Journal of the Brazilian Computer Society, 202X, XX:1, doi: 10.5753/jbcs.202X.XXXXXX

Towards a Lightweight Multi-View Android Malware Detection Model with Multi-Objective Feature Selection

Philipe Fransozi [**Pontifical Catholic University of Parana (PUCPR)** | ***philipe.hfransozi@ppgia.pucpr.br***]
Jhonatan Geremias [**Pontifical Catholic University of Parana (PUCPR)** | ***jgeremias@ppgia.pucpr.br***]
Eduardo K. Viegas [**Pontifical Catholic University of Parana (PUCPR)** | ***eduardo.viegas@ppgia.pucpr.br***]
Altair O. Santin [**Pontifical Catholic University of Parana (PUCPR)** | ***santin@ppgia.pucpr.br***]

Graduate Program in Informatics (PPGIa), Block 8 -Technological Park - 2nd floor, Imaculada Conceição Street, 1155 - Prado Velho, ZIP Code 80215-901 - Curitiba - PR

Received: DD Month YYYY • **Accepted:** DD Month YYYY • **Published:** DD Month YYYY

Abstract. In recent years, a wide range of new Machine Learning (ML) techniques with high accuracy have been developed for Android malware detection. Despite their high accuracy, these techniques are seldom implemented in production environments due to their limited generalization capabilities, leading to reduced performance when applied to real-world scenarios. In light of this, this paper introduces a novel multi-view Android malware detection model implemented in two stages. The first stage involves extracting multiple feature sets from the analyzed Android application package, offering complementary behavioral representations that improve the system's generalization in the classification process. In the second stage, a multi-objective optimization is conducted to identify the optimal feature subset from each view and fine-tune the hyperparameters of individual classifiers, enabling an ensemble-based classification approach. The core innovation of our approach lies in the proactive selection of feature subsets and the optimization of hyperparameters that together enhance classification accuracy while minimizing processing overhead within a multi-view framework. Experiments conducted on a newly developed dataset, consisting of over 40 thousand Android application samples, validate the effectiveness of our proposal. The results indicate that our model can increase true-positive rates by up to 18% while reducing inference processing costs by as much as 72%.

Keywords: Android Malware, Machine Learning, Multi-view, Multi-objective

1 Introduction

Android is the most used mobile operating system globally, reaching over 3 billion active users and capturing nearly three-quarters of the current market share [Curry, 2024]. However, this popularity has also made Android a prime target for cyber threats, particularly in the form of malicious applications, namely *malware*, designed to exploit vulnerabilities in the system. As an example, according to a recent cybersecurity report [Kaspersky, 2024], the number of newly identified Android malware samples surged by 52% in 2023 alone, with several of these malicious apps even infiltrating official app stores, exposing users to security risks. Therefore, despite the various malware detection strategies currently in place, these approaches have proven inadequate in reaching protection, leaving many users vulnerable to security breaches.

Android malware detection typically relies on either dynamic or static analysis methods [Qiu *et al*., 2020]. On the one hand, dynamic-based techniques involve executing and monitoring the target application in a controlled environment to detect malicious footprints [Li *et al*., 2022]. These techniques often capture runtime indicators, such as network connections, file system changes, or unauthorized access to system resources to conduct detection. Therefore, they often demand substantial computational resources, time, and sophisticated monitoring setups implemented utilizing sandbox environments. Unfortunately, some malware types can conceal their behavior or activate malicious functions only under specific conditions, further complicating detection and reducing the effectiveness of dynamic analysis [Cui *et al*., 2023]. On the other hand, static-based approaches analyze an application's files, including its code, permissions, and configuration files, without requiring the analyzed app execution. This approach is typically more efficient and scalable, allowing for analyzing large volumes of applications with fewer resources. Due to these advantages, static methods are often preferred in research and industry, where they have demonstrated promising detection rates with reduced operational complexity trade-offs [Bhat *et al*., 2023].

