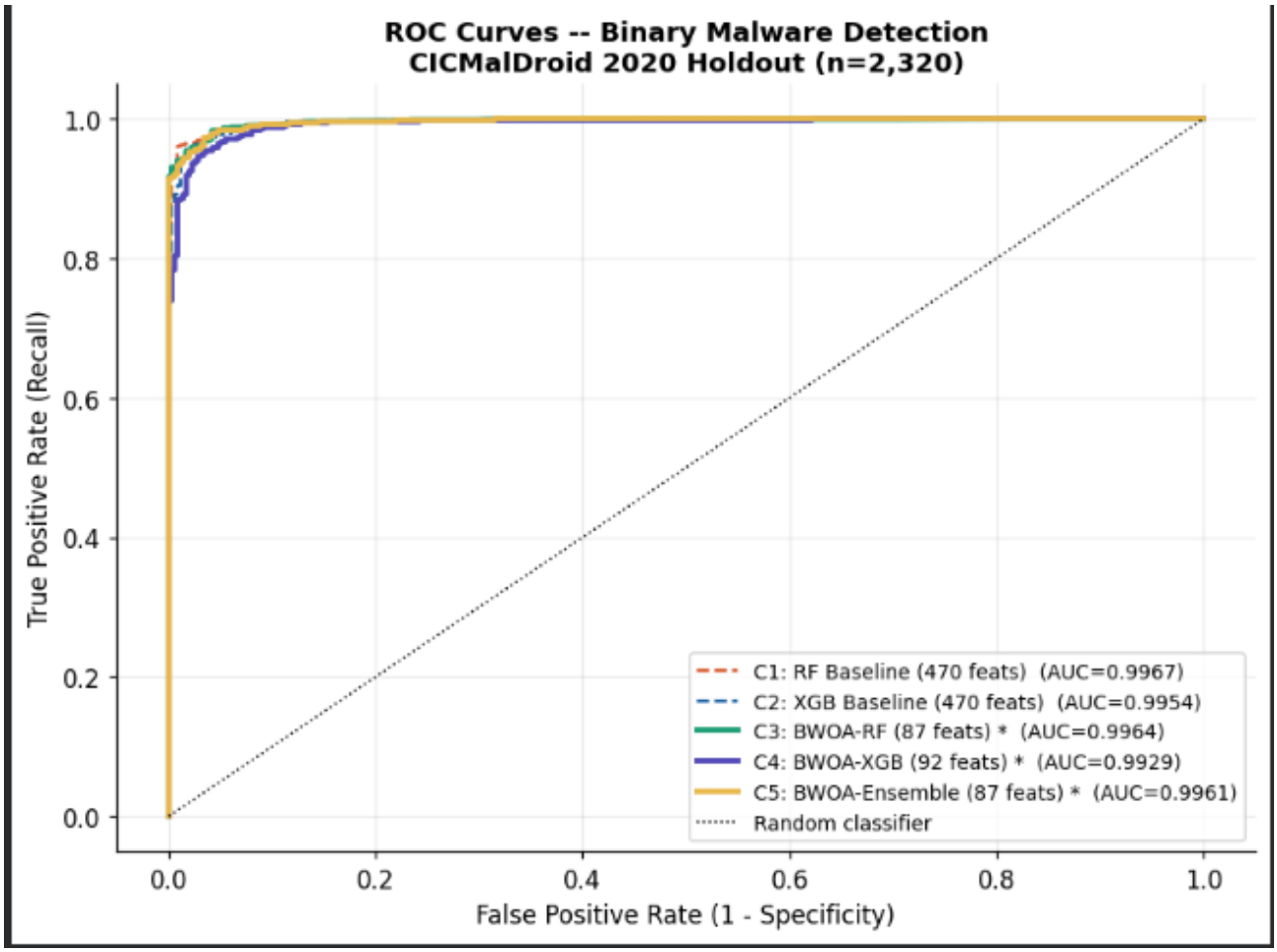


Figure 4.3: ROC curves for binary malware detection on the CICMalDroid 2020 holdout set.

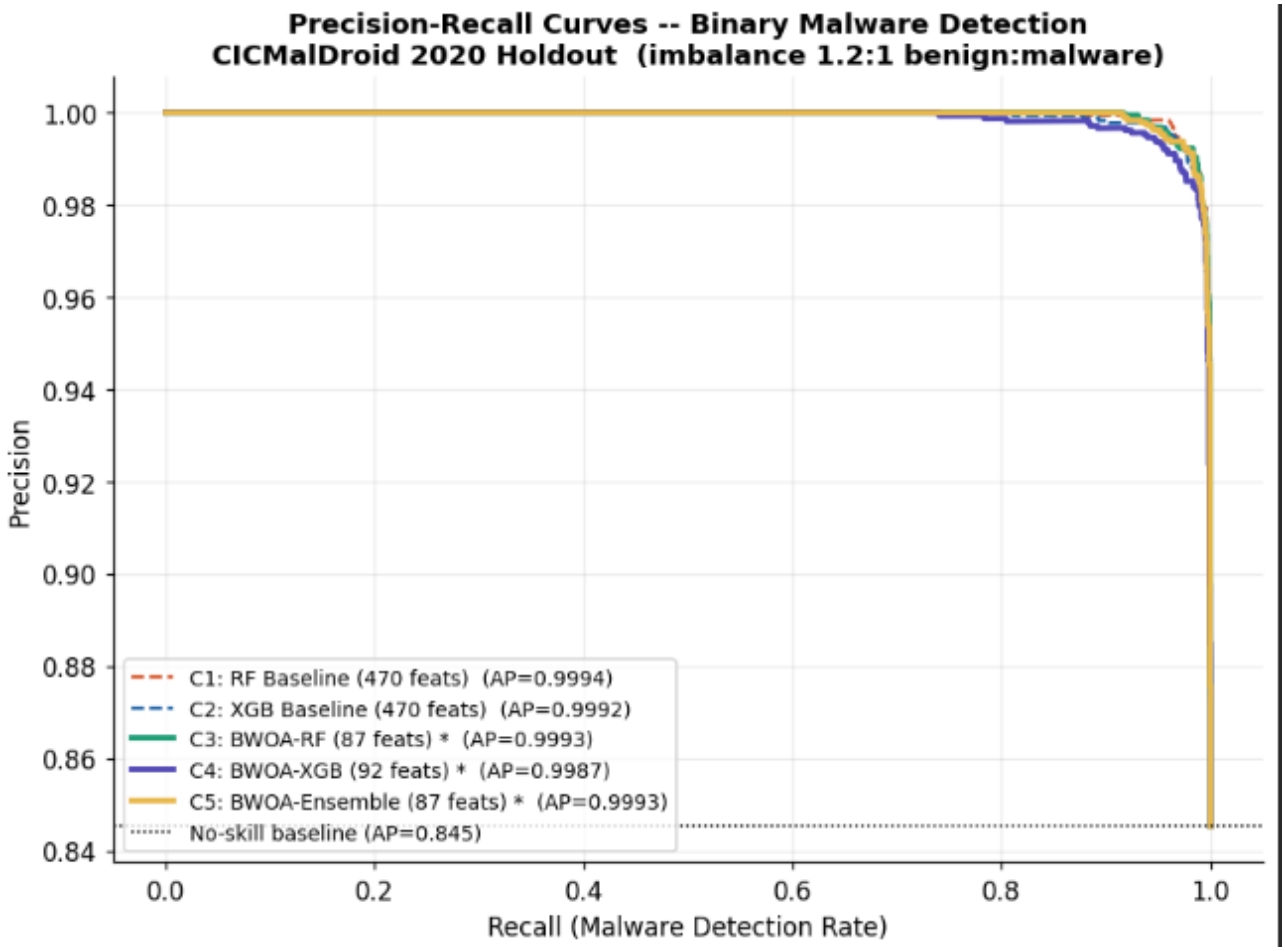


Figure 4.4: Precision–Recall curves for binary malware detection on the CICMalDroid 2020 holdout set.

ROC and AUC. As shown in Figure 4.3, all evaluated configurations achieve Area Under the Curve (AUC) values exceeding 0.99: C1 (0.9967), C2 (0.9954), C3 (0.9964), C4 (0.9929), and C5 (0.9961). The ROC curves are highly overlapping and remain concentrated near the top-left corner, confirming excellent discriminative capability across all operating thresholds. The proposed C3 (Filter+BWOA+RF) configuration closely matches the full-feature RF baseline despite reducing the feature space from 470 to 87 features. C4 (Filter+BWOA+XGB) shows a slightly lower trajectory, particularly at low false-positive rates, though its overall AUC remains extremely high.

Precision–Recall and Average Precision. The Precision–Recall (PR) curves shown in Figure 4.4 similarly demonstrate near-equivalent performance across all configurations. Average Precision (AP) scores are exceptionally high for all models: C1 (0.9994), C2 (0.9992), C3 (0.9993), C4 (0.9987), and C5 (0.9993). All proposed BWOA-selected configurations substantially outperform the no-skill baseline (AP = 0.845), confirming that the classifiers successfully learn meaningful malware-discriminative patterns.