Static-based Android malware detection primarily involves analyzing the Android Application Package (APK) files to identify potential threats. This analysis typically encompasses several key components, including examining the app's requested permissions, detailed in the *manifest.xml* file, and the analysis of the binary source code contained in the *.dex* files [Pan *et al*., 2020a]. By inspecting these elements, researchers can extract relevant features to understand the app's behavior and functionality [Darwaish and Nait-Abdesselam, 2020]. In recent years, the literature has seen a surge in various approaches aimed at improving the classification of Android applications. Many of these approaches leverage Machine Learning (ML) techniques, which have become increasingly popular due to their ability to handle large datasets and discover complex patterns within the data [Mahindru and Sangal, 2020]. These approaches use a feature vector generated from the extracted behavioral characteristics of the

application. This feature vector serves as input to an ML model that processes the data to classify the app as either *goodware* or *malware*.

The reliance on ML techniques has significantly improved the accuracy of current malware detection strategies. By training models on diverse datasets that include both *goodware* and *malware* applications, researchers can enhance the models' ability to generalize and identify new malware variants effectively. Various algorithms, such as decision trees, support vector machines, and deep learning models, have been applied in this context, each with strengths and weaknesses [Odat and Yaseen, 2023]. Overall, static-based detection methods represent a critical component of Android security frameworks, as they offer a proactive approach to identifying potential threats before they can execute and harm users.

Surprisingly, despite the promising results reported in the literature, including high classification accuracies, current ML-based techniques are seldom deployed in production environments [Smith *et al*., 2020]. Android malware behavior's inherent complexity is a major challenge, typically marked by various malicious indicators. This complexity often necessitates the examination of multiple files and attributes to ensure the accurate classification of an application. For instance, an analyzed APK might request several highly sensitive permissions; however, a detailed analysis of its source code could reveal that these permissions are used appropriately and for legitimate purposes [Molina-Coronado *et al*., 2023]. As a result, if classification decisions are based solely on requested permissions, there is a risk of false positives, where legitimate applications are misclassified as *malware*, potentially undermining user trust.

Many current approaches in the literature rely heavily on analyzing a single APK file for the classification task, often overlooking the multi-view nature of malware behavior [Pan *et al*., 2020b]. This single-file focus can limit detection accuracy, as it fails to account for the broader context of an application's behavior and interactions. Consequently, methodologies that integrate multiple behavioral attributes and operational contexts are needed to improve the robustness and reliability of Android malware detection systems [Geremias *et al*., 2023]. In addition, analyzing multiple files also introduces considerable computational costs, making such techniques challenging to implement on resource-constrained devices like smartphones [Ma *et al*., 2024]. These devices typically lack the processing power and memory needed to support extensive, multi-file analysis, which further hinders the deployment of advanced malware detection techniques in mobile environments.

Combining multiple feature sets for Android malware classification has been explored in several studies [Darwaish and Nait-Abdesselam, 2020]. Typically, these schemes employ Deep Neural Network (DNN)-based techniques, which enhance detection accuracy by capturing complex patterns across diverse feature spaces. Yet, these methods often entail significant memory and processing demands, posing challenges for deployment on mobile and other resource-limited devices [Ravi *et al*., 2022]. Moreover, finding the optimal feature sets in a multi-view setting remains complex. Each classifier involved must evaluate its performance within the broader pool of selected features, ensuring that the final configuration maximizes classification efficacy without imposing excessive computational overhead on devices that may not be able to sustain such intensive analysis.

Contribution. In light of this, this paper introduces a novel multi-objective optimization model for multi-view classification of Android malware, implemented in two stages. In the first stage, multiple feature sets are extracted from an analyzed Android APK, with each feature set derived from a distinct APK file component, providing a complementary behavioral vector for comprehensive classification. In the second stage, a multi-objective optimization is conducted to identify an optimal subset of features from each feature set and to tune the associated hyperparameters of individual classifiers, enabling an ensemble-based classification approach. The core insight of our proposal is to identify and select features that not only enhance classification accuracy but also reduce inference processing requirements, making it suitable for deployment on resource-constrained devices. As a result, our scheme leverages complementary feature sets for Android malware detection, achieving reliable classification performance while balancing processing demands for a more efficient multi-view detection approach.

In summary, the main contributions of our work are:

- A new publicly available multi-view Android malware dataset with complementary feature sets extracted from the APK API Calls, OPCodes, and Permissions related files. The dataset was built through the analysis of ≈ 40 thousand Android samples collected from a two-year interval;
- A new multi-view Android malware detection model implemented through a multi-objective optimization strategy. Our proposed scheme can improve true-positive accuracy by up to 18% while reducing inference processing costs by as much as 72%.
- An explainable machine learning analysis that demonstrates the significance of each feature set in our multi-view approach, providing insights into how multi-view integration enhances classification accuracy and efficiency in our proposed model;
- A lightweight prototype implementation of our multi-view classification model on an Android device, demonstrating its feasibility for real-time malware detection with minimal resource consumption, suitable for mobile environments;

Roadmap. The remainder of this paper is organized as follows. Section 2 further describes ML-based Android malware detection. Section 3 describes the related works. Section 4 presents our proposal, while Section 5 describes our prototype implementation. Section 6 evaluates the performance of our model, and Section 7 concludes our work.

2 Background

This section provides an in-depth discussion on detecting Android *malwares* using ML-based techniques. First, we outline the application of ML in identifying Android malware, detailing the general process and techniques used. Following this, we examine the challenges inherent in applying these

methods, addressing the limitations and obstacles that affect their effectiveness in practical scenarios.

2.1 Static-based Android Malware Detection

Android malware detection using ML-based techniques generally follows a structured four-phase process [Qiu *et al*., 2020]. In the initial phase, the *Data Extraction* module gathers relevant APK files, such as through retrieving requested permissions from the app's *manifest.xml* file. Next, a *Feature Extraction* module analyzes the behavioral attributes of the extracted file to generate a feature vector tailored to the file type being evaluated. For instance, a permissions-based feature vector would comprise a binary list indicating which permissions are requested by the app. Similarly, API calls the app makes can also be analyzed, producing a feature vector based on the frequency or type of API calls, which may signal potentially suspicious behavior. The third phase involves a *Classification* module, which applies a trained ML model to categorize the feature vector as either *goodware* or *malware*. Finally, samples classified as *malware* trigger the *Alert* module, which signals a response to mitigate potential threats.

The successful application of ML-based detection of Android malware depends on using a representative training dataset [Smith *et al*., 2020]. This is crucial because the ML model is trained by analyzing the behavior patterns present in the dataset. Therefore, it must reflect various possible behaviors across different samples. A diverse and comprehensive dataset ensures that the model can generalize well and detect various malware types effectively. However, building such a representative training dataset presents a significant challenge. It requires collecting a large and varied number of malware samples that account for different malware behaviors, variants, and evasive techniques. Additionally, the dataset must also include a wide range of benign applications to avoid class imbalance and ensure the model can distinguish between *goodware* and *malware* accurately [Guerra-Manzanares *et al*., 2021]. Surprisingly, much of the existing literature relies on outdated training datasets, often containing only hundreds or a few thousand malware samples. This situation limits the model's ability to learn from a broader range of malicious behaviors.

Another challenge lies in analyzing multiple feature sets, which is increasingly necessary for more accurate Android malware detection. Multiple behavioral feature sets, such as app permissions, API calls, and system interactions, are often combined to improve detection [Millar *et al*., 2021]. Here, selecting and analyzing the most relevant features from each set requires careful consideration of their relationships and potential redundancies. Moreover, this multi-feature analysis significantly increases the complexity of the detection task, as it demands more computational resources, making their implementation a challenge on resource-constrained devices like smartphones.

2.2 The Challenges of ML-based Android Malware Detection

Over the past few years, several works have proposed highly accurate ML-based techniques for Android malware detection [Molina-Coronado *et al*., 2023]. However, despite the promising results, these proposed schemes are rarely adopted in production environments. A key challenge lies in the need for a large number of training samples to design a reliable ML-based scheme. This is because the classifier must be able to generalize the behaviors observed in the training data to those encountered during production deployment, ensuring that it can handle diverse, real-world scenarios.