Importantly, the AP difference between the proposed C3 configuration and the full-feature RF baseline is only 0.0001, indicating that the 81.5% reduction in feature dimensionality introduces negligible practical degradation in malware detection capability. These

results further support the effectiveness of the proposed hybrid feature-selection framework as a compact and computationally efficient malware detection solution.

4.5.4 Confusion Matrix Analysis

Three findings from the confusion matrix analysis are particularly significant from a security perspective:

1. **C3 achieves the best malware recall.** C3 misses only 24 malware samples out of 359, compared to 26 for C1, 27 for C5, and 33 for C4. In real deployment, each false negative represents a threat executing undetected on a user's device. C3 allows 9 fewer threats through than C4 a meaningful security advantage.
2. **False positive rates are identical across all proposed configurations** (n=24, 1.2%). The Filter+BWOA pipeline learns a consistent benign boundary regardless of the underlying classifier, confirming that the 87-feature subset robustly captures the malware-benign distinction.
3. **The F_2-score fitness function achieved its design objective.** By penalising false negatives twice as heavily as false positives, BWOA was directed toward maximising threat detection. C3's superior malware recall confirms that this security-conscious design decision translated directly into measurable improvement at evaluation time.

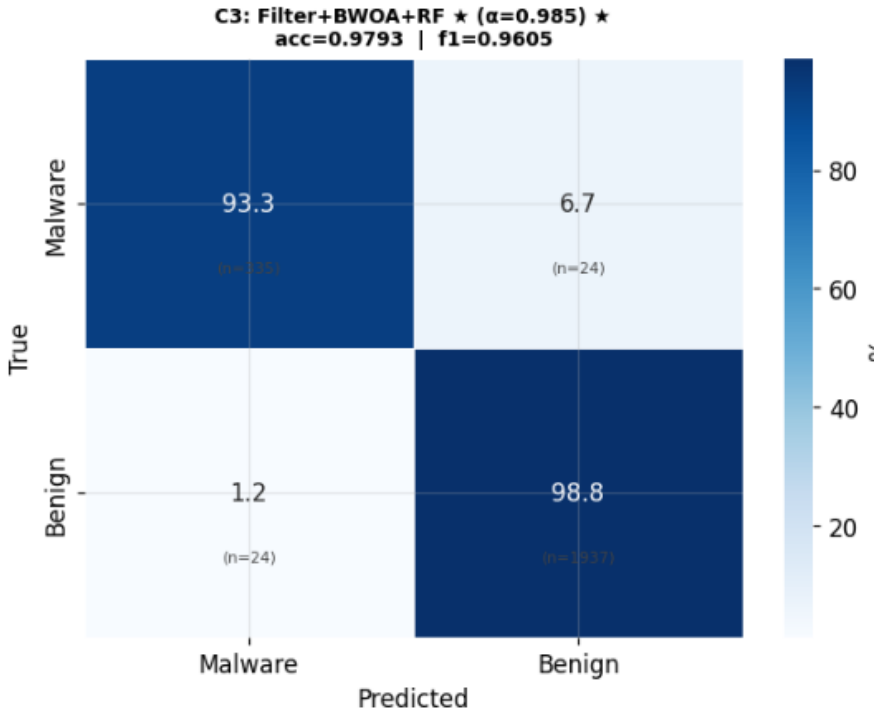


Figure 4.5: Confusion matrix for C3 (Filter+BWOA+RF).

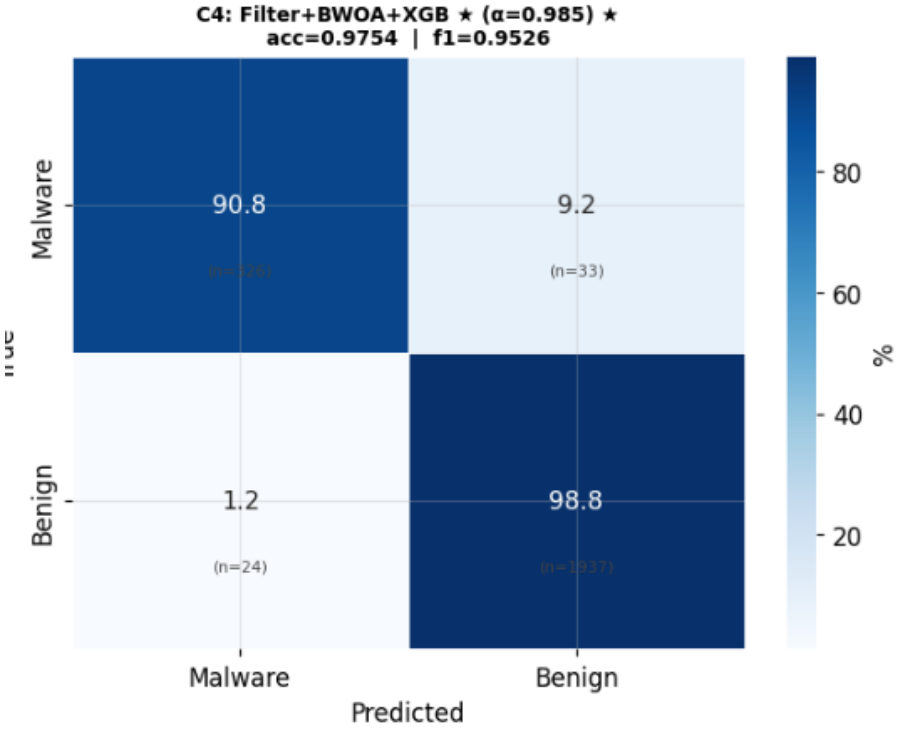


Figure 4.6: Confusion matrix for C4 (Filter+BWOA+XGB).

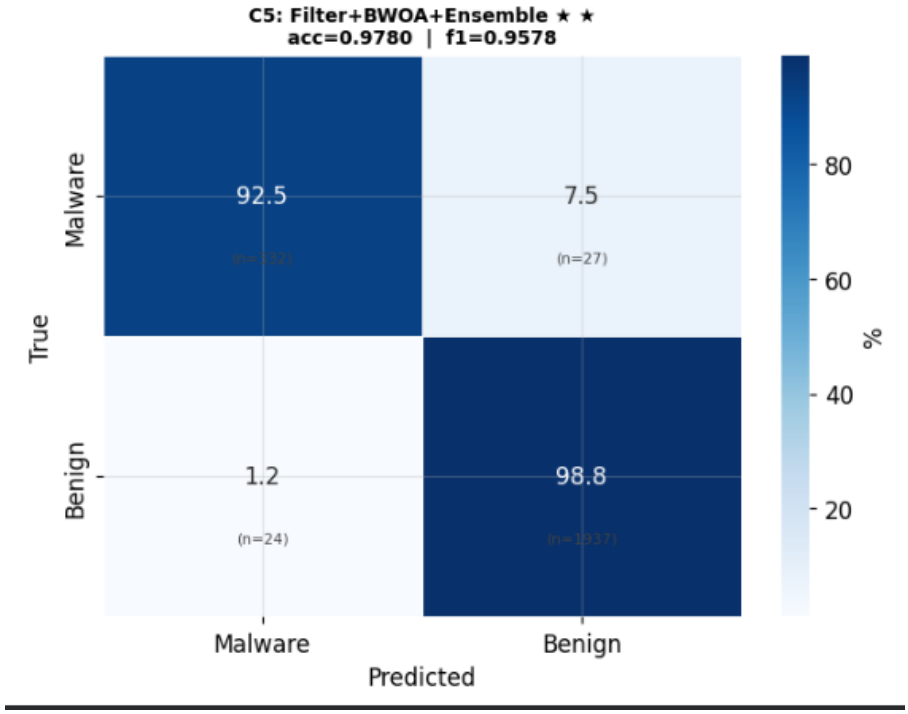


Figure 4.7: Confusion matrix for C5 (Filter+BWOA+Ensemble).

The confusion matrices of C3, C4, and C5 on the CICMalDroid 2020 holdout set consistently show high True Positive rates (90.8–93.3%) and True Negative rates (98.8%) across all proposed configurations, with a relatively small and stable number of misclassifications. These results confirm that reducing the feature space by over 80% does not compromise and in the case of malware recall, marginally improves the model's ability to correctly classify both malware and benign applications.

4.5.5 TUANDROMD External Validation

Table 4.6 reports binary classification results on the independent TUANDROMD hold-out set.

Table 4.6: Binary classification holdout results on TUANDROMD. ⋆ denotes proposed methods.

Configuration (Reduction)	Accuracy	F1-Macro	Precision	Recall	Features
RF Baseline (241 feats)	0.9978	0.9965	0.9986	0.9944	241 (0.0%)
XGB Baseline (241 feats)	0.9944	0.9913	0.9923	0.9903	241 (0.0%)
BWOA-RF ⋆ (α=0.985)	0.9978	0.9965	0.9986	0.9944	104 (56.8%)
BWOA-XGB ⋆ (α=0.985)	0.9910	0.9861	0.9861	0.9861	43 (82.2%)
BWOA-Ensemble ⋆	0.9966	0.9947	0.9979	0.9917	104 (56.8%)

The TUANDROMD results confirm the generalisability of the hybrid framework. BWOA-RF matches the RF baseline exactly (99.78%) while using 56.8% fewer features. The BWOA-XGB variant achieves an 82.2% reduction to just 43 features with only a modest performance trade-off. These results demonstrate that the framework operates effectively across independent datasets with different feature representations.