Providing a well-generalized ML model is particularly challenging when relying on a multi-view strategy. Android malware often requires evaluating multiple complementary behaviors to be accurately identified [Darwaish and Nait-Abdesselam, 2020]. For instance, sensitive permission might only be indicative of malicious activity if a source code with malicious intent is also identified. In such cases, the ML-based technique must be capable of incorporating multiple views during the classification task. However, this multi-view approach often comes at the cost of higher computational requirements, making it more difficult to deploy on resource-constrained devices such as smartphones [Millar *et al*., 2020]. The challenge of smartphone implementation lies in optimizing these techniques for mobile devices' limited processing capabilities, memory constraints, and battery life, which makes it difficult to perform complex malware analysis efficiently.

Notwithstanding, analyzing multiple feature sets and combining them adequately presents another significant challenge [Alani and Awad, 2022]. Different features, such as app permissions, API calls, and system behaviors, must be carefully selected and integrated to create a comprehensive feature vector. This process requires handling potential feature redundancies, ensuring that all relevant information is captured without introducing the model with unnecessary complexity. Balancing the richness of the feature sets with the need for computational efficiency is a delicate task, as the complexity of analyzing and combining multiple features can increase the computational cost, further hindering the deployment of these techniques on smartphones [Geremias *et al*., 2022].

3 Related Work

Android malware detection through ML-based techniques has been widely explored in the literature in recent years [Qiu *et al*., 2020]. The primary objective of these approaches is to achieve high accuracy on a given test dataset, often focusing on refining classification techniques to maximize detection performance. For example, D. O. Sahin *et al.* [Şahin *et al*., 2021] employs a feature selection technique on a permission-based ML model to detect malware. Their approach improves classification accuracy but relies on an outdated dataset with a limited number of samples, which restricts its ability to generalize to new malware samples. Similarly, S. Seraj *et al.*[Seraj *et al*., 2022] propose an updated dataset that includes a more

Table 1. A summary of related work and the characteristics of their Android malware detection strategy.

Work	Device Implementation	Multi-view Features	Representative Dataset	Feature Selection	Feature Analysis
D. O. Sahin *et al.*	×	×	×	✓	✓
S. Seraj *et al.*	×	×	✓	×	×
A. Pektas *et al.*	×	×	✓	×	×
A. Darwaish *et al.*	×	✓	✓	×	×
S. Millar *et al.*	×	✓	✓	×	×
V. Ravi *et al.*	×	✓	✓	×	×
A. Kyadige *et al.*	×	✓	×	×	✓
J. Geremias *et al.*	×	✓	✓	×	×
Y. Wu *et al.*	×	✓	✓	✓	✓
M. Azad *et al.*	×	×	✓	✓	✓
H. Hawks *et al.*	×	×	×	✓	×
L. da Costa *et al.*	✓	✓	×	×	×
C. Gao *et al.*	✓	✓	✓	×	×
M. M. Alani *et al.*	✓	×	✓	×	×
Ours	✓	✓	✓	✓	✓

significant number of malware samples. They use a DNN technique to improve accuracy compared to previous works. Unfortunately, their approach does not consider how incorporating complementary views—such as additional behavioral features or analysis of multiple components—could enhance the model's generalization, making it more robust in detecting previously unseen malware. A. Pektas *et al.*[Pektaş and Acarman, 2020] focus on using opcode sequences for malware detection, improving accuracy when combined with feature selection. However, like the previous approaches, they do not explore the potential of multi-view strategies to improve system reliability and robustness. Another approach, proposed by A. Darwaish *et al.* [Darwaish and Nait-Abdesselam, 2020], translates source code binaries into images for the classification task. While this proposal boosts accuracy, it also overlooks the advantages of integrating multiple feature sets, which could further refine malware detection by leveraging diverse perspectives of app behavior.

Combining multiple feature sets for Android malware detection has been the subject of several studies over the past years. S. Millar *et al.*[Millar *et al*., 2021] propose utilizing OPcodes, permissions, and API packages for the classification task. While their proposal improves classification accuracy, it fails to address how to optimize the combination of these views to enhance performance further. V. Ravi *et al.*[Ravi *et al*., 2022] take a step further by combining multiple views using a DNN-based approach. Their scheme boosts classification accuracy, but it comes with a significant tradeoff in processing costs, making it less suitable for resource-constrained devices. A similar approach is presented by A. Kyadige *et al.* [Kyadige *et al*., 2020], who also combine multiple views through a DNN-based scheme. While their method improves classification accuracy, it does not explore how these views can be combined optimally to achieve better results while minimizing computational overhead. J. Geremias *et al.* [Geremias *et al*., 2022] proposes a multi-view implementation making use of DNN-based classification strategy. Their approach combines multiple features to perform image-based classification with high accuracies. Similarly, the computational costs are overlooked.