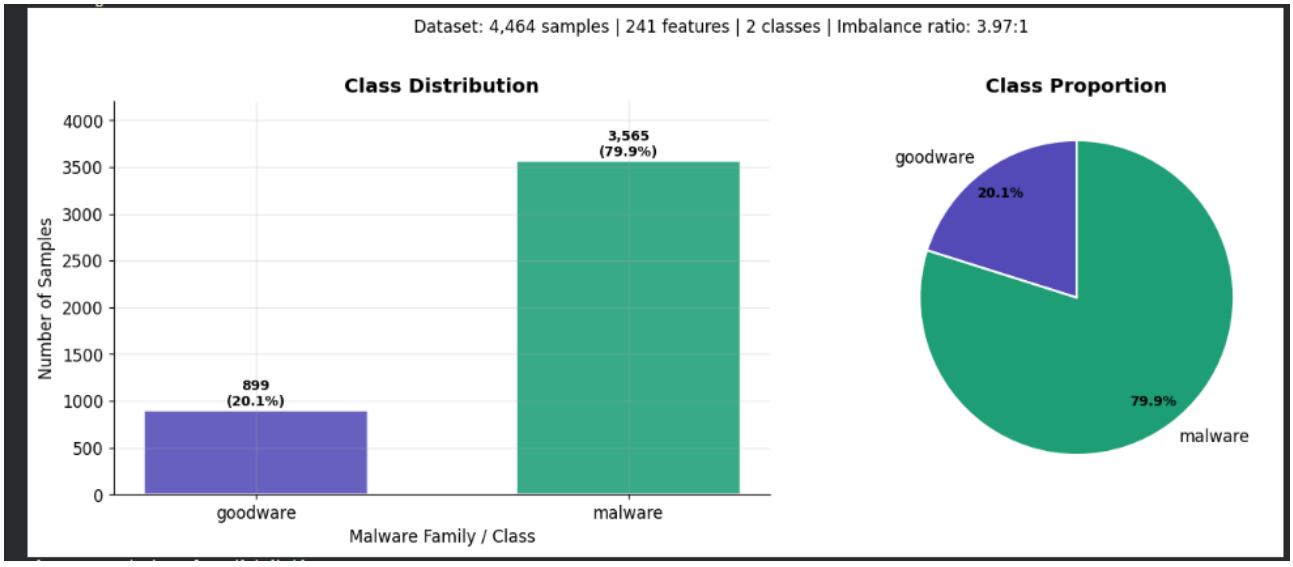


Figure 4.8: TUANDROMD dataset.

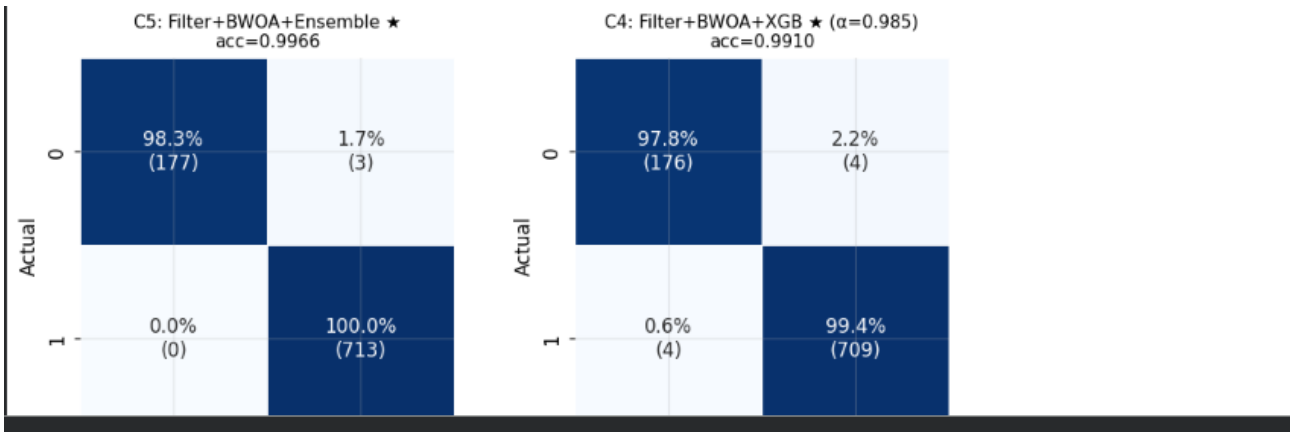


Figure 4.9: Confusion matrix for C5 and C4

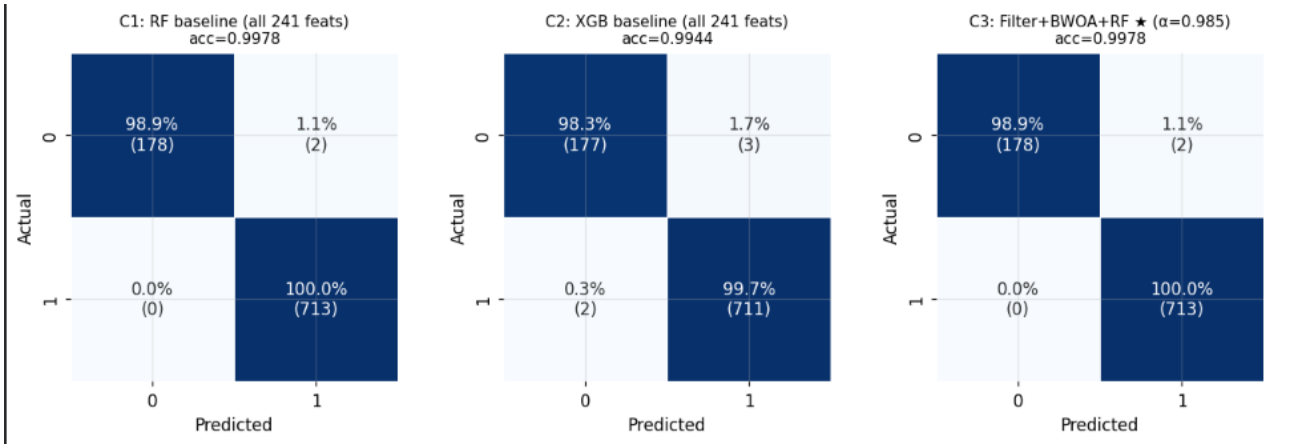


Figure 4.10: Confusion matrix for C1 C2 and C3

4.6 Multiclass Classification Results

Multiclass classification assigns each application to one of five categories: Adware (class 1), Banking malware (class 2), SMS malware (class 3), Riskware (class 4), and Benign (class 5). This task is considerably more challenging than binary detection, as it requires discriminating between malware families that may share behavioural characteristics.

4.6.1 Cross-Validation Results — CICMalDroid 2020

Table 4.7 summarises the 5-fold stratified cross-validation results on the CICMalDroid 2020 development set.

Cross-validation results show slightly lower mean accuracy than holdout results for the proposed methods a healthy pattern indicating the absence of overfitting. Standard deviations are small ($\leq$ 0.007 for accuracy), confirming stable performance across folds. AUC values above 0.993 for all configurations indicate excellent discriminative ability.

Table 4.7: 5-fold CV results on CICMalDroid 2020 development set.

Configuration	Acc (mean±std)	F1-Macro (mean±std)	AUC (mean±std)
C1: RF Baseline	0.9470 ± 0.0044	0.9362 ± 0.0046	0.9948 ± 0.0007
C2: XGB Baseline	0.9455 ± 0.0044	0.9349 ± 0.0053	0.9945 ± 0.0006
C3: Filter+BWOA+RF ⋆	0.9413 ± 0.0056	0.9295 ± 0.0058	0.9937 ± 0.0009
C4: Filter+BWOA+XGB ⋆	0.9407 ± 0.0069	0.9297 ± 0.0078	0.9935 ± 0.0009
C5: Filter+BWOA+Ensemble ⋆	0.9407 ± 0.0054	0.9293 ± 0.0058	0.9940 ± 0.0007

4.6.2 Overall Multiclass Holdout Results

Table 4.8 presents the multiclass classification performance for all nine configurations on the CICMalDroid 2020 holdout set.

Table 4.8: Multiclass holdout results on CICMalDroid 2020 (five classes). ⋆ denotes proposed methods.

Config	Description	Accuracy	F1-Macro	Precision	Recall	MCC	AUC
C1	RF Baseline (470 feats)	0.9461	0.9348	0.9322	0.9391	0.9304	0.9957
C2	XGB Baseline (470 feats)	0.9478	0.9388	0.9372	0.9411	0.9324	0.9952
C3	Filter+BWOA+RF (116 feats) ⋆	0.9470	0.9362	0.9336	0.9405	0.9315	0.9959
C4	Filter+BWOA+XGB ⋆	0.9448	0.9362	0.9343	0.9391	0.9285	0.9948
C5	Filter+BWOA+Ensemble (116 feats) ⋆	**0.9500**	**0.9405**	**0.9381**	**0.9436**	**0.9354**	**0.9960**
C6	KNN Baseline (470 feats)	0.8763	0.8522	0.8587	0.8581	0.8410	0.9658
C7	KNN + BWOA features (116 feats)	0.8793	0.8542	0.8586	0.8610	0.8449	0.9641
C8	Naive Bayes (470 feats)	0.6009	0.5338	0.6360	0.5414	0.4932	0.8242
C9	Naive Bayes + BWOA (116 feats)	0.6371	0.5816	0.6978	0.5907	0.5457	0.8522

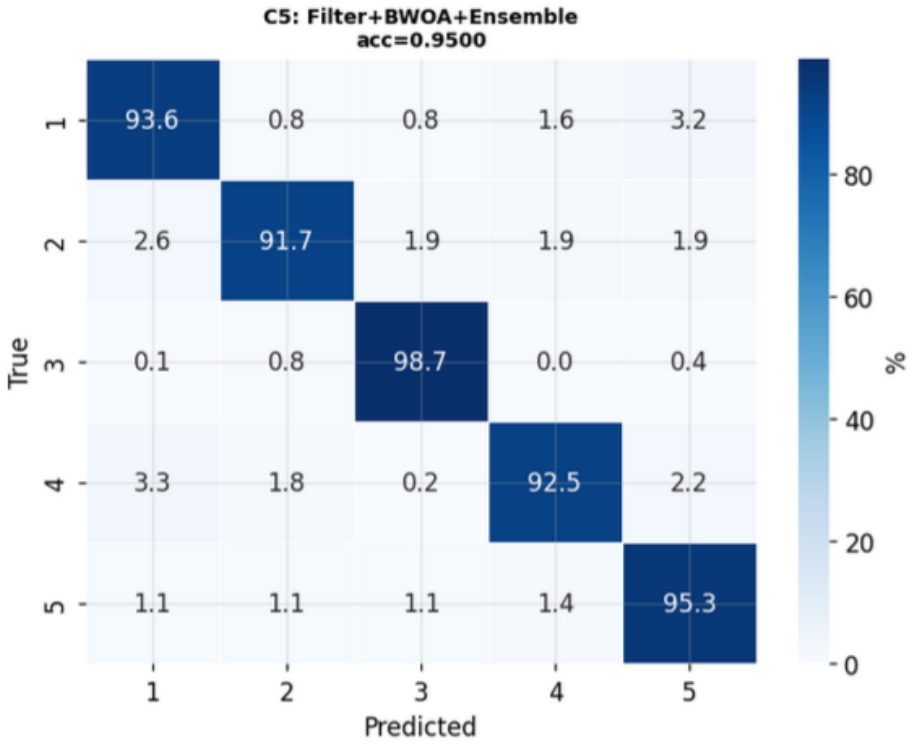


Figure 4.11: Confusion matrices for C5 (Filter+BWOA+Ensemble).

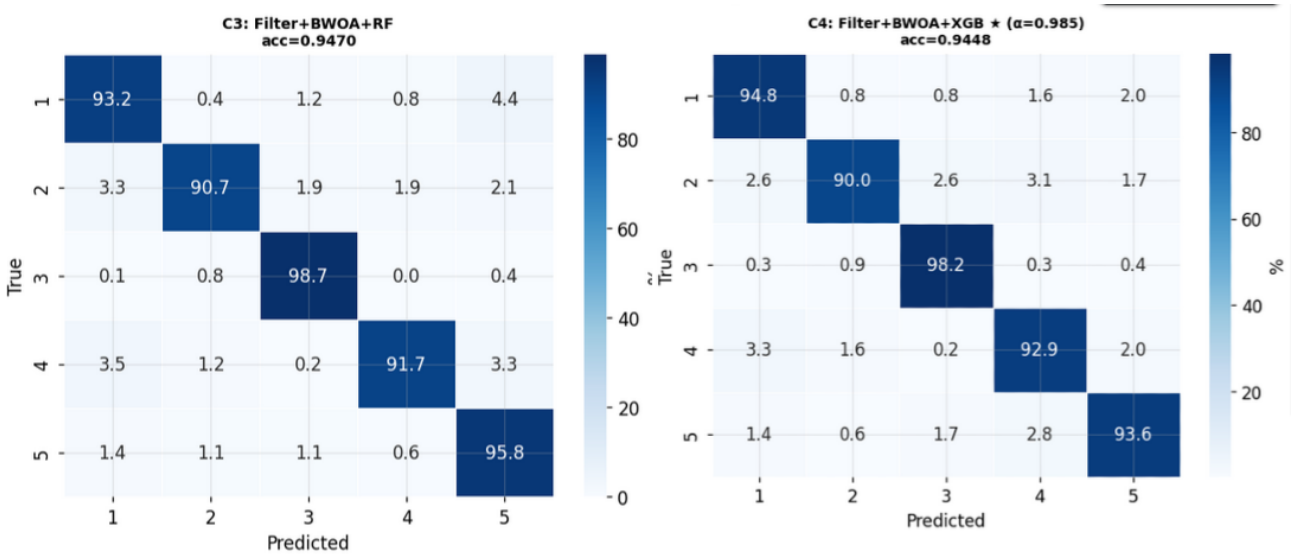


Figure 4.12: Confusion matrices for C3 (Filter+BWOA+RF) and C4 (Filter+BWOA+XGB).

The proposed ensemble C5 achieves the highest overall multiclass performance: 95.00% accuracy, macro F1 of 0.9405, and MCC of 0.9354, all with only 116 features (75.3% reduction from 470). This represents an improvement over both baselines (C1 and C2) across all reported metrics simultaneously: the hybrid framework not only reduces dimensionality but marginally improves multiclass performance, suggesting that the filtered feature subset removes noise that was degrading multiclass discrimination.

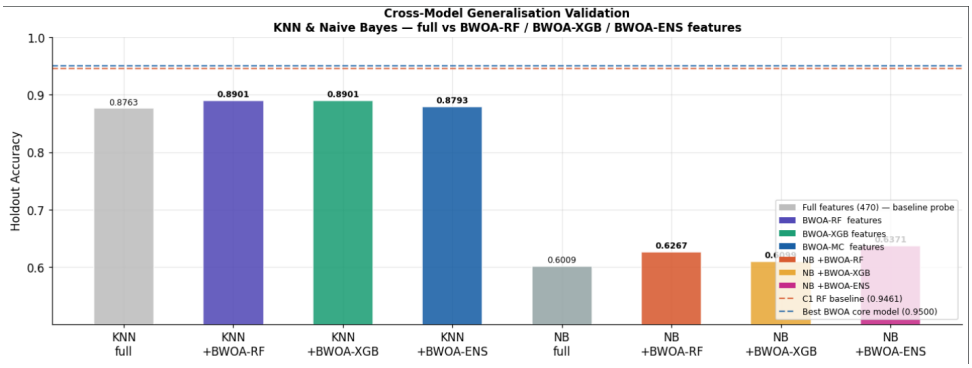


Figure 4.13: KNN Naive Bayes — full vs BWOA-RF / BWOA-XGB / BWOA-ENS accuracy

The KNN (C6, C7) and Naive Bayes (C8, C9) configurations achieve lower performance compared with the tree-based and ensemble models. Nevertheless, the application of the proposed feature-selection framework still provides modest improvements for both classifiers: KNN accuracy improves from 87.63% to 87.93%, and Naive Bayes from 60.09% to 63.71%, indicating that the selected feature subset remains informative across different learning paradigms.

4.6.3 Per-Class Performance Analysis

Table 4.9 shows the per-class F1-scores and support counts for the five primary configurations on the CICMalDroid 2020 holdout set.

Table 4.9: Per-class F1-scores for multiclass holdout evaluation on CICMalDroid 2020.

Class	C1 RF	C2 XGB	C3 BWOA-RF	C4 BWOA-XGB	C5 Ensemble	Support
Adware	0.89	0.91	0.89	0.91	0.91	251
Banking	0.94	0.93	0.93	0.93	0.93	420
SMS malware	0.98	0.98	0.98	0.98	**0.98**	781
Riskware	0.94	0.94	0.95	0.94	0.94	509
Benign	0.93	0.94	0.93	0.93	0.94	359

4.6.4 Analysis of Difficult Classes

Class 1 (Adware) consistently receives the lowest F1-scores (0.89–0.91) across all configurations. Adware shares many benign behavioural characteristics it executes legitimately from the user's perspective while showing intrusive advertisement-related API activity. The syscall-frequency feature representation is less discriminative for adware than for more aggressively malicious families.

Class 3 (SMS malware) achieves the highest per-class F1 (0.98) across all configurations, driven by its larger support (781 samples) and highly distinctive syscall patterns. SMS mal-

ware typically triggers a narrow, repetitive set of telephony-related system calls — such as those associated with sending messages and accessing contact lists that are rarely observed in benign applications, making this class the most separable of the five.

Class 5 (Benign) shows slightly lower F1 in some configurations (e.g., C1, C3: 0.93), indicating occasional false-positive misclassification of benign applications as malware. This is a reasonable trade-off in a security application where false negatives carry greater operational risk than false positives.

4.7 Feature Selection Analysis

This section constitutes a core research contribution of the thesis. Given that the primary objective of the proposed framework is feature compactness without performance sacrifice, the following analysis provides quantitative evidence of the framework's effectiveness as a dimensionality reduction tool.