Feature selection has been proposed as a means to improve classification accuracy in a multi-view setting. Y. Wu *et al.*[Wu *et al*., 2023] utilize reinforcement learning to perform the feature selection task, improving accuracy. However, their approach does not consider the application of multiple views, which could enhance the system's generalization and robustness. M. Azad *et al.*[Azad *et al*., 2022] apply particle swarm optimization for feature selection on a DNN-based classifier. While their approach improves accuracy, it overlooks the potential of integrating multiple views to enhance detection capabilities further. Similarly, H. Hawks *et al.* [Hijazi *et al*., 2023] propose a feature selection method combined with an ensemble of ML classifiers for the detection task. This method can increase classification accuracy but neglects to explore multi-view approaches, which could optimize feature selection and improve the overall detection process. Consequently, most current works fail to explore how the multi-view approach can be leveraged to achieve more reliable Android malware detection.

The implementation of Android malware detection strategies on resource-constrained devices, such as smartphones, is rarely addressed in the literature. L. da Costa *et al.*[Costa and Moia, 2023] propose a multi-detector strategy that identifies malware samples based on rule-based and triggering mechanisms. Their approach reduces processing costs by analyzing the malware only when certain preconditions are met. However, this activation strategy can be easily evaded by more sophisticated malware, potentially compromising the system's reliability. C. Gao *et al.*[Gao *et al*., 2024] develop a lightweight DNN implementation for image-based Android malware classification. Their method improves classification accuracy while minimizing the impact on processing costs, but it does not address the multi-view approach, which could further optimize the detection process. M. M. Alani *et al.* [Alani *et al*., 2023] propose a lightweight ML classifier that conducts Android malware detection using memory dump features and a tree-based classifier for efficient model implementation. However, their approach does not assess the scheme's performance on resource-constrained devices, making it unclear how it would perform in real-world smartphone environments. Therefore, while some approaches aim to improve efficiency, the challenge of implementing effective malware detection on smartphones remains largely unexplored in the literature.

3.1 Discussion

Table 1 overview the related works and the characteristics of their Android malware detection strategy. A critical gap in the literature on Android malware detection is the lack of practical implementation on resource-constrained devices, such as smartphones. Many works focus on improving detection accuracy but overlook the real-world applicability of

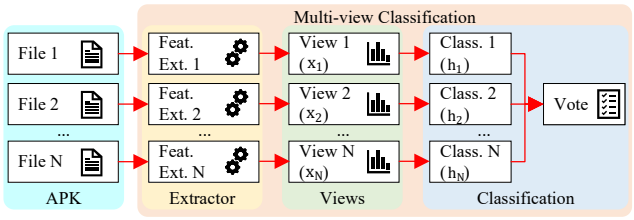


Figure 1. Overview of our proposed multi-view Android malware classification pipeline.

their methods. While several studies propose multi-view approaches, they rarely assess how such methods would perform on actual devices, leaving the effectiveness of these models in production environments unclear. Additionally, there is a consistent trend of not utilizing representative datasets that accurately capture the full spectrum of Android malware behaviors. This results in models that may not generalize well to new, unseen threats. Furthermore, the literature frequently neglects the importance of feature selection, a key step in reducing computational costs and improving model efficiency. While some studies do attempt feature selection, these efforts are often limited or not integrated with multi-view frameworks, limiting the potential for optimizing both accuracy and processing efficiency. Thus, while there has been significant advancement in Android malware detection, these works often fail to address the critical challenges of device implementation, dataset representativeness, and effective feature selection in a multi-view context.