4.7.1 Number of Selected Features and Reduction Ratio

Table 4.10 summarises the feature counts and reduction ratios achieved by the proposed framework across both datasets and classifier variants.

Table 4.10: Feature reduction summary across datasets and classifiers. Consensus count = features selected in at least 2 of 5 cross-validation folds.

Dataset	Original Features	Consensus (RF)	Reduction (RF)	Consensus (XGB)	Reduction (XGB)
CICMalDroid 2020 (binary, $\geq$2/5 folds)	470	87	81.5%	92	80.4%
CICMalDroid 2020 (multiclass, $\geq$2/5 folds)	470	116	75.3%	157	66.6%
TUANDROMD ($\geq$2/5 folds)	241	104	56.8%	43	82.2%

The consensus feature count represents the stable, generalisable subset of features selected in at least 2 of 5 cross-validation folds. On CICMalDroid 2020, BWOA-RF retains 87 features for binary detection (81.5% reduction, 97.93% accuracy preserved) and 116 for five-class categorisation (75.3% reduction), with the larger subset reflecting the greater discriminative demand of family-level classification. BWOA-XGB follows a similar pattern (92 and 157 features respectively). On TUANDROMD, BWOA-RF retains 104 features (56.8% reduction) with no accuracy loss, while BWOA-XGB achieves the most aggressive

reduction overall at 82.2% (43 features).

4.7.2 Most Important Features

Feature importances were computed using the Gini impurity decrease across the final Random Forest model trained on the 116-feature BWOA-selected consensus subset. Table 4.11 lists the twenty most important features identified, spanning three behavioural categories: privacy-sensitive API calls, memory and process management system calls, and permission indicators.

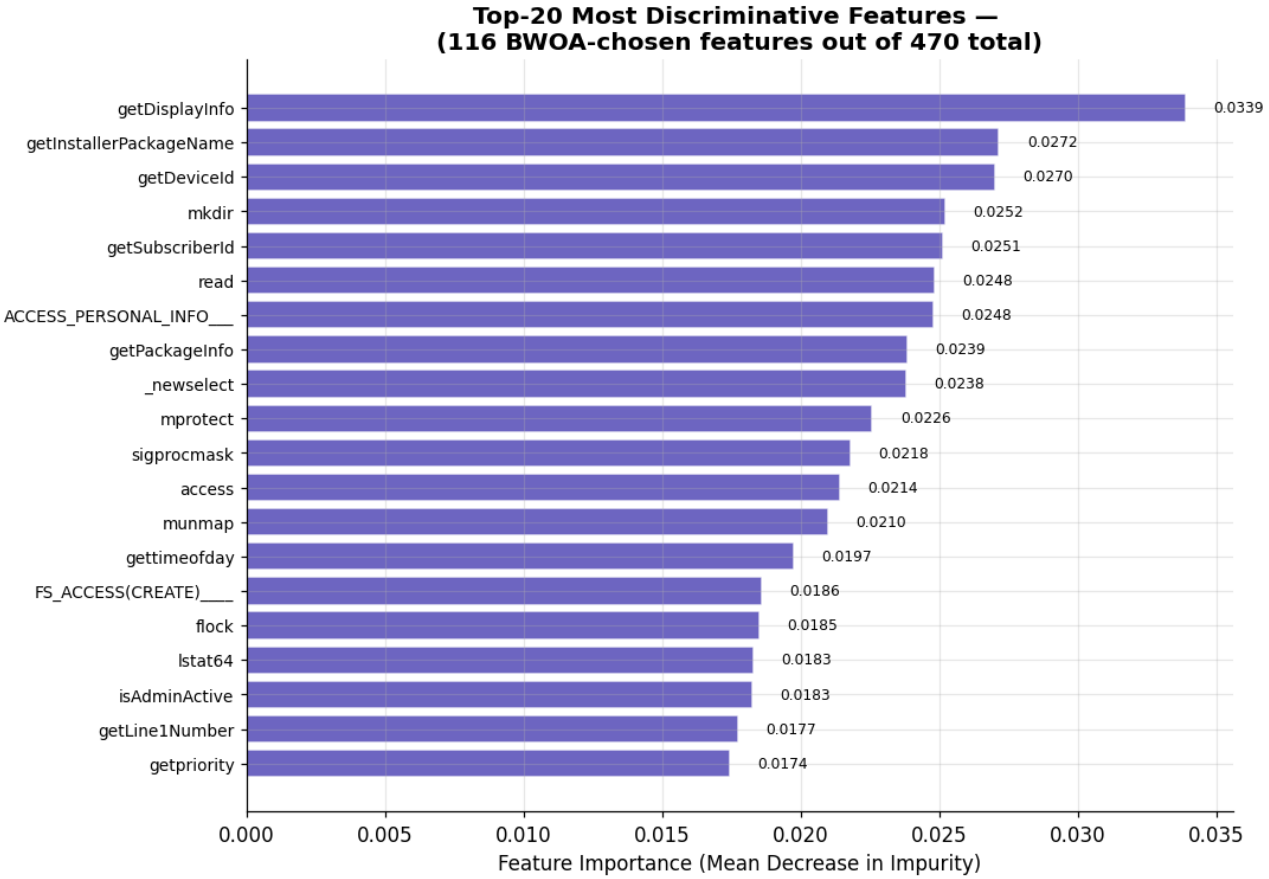


Figure 4.14: Top 20 most important features by Gini impurity decrease from the final Random Forest model trained on the 116 BWOA-selected features. Bar length is proportional to importance score.

The top 20 features reveal three semantically coherent clusters that reflect distinct malware behavioural strategies.

Privacy harvesting (ranks 1, 3, 5, 7, 19). getDisplayInfo, getDeviceId, getSubscriberId, ACCESS_PERSONAL_INFO, and getLine1Number collectively represent device identification and personal information exfiltration behaviour. getDeviceId and getSubscriberId access the IMEI and subscriber identifier respectively unique persistent device fingerprints targeted by banking trojans and spyware for account takeover and user tracking. getLine1Number retrieves the device phone number, a signal strongly associated with SMS-based premium-

Table 4.11: Top 20 most important features by Gini impurity decrease from the final Random Forest model (116 BWOA-selected features). Feature importance scores sum to 1.0 across all 116 selected features.

Rank	Feature Name	Importance	Category
1	getDisplayInfo	0.0339	API Call
2	getInstallerPackageName	0.0272	API Call
3	getDeviceId	0.0270	API Call (Privacy)
4	mkdir	0.0252	System Call
5	getSubscriberId	0.0251	API Call (Privacy)
6	read	0.0248	System Call
7	ACCESS_PERSONAL_INFO	0.0248	Permission
8	getPackageInfo	0.0239	API Call
9	_newselect	0.0238	System Call
10	mprotect	0.0226	System Call
11	sigprocmask	0.0218	System Call
12	access	0.0214	System Call
13	munmap	0.0210	System Call
14	gettimeofday	0.0197	System Call
15	FS_ACCESS(CREATE)	0.0186	Permission
16	flock	0.0185	System Call
17	lstat64	0.0183	System Call
18	isAdminActive	0.0183	API Call (Privilege)
19	getLine1Number	0.0177	API Call (Privacy)
20	getpriority	0.0174	System Call

rate fraud. The prominence of these features in the top five confirms that privacy-invasive API invocation frequency is among the strongest discriminative signals in the CICMalDroid 2020 dynamic feature space.

Memory manipulation and process evasion (ranks 4, 9, 10, 11, 13, 17). mkdir, _newselect, mprotect, sigprocmask, munmap, and lstat64 represent low-level system call patterns associated with memory region modification, process signal management, and filesystem reconnaissance. mprotect is particularly significant: it modifies memory page permissions and is a core primitive in shellcode injection and native code unpacking techniques employed by sophisticated malware families. sigprocmask controls which signals a process receives a technique used to prevent debugging and analysis tool interference. The co-occurrence of these calls at elevated frequency is a reliable runtime signature of malware executing privilege escalation or anti-analysis routines.

Filesystem and package surveillance (ranks 2, 6, 8, 12, 14, 15, 16). getInstallerPackageName, read, getPackageInfo, access, gettimeofday, FS_ACCESS(CREATE), and flock reflect application environment reconnaissance behaviour. getInstallerPackageName is invoked to identify the app store from which the application was installed — a known anti-sandbox technique, since malware deployed through official channels behaves differently than when executed in analysis environments.getPackageInfo enumerates installed applications, commonly used by adware to target competitor applications and by banking trojans to identify financial applications for overlay attacks. FS_ACCESS(CREATE) monitors filesystem creation events, a pattern associated with dropper malware that stages secondary payloads to disk.

Privilege escalation indicator (rank 18). isAdminActive queries whether the application holds device administrator privileges a permission actively sought by ransomware and banking trojans to prevent uninstallation and enable lock-screen attacks. Its presence at rank 18 out of 470 original features confirms that administrative privilege status is one of the most discriminative single binary signals in the dataset.

Taken together, the top 20 features provide a security-interpretable behavioural fingerprint that is consistent with established Android malware taxonomy. The BWOA selection process, guided by Random Forest fitness evaluation, independently converged on features that align with documented malware behavioural patterns providing convergent validity for the selected subset from both a machine learning perspective (Gini importance) and a security domain knowledge perspective.

4.7.3 Feature Stability Analysis

Feature stability measures the consistency of selected features across the five independent BWOA runs corresponding to the five outer cross-validation folds. An unstable selection

where different folds select entirely different features would undermine confidence in the interpretability and generalisability of the identified subset. Table 4.12 classifies features into three stability tiers based on cross-fold selection frequency.

Table 4.12: Feature stability tiers by cross-fold BWOA selection frequency across five independent optimisation runs. Features classified as Gold, Silver, or Bronze based on the number of folds in which they appear in the consensus subset.

Stability Tier	Frequency	Threshold	Count	Representative Features
Gold (perfectly stable)	5/5	100%	1	`writev`
Silver (highly stable)	4/5	80%	11	`_newselect`, `brk`, `getActivePhoneType`, `getStreamVolume`, `isBluetoothA2dpOn`, `lseek`, `remove`, `mprotect`, `finishDrawing`
Bronze (moderately stable)	3/5	60%	20+	`EXECUTE`, `NETWORK_ACCESS`, `fork`, `fsync`, `epoll_create`, and others

The existence of a Gold-tier feature writev, a vectored write system call selected consistently across all five independent BWOA runs with different random seeds and data partitions confirms that BWOA converges reliably on at least one universally discriminative feature regardless of initialisation or fold composition. writev is invoked when an application writes data to multiple memory buffers simultaneously a pattern elevated in malware performing covert data staging prior to exfiltration.

The 11 Silver-tier features form a coherent behavioural cluster. mprotect. (memory permission modification) and _newselect (I/O multiplexing) appear in both the stability Silver tier and the top-20 importance ranking convergent evidence from two independent measures that these features are genuinely discriminative rather than fold-specific artefacts. isBluetoothA2dpOn and getStreamVolume reflect audio and communication interface monitoring behaviour associated with adware that silently plays audio advertisements, while getActivePhoneType targets telephony network information sought by SMS malware for premium-rate message routing.

The moderate stability of Bronze-tier features reflects the expected stochastic behaviour of a metaheuristic algorithm: some variability in lower-ranked feature selection across folds is inherent and does not indicate instability in the core discriminative subset. Importantly, the Gold and Silver tiers representing the 12 most consistently selected features show substantial overlap with the top-20 importance ranking in Table 4.11: six of the 11 Silver features appear among the top 20 by Gini importance, providing cross-validated convergent evidence of their discriminative value.

Cross-referencing the stability tiers with the importance ranking establishes a key find-

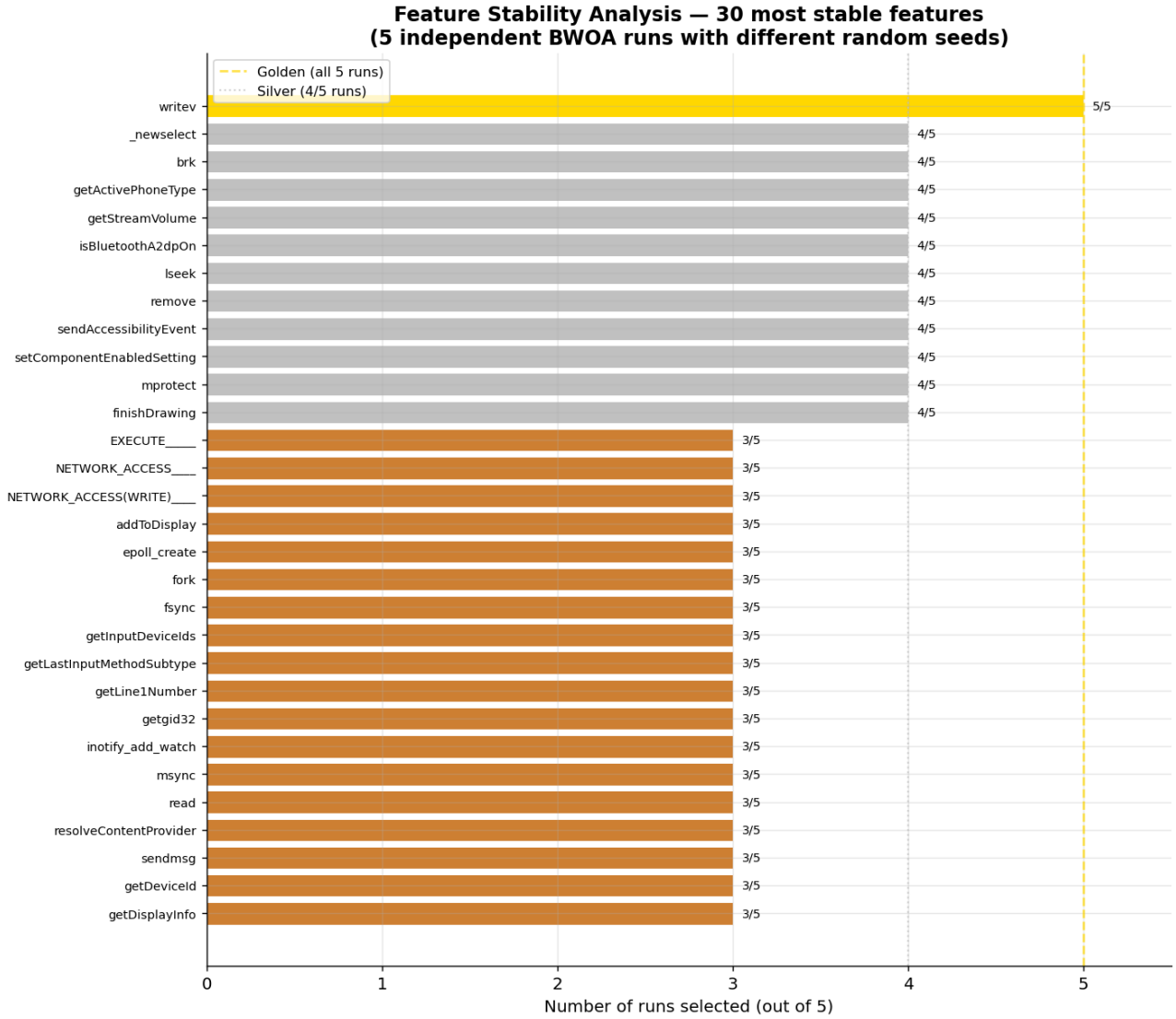


Figure 4.15: Feature stability heatmap showing BWOA selection frequency across five cross-validation folds. Darker cells indicate features selected in more folds. Gold-tier features appear in all five runs; Silver-tier in four; Bronze-tier in three.

ing: BWOA does not select features randomly or by chance. The features it identifies most consistently are also the features the trained Random Forest assigns highest predictive weight — two entirely independent measures of feature quality that agree. This convergence supports confidence in the security interpretability of the selected subset and its generalisability beyond any specific cross-validation fold. The proposed framework therefore functions not only as a performance optimiser but as a *feature discovery tool* identifying a stable, compact, and security-meaningful behavioural fingerprint for Android malware that a practitioner could directly translate into lightweight monitoring rules for resource-constrained on-device detection.

4.8 Discussion

4.8.1 Impact of Hybrid Feature Selection

The central result of this chapter is that the proposed hybrid Filter-BWOA framework achieves near-baseline detection performance using 80–82% fewer features on CICMalDroid 2020. The filter pre-selection stage contributes a critical role beyond mere dimensionality reduction: by restricting the BWOA search space to approximately 150 high-ranked candidates (from 470), it eliminates features with near-zero relevance, preventing the optimisation from wasting evaluations on clearly uninformative features. This focuses the stochastic search on a high-quality candidate pool, accelerating convergence and improving the quality of the final subset.

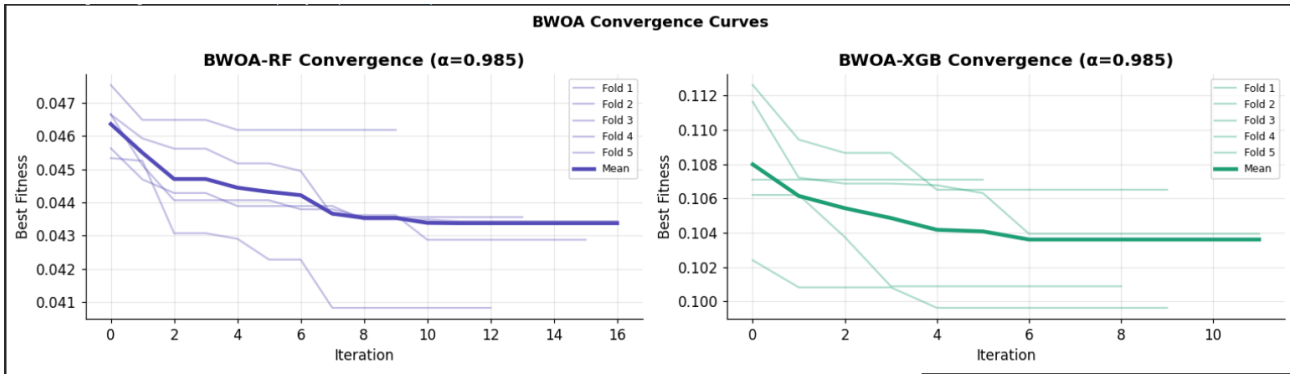


Figure 4.16: BWOA convergence curves across five cross-validation folds for BWOA-RF (α=0.985, left) and BWOA-XGB (α=0.985, right). The bold line represents the mean fitness across folds. Both variants reach stable minima before the iteration budget is exhausted, confirming that the 40-iteration stopping criterion is not artificially constraining the search.