4 A Lightweight Multi-view Android Malware Detection Model

To address the aforementioned challenge of lightweight Android malware detection in a multi-view setting, our proposed scheme introduces a multi-objective optimization approach that aims to balance accuracy with computational efficiency. By leveraging multiple feature sets, our model captures a more comprehensive view of the application behavior, enhancing its ability to detect diverse malware patterns while minimizing processing demands. Figure 1 illustrates the detailed operation of our proposed model, showcasing the sequence of data extraction, feature selection, and optimized classification processes that enable malware detection in a resource-constrained setting.

The proposed system implements a multi-view ML-based classification pipeline that operates using a static analysis of Android apps. In this approach, the Android APK file is analyzed through a multi-view process, where multiple files—such as the requested permissions from the *manifest.xml* and Opcodes from the *dex* file—are used to inform the classification task (Fig. 1, from *Files 1* to *N*). Our primary hypothesis is that incorporating multiple views can enhance the generalization and reliability of Android malware detection. Each file undergoes analysis through a dedicated feature extraction module (Fig.1, from *Feat. Ext. 1* to *N*), generating feature vectors that represent different behavioral aspects of the app. These feature vectors are subsequently processed by associated classifiers, producing individual classification outcomes for each view (Fig.1, from *Classifier 1* to *N*). To synthesize these outcomes, a combination module aggregates the classifications using a majority voting strategy, resulting in a final classification decision. As a result, this approach leverages complementary views, improving detection accuracy and adapting to diverse malware patterns.

To tackle the challenge of integrating multiple classifications from complementary views, we apply a multi-objective optimization approach. This approach simultaneously optimizes the selection of features within each view and the classifier's hyperparameters, balancing classification accuracy with processing costs. We measure the effectiveness of this optimization by evaluating both the accuracy and processing costs of the resulting classifier pool, ensuring a well-rounded performance. Our primary insight is that by jointly refining feature subsets and classifier configurations, we can achieve high malware detection accuracy while maintaining a lightweight, multi-view classification process, making it suitable for deployment on resource-constrained devices.

The following subsections further describe our proposed model, including its implementation components and the multi-objective optimization process.

4.1 Multi-view Classification

Most existing Android malware detection methods in the literature depend on a single view for classification. This reliance often limits their generalization capabilities, making them unsuitable for robust production use. To overcome this limitation, our proposed model adopts a multi-view classification process, which leverages an ensemble of classifiers. By incorporating multiple perspectives on the current analyzed Android APK behavior, our approach aims to enhance detection accuracy and reliability, providing a more comprehensive solution for Android malware detection.

Our scheme's operation is illustrated in Figure 1. It begins by analyzing the behavior of the Android APK to be classified, where the relevant files for feature extraction are obtained. Specifically, our scheme incorporates three views: *Permission* (manifest.xml), *Opcode* (dex), and *API Calls* (dex), each of which is further detailed in Section 5. The behavior of each file is then processed by an associated feature extractor to create a feature vector. Each feature vector is subsequently classified by its corresponding classifier (Fig. 1, *Classifier 1* to *N*). Finally, our scheme integrates these classifications using a *Combination* module, which applies a majority voting mechanism to determine the final decision.

Consequently, our proposed model leverages a multi-view approach to enhance system generalization and reliability. However, combining multiple views is not easily achievable due to the inherent complexities of integrating diverse feature sets and classifiers. By using a straightforward majority voting procedure, our scheme maintains minimal inference processing costs while ensuring robust decision-making. Below, we describe in detail how we optimize the combination of these views, addressing the challenges involved in this process.

Algorithm 1 Feature selection fitness computation for each individual

Require: Ensemble pool hyperparameters α, classifiers pool h, classifier views x
1: **Preprocess** training ($\mathcal{D}_{train}$) and testing ($\mathcal{D}_{test}$) datasets to filter selected views in x
2: **for** each classifier $h_i \in h$ **do**
3: **Fit** classifier h_i using hyperparameters α_i on training dataset $\mathcal{D}_{train}$ using view x_i
4: **end for**
5: **Measure** the pool accuracy using majority voting on the test data (Eq. 2)
6: **Measure** computational costs of the pool (Eq. 1)
7: **Return** pool accuracy and computational cost