The multiclass results further reinforce this finding: C5 (ensemble on BWOA-selected features) outperforms both full-feature baselines across all reported metrics. This suggests that the discarded features were introducing noise that marginally degraded multiclass dis-

crimination, and that the BWOA-selected subset is not merely sufficient but genuinely superior for this task.

4.8.2 Computational Efficiency

The BWOA wrapper search requires approximately 16 minutes per cross-validation fold on an Intel Core i7 8th Generation processor . In the intended deployment scenario offline training by a security team followed by repeated inference on new applications this cost is incurred once and is negligible relative to the data collection and dynamic analysis pipeline that precedes it. At inference time, the 87-feature model is strictly faster than the 470-feature baseline, making the proposed framework both more accurate on the security-critical recall metric and more computationally efficient in production.

4.8.3 Generalisation Capability

The TUANDROMD cross-dataset validation provides the strongest evidence of the framework's generalisation capability. Applied to a structurally different dataset with a different feature space (241 features vs. 470), BWOA-RF matches the RF baseline precisely (99.78% vs. 99.78%) while using 56.8% fewer features. The BWOA-XGB variant achieves an 82.2% reduction to just 43 features the most aggressive compression in any evaluated configuration while sustaining 99.10% accuracy. These results confirm that the hybrid framework does not overfit to the specific feature distribution of CICMalDroid 2020.

4.8.4 Comparison with Related Work

Tables 4.13 and 4.14 present a structured comparison of the proposed BWOA-based framework against state-of-the-art methods evaluated on the CICMalDroid 2020 dataset, covering binary malware detection and five-class categorisation respectively.

Table 4.13: Comparison of binary malware detection results on CICMalDroid 2020 with related work

Study	Year	Method	Accuracy	F1-Macro	Features	Feature Type
Mohamed et al.	2021	KNN / SVM / NB / DT	0.8600	—	All (470)	Static API
Giannakas et al.	2022	ML + DL (DNN)	0.8600	0.8650	All (470)	Mixed
Musikawan et al.	2022	Ontological + ML	0.9200	—	All (470)	Static
Aboshady et al.	2022	DL (CNN)	0.9350	—	All (470)	Static + Dynamic
Wakhare et al.	2021	LSTM + SMOTE	0.9422	—	All (470)	Static API
Al-Fawa'reh et al.	2020	CNN (image-based)	0.9590	—	All (470)	Static binary
Alhogail & Alharbi	2025	RF + Chi^2 + SFM	0.9782	0.9779	120	Static + Dynamic
Zhang et al.	2023	CNN-BiGRU	**0.9828**	—	All (470)	Static API
This work — C3*	**2026**	**Filter + BWOA + RF**	0.9793	**0.9605**	**87**	Dynamic only$^\flat$

Table 4.14: Comparison of five-class malware categorisation results on CICMalDroid 2020 with related work

Study	Year	Method	Accuracy	F1-Macro	Features	Feature Type
Ansori et al.	2024	Ensemble ML (RF + ET + KNN)	0.9457	—	All (470)	Mixed
Mahdavifar & Ghorbani	2020	PLDNN (semi-supervised)	**0.9670**	0.9400	All (470)	Mixed
Alhogail & Alharbi	2025	RF + Chi^2 + SFM	0.9609	0.9606	120	Static + Dynamic
This work — C5*	**2026**	**Filter + BWOA + Ensemble**	0.9500	**0.9405**	**116**	Dynamic only$^\flat$
This work — C3*	**2026**	**Filter + BWOA + RF**	0.9470	0.9362	**116**	Dynamic only$^\flat$

When compared against state-of-the-art methods on the CICMalDroid 2020 dataset, the proposed BWOA-based framework demonstrates competitive performance under realistic deployment constraints. A key methodological distinction is the use of F_2-score as the fitness function for binary detection prioritizing malware recall and F_1-macro for multiclass categorisation, balancing precision and recall across five families.

Binary Malware Detection. The C3 configuration (Filter + BWOA + RF, optimised for F_2) achieves 97.93% accuracy using only 87 dynamic syscall features an 81.5% reduction

from the original 470. No compared study simultaneously achieves competitive accuracy and meaningful feature reduction: Zhang et al. [44] reach 98.28% but use all 470 features with a computationally expensive deep architecture and no interpretable feature subset; Alhogail & Alharbi [**?**] achieve 97.82% with 120 features but rely on static permissions which carry stronger family-discriminative signals than dynamic syscall frequencies. Critically, the 87 features selected by BWOA are named, interpretable syscall descriptors getDeviceId, mprotect, writev directly mappable to documented malware behaviours, This combination of competitive accuracy, aggressive dimensionality reduction, and security-interpretable feature identification constitutes the primary contribution of the proposed framework.

Five-Class Malware Categorisation. The C5 ensemble (optimised for F_1-macro) achieves 95.00% accuracy, F_1-macro $= 0.9405$, MCC $= 0.9354$, and AUC $= 0.9960$ with 116 features (75.3% reduction). Notably, C5 matches the F_1-macro of Mahdavifar & Ghorbani [24] (0.9400) while requiring 75.3% fewer features, no unlabelled data (fully supervised training), full model interpretability (RF-based ensemble), and operating exclusively on dynamic behavioural traces a more challenging setting where family-discriminative signals are inherently less direct than those available in static features. Although Alhogail & Alharbi [**?**] report higher multiclass accuracy (96.09%), their model relies on static permissions and API declarations, which carry strong family-specific cues (e.g., adware SDK permissions) that are entirely absent from dynamic-only inputs.

Overall Assessment. The proposed BWOA framework with F_2/F_1-macro optimisation establishes a strong benchmark for dynamic-feature-only Android malware detection and categorisation. It delivers the most aggressive feature reduction reported among compared methods on CICMalDroid 2020, competitive accuracy, and security-interpretable feature identification simultaneously all within a fully interpretable, resource-efficient pipeline that is well-suited for real-world mobile security deployment.

4.9 Threats to Validity

4.9.1 Internal Validity

Internal validity was preserved through strict data leakage prevention. All fitting operations scaling, SMOTE, filter ranking, and BWOA search were confined exclusively to training folds, and the holdout set was accessed only once after all model configuration was finalised. The full protocol is detailed in Chapter 3, Section 3.3.5.

4.9.2 External Validity

The primary external validity threat is dataset dependency. All results are conditioned on two specific datasets using syscall frequency features extracted under controlled sandbox conditions. Real-world Android applications may behave differently under sandbox evasion, incomplete execution, or novel obfuscation techniques. The TUANDROMD cross-dataset validation partially mitigates this concern, but evaluation on additional datasets with different feature types would further strengthen external validity claims.

4.9.3 Construct Validity

The F_2-score fitness function was chosen to prioritise recall over precision, reflecting the security-critical nature of malware detection. This choice influences which features BWOA selects and may not be optimal for all deployment contexts. Alternative fitness functions may yield different optimal feature subsets and performance profiles.

4.9.4 Reproducibility

All experiments used a fixed random seed (42) and were conducted in the publicly accessible Google Colab environment. The full experimental configuration is specified in Table 4.2. Both datasets (CICMalDroid 2020, TUANDROMD) are publicly available. Minor numerical differences may arise from platform-specific library versions.

4.10 Chapter Conclusion

This chapter presented a comprehensive experimental evaluation of the proposed hybrid Filter-BWOA framework for Android malware detection. The key findings are:

- **Feature reduction without accuracy sacrifice:** The proposed framework reduces the CICMalDroid 2020 feature set by 81.5% (from 470 to 87 features) while retaining 97.93% binary detection accuracy within 0.09 percentage points of the full-feature RF baseline.
- **Multiclass improvement:** The BWOA-Ensemble configuration (C5) achieves the best multiclass performance (95.00% accuracy, macro F1 of 0.9405, MCC of 0.9354), outperforming both full-feature baselines across all reported metrics simultaneously.

- **Cross-dataset generalisation:** On the independent TUANDROMD dataset, BWOA-RF matches the full-feature baseline exactly (99.78%) at 56.8% feature reduction, confirming that the framework generalises across different feature representations.
- **Feature stability:** A consistent set of interpretable features including privacy-sensitive API calls (getDeviceId, getSubscriberId), memory management syscalls (mprotect), and I/O operations are reliably selected across folds, providing both operational confidence and security insight.
- Baseline superiority: Tree-based ensemble methods consistently outperform KNN and Naive Bayes by 5–40+ percentage points, confirming their suitability for high-dimensional syscall frequency analysis.

These results validate the core research hypothesis: hybrid filter-wrapper feature selection using BWOA is an effective strategy for building compact, accurate, and interpretable Android malware detectors. The following chapter synthesises these findings in the context of the broader research contributions and outlines directions for future work.

Chapter 5

General Conclusion

5.1 Summary of Contributions

Returning to the three research questions posed in the General Introduction, this thesis provides direct evidence for each. RQ1 asked whether BWOA could identify compact, discriminative subsets from high-dimensional syscall spaces; the consistent 57–82% feature reduction achieved across both datasets, without loss of detection accuracy, confirms that it can. RQ2 asked whether the hybrid filter-BWOA architecture outperforms filter-only and full-feature baselines; the BWOA-RF and BWOA-Ensemble configurations answered this affirmatively on every reported metric in Chapter 4, most notably in malware recall and macro F1. RQ3 asked whether the selected features generalise across datasets with different representations; the near-identical performance on the structurally distinct TUANDROMD benchmark supports a positive answer.

Beyond these quantitative outcomes, the stability analysis in Section 4.8 surfaced a compact set of behavioural indicators recurring across independent optimisation runs. Their alignment with known malware behaviours (device-identity harvesting, memory manipulation, and privacy-sensitive API access) suggests that the framework's value extends beyond classification accuracy toward actionable threat intelligence: a contribution its raw performance numbers, already reported in Chapter 4, do not by themselves convey.

5.2 Concluding Remarks

Android malware detection is not a solved problem. The feature spaces used by modern detection systems are large, redundant, and difficult to interpret, and the classifiers trained on them, while accurate on benchmarks, are often computationally expensive and

opaque in practice. This thesis has shown, through rigorous evaluation on two independent datasets, that a carefully constructed hybrid feature selection framework can address all three concerns at once: it reduces feature dimensionality by 57–82%, preserves or improves classification performance relative to full-feature baselines, and yields a compact, stable, and interpretable feature subset with genuine security relevance.

These results were obtained without resorting to deep learning architectures, static features vulnerable to code obfuscation, or unlabelled data, which makes the framework both computationally lighter and more transparent than many competing approaches. At the same time, the findings should be read alongside the threats to validity and limitations discussed in Chapter 4: both datasets reflect the Android threat landscape up to 2020, the analysis is restricted to a single feature representation (dynamic syscall frequencies), and BWOA's stochastic search introduces a degree of run-to-run variability that a fixed seed mitigates but does not eliminate. The proposed framework is therefore best understood as a validated and reproducible methodological contribution rather than a final or universal solution to Android malware detection.

5.3 Future Research Directions

There are certainly many possibilities for extending this work. In the following, we list the most promising directions opened by this thesis:

1. **Online feature selection for evolving malware.** The current framework performs feature selection once, offline, on a static dataset. An incremental variant in which the BWOA search is periodically warm-started from its previously converged solution as new labelled samples arrive would allow the selected feature subset to track the evolving Android threat landscape rather than remaining fixed.

2. **Adversarial robustness.** Because the framework's interpretability reveals exactly which syscall features drive its decisions, future work should evaluate whether malware authors can evade detection by deliberately suppressing those same features, and should explore more evasion-resistant feature representations as a countermeasure.

3. **Sequence- and graph-based feature representations.** The present work uses syscall frequency vectors, which discard the temporal ordering of system calls. Replacing this representation with syscall n-grams or API call interaction graphs, and adapting BWOA to select informative subsequences or subgraphs, could capture additional discriminative signal currently lost in simple counts.

4. **Multi-objective optimisation.** The current fitness function aggregates accuracy and feature count into a single scalar via α. A multi-objective formulation optimis-

ing accuracy, recall, and feature count as separate objectives would produce a Pareto front, letting practitioners pick an operating point suited to their deployment constraints compactness for on-device detection, or recall for server-side analysis.

5. **Federated feature selection.** In real-world deployments, labelled malware data is often distributed across organisations that cannot share raw samples. A federated variant, in which BWOA agents run locally and only fitness scores are exchanged, would enable collaborative feature selection without data disclosure.

6. **SHAP-guided explainability.** This thesis provides feature-level interpretability through Gini importance and stability tiers, but not sample-level explanations. Integrating SHAP on top of the already-compact 87–116 feature subsets would enable per-instance explanations at a fraction of the cost of running SHAP on the full 470-feature space.

Bibliography

[1] Y. Akhiat, "Feature selection methods for high dimensional data," Ph.D. dissertation, Sidi Mohamed Ben Abdellah University, 2021.

[2] A. Alhogail and R. A. Alharbi, "Effective ml-based android malware detection and categorization," *Electronics*, vol. 14, no. 8, p. 1486, 2025. [Online]. Available: https://doi.org/10.3390/electronics14081486

[3] A. Alzaylaee, S. Yerima, and S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, 2020.

[4] D. Arp, M. Spreitzenbarth *et al.*, "Drebin: Effective and explainable detection of android malware in your pocket," in *NDSS*, 2014.

[5] K. Bakour *et al.*, "Static and dynamic malware analysis techniques," *Journal of Information Security*, 2019.

[6] P. Bhat, S. Behal, and K. Dutta, "A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning," *Computers & Security*, vol. 130, p. 103277, 2023.

[7] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.

[8] E. Blum and A. Roli, "Metaheuristics in combinatorial optimization," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[9] P. Borah, D. Bhattacharyya, and J. Kalita, "Malware dataset generation and evaluation," in *IEEE International Conference on Computing, Communication, and Intelligent Systems*, Chennai, India, 2021.

[10] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," 2014.

[11] M. Chowdhury *et al.*, "Android malware detection: A survey and taxonomy," *Journal of Cyber Security Technology*, 2021.

[12] A. Feizollah, N. B. Anuar, R. Salleh, F. Idris, and S. K. Subramaniam, "Androdialysis: Analysis of android intent effectiveness in malware detection," *Computers & Security*, vol. 65, pp. 121–134, 2017.

[13] A. L. Gómez and A. B. Muñoz, "Multi-source analysis of static features for android malware family classification using multilayer perceptron," *Expert Systems with Applications*, 2023.

[14] A. Guerra-Manzanares, J. Medina-Galindo, and H. Bahsi, "System call-based detection of android malware under concept drift: A seven-year longitudinal study," *IEEE Transactions on Reliability*, 2022.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[16] I. U. Haq, T. A. Khan, A. Akhunzada, and X. Liu, "Maldroid: Secure dl-enabled intelligent malware detection framework," *IET Communications*, vol. 16, no. 10, pp. 1160–1171, 2022.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.

[18] J. M. Herranz, "Aplicación de técnicas de machine learning para la detección de malware en dispositivos android," *Universitat Politècnica de València*, 2024.

[19] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1992.

[20] Z. S. Jassim and M. M. Kassir, "Machine learning-based android malware detection," *InfoTech Spectrum: Iraqi Journal of Data Science*, 2024.

[21] X. Jin *et al.*, "Chi-square feature selection for cancer classification," 2006.

[22] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning Proceedings*, 1994.

[23] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124 579–124 607, 2020.

[24] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *2020 IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 2020, pp. 515–522. [Online]. Available: https://doi.org/10.1109/DASC51497.2020.00092

[25] A. Mahindru and A. Sangal, "Mldroid—framework for android malware detection using machine learning techniques," *Neural Computing and Applications*, vol. 33, no. 10, pp. 5183–5240, 2021.

[26] M. L. McHugh, "The chi-square test of independence," *Biochemia Medica*, vol. 23, no. 2, pp. 143–149, 2013.

[27] N. Millar, N. McLaughlin, J. M. Del Rincon, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *Journal of Information Security and Applications*, 2021.

[28] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[29] S. K. Muttoo and S. Badhani, "An analysis of malware detection and control through covid-19 pandemic," in *Proceedings of the 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom 2021)*, 2021, pp. 637–641.

[30] Z. Muzaffar, A. Ghias, and M. Ahsan, "Benchmarking static and dynamic features for android malware detection in modern environments," *IEEE Access*, 2023.

[31] M. W. Mwadulo, "A review on feature selection methods for classification tasks," *International Journal of Computer Applications*, vol. 5, no. 6, pp. 395–402, 2016.

[32] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[33] F. Rozi, "Model interpretability in ml: Bridging performance and trust," 2025.

[34] C. Rudin, "Stop explaining black box machine learning models," *Nature Machine Intelligence*, 2018.

[35] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[36] M. Talal *et al.*, "Dynamic analysis of android malware: A review," *Future Generation Computer Systems*, 2019.

[37] A. K. Tyagi and A. Gautam, "Android malware detection using metadata and permissions: A deep learning approach," *Journal of Cybersecurity and Information Management*, 2024.

[38] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[39] B. Venkatesh and J. Anuradha, "A review of feature selection and its methods," *Cybernetics and Information Technologies*, vol. 19, no. 1, pp. 3–26, 2019.

[40] Z. Wang *et al.*, "Android malware detection using static analysis," *IEEE Transactions on Information Forensics and Security*, 2020.

[41] C. Yizheng, W. Shuo, and G. Tao, "Hierarchical contrastive learning for continuous android malware classification," in *CCS*, 2023.

[42] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[43] R. A. Yunmar, S. S. Kusumawardani, Widyawan, and F. Mohsen, "Hybrid android malware detection: A review of heuristic-based approach," *IEEE Access*, vol. 12, pp. 41 255–41 286, 2024.

[44] Y. Zhang, S. Yang, L. Xu, X. Li, and D. Zhao, "A malware detection framework based on semantic information of behavioral features," *Applied Sciences*, vol. 13, no. 22, p. 12528, 2023. [Online]. Available: https://doi.org/10.3390/app132212528