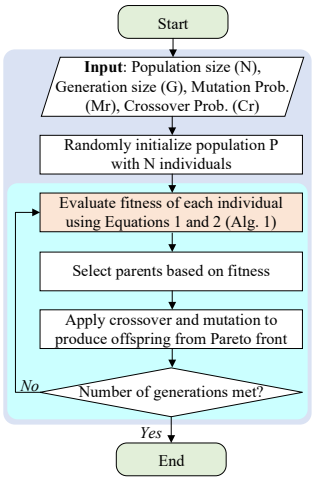


Figure 2. Flowchart of our proposed multi-objective optimization model building for multi-view Android malware classification.

4.2 Multi-objective Optimization

To combine multiple views for the Android malware detection task, we employ a multi-objective optimization approach. Specifically, we aim to simultaneously optimize inference processing time and the resulting ensemble accuracy. To achieve this, we implement the multi-objective optimization through a wrapper-based feature selection and classifier hyperparameter search method, utilizing a multi-objective genetic search algorithm. This approach enables the efficient exploration of feature subsets and the associated classifier hyperparameters while balancing both classification accuracy and processing costs. Figure 2 overviews the implementation of our multi-objective model building procedure for multi-view Android malware classification.

We consider a multi-view Android malware detection scheme implemented through an ensemble of classifiers (see Fig. 1). Each i^{th} classifier h_i conducts the classification task using its own view x_i. The multi-objective optimization goal is to find for each individual classifier i the best subset of features from view x_i and the associated classifier hyperparameters α_i that simultaneously improve the resulting accuracy and processing costs of the ensemble. To achieve such a goal, the multi-objective optimization task starts with a given population size N, a generation size G, a mutation rate M_r, and a crossover rate C_r. It then proceeds to initialize the population P with N individuals randomly. Here, each individual is represented by a set of views used by the classification system. Each view within an individual is characterized by a specific subset of features selected for classification, along with its corresponding classifier hyperparameter settings. This approach allows us to identify feature subsets and classifier hyperparameters that jointly enhance both classification accuracy and processing efficiency when integrated into the ensemble pool.

Algorithm 1 overviews the implementation of our model-building procedure for each individual. It receives as input a set of pool hyperparameters α, the classifiers pool h, and their associated views x. It then preprocesses the training and testing datasets to filter the selected views based on x. The preprocessed dataset fits each classifier h_i using their associated hyperparameter α_i and view x_i. Finally, using the built classifier pool, we measure the resulting accuracy and processing costs.

In practice, our multi-objective optimization task aims at finding a feature subset space x_i, and classifier hyperparameters α_i for each view such that it minimizes processing time and error rate. We measure processing time through the following equation:

$$obj_{proc}(h, \mathcal{D}_{test}) = \frac{\sum_{(x_i,y_i)}^{\mathcal{D}_{test}} time(h(x_i))}{N} \tag{1}$$

where $\mathcal{D}_{test}$ denotes the testing dataset, h the ensemble pool, $time$ a function that measures the ensemble inference time on a given test dataset sample x_i, and N the testing dataset size. Therefore, we measure the processing time as the average inference time of the ensemble pool on a given testing dataset. Similarly, we measure the ensemble pool error rate through the following equation:

$$obj_{error}(h, \mathcal{D}_{test}) = \frac{\sum_{(x_i,y_i)}^{\mathcal{D}_{test}} h(x_i) \neq y_i}{N} \tag{2}$$

where $\mathcal{D}_{test}$ denotes the testing dataset, $h(x_i)$ a function that outputs the pool-assigned label for the event, y_i the ground-truth label, and N the testing dataset size. Hence, we measure the error rate as the percentage of events incorrectly classified by the pool.

Given that iterating over all possible feature subspaces x_i and classifiers hyperparameters α_i is not feasible, we make use of the multi-objective optimization approach (Fig. 2). As a result, we aim to solve the following equation:

$$\underset{\{x_i,\ldots,x_N\},\{\alpha_i,\ldots,\alpha_N\}}{\arg\min} \quad obj_{proc}(fit(h, x_i, \alpha_i), \mathcal{D}_{test})$$

and

$$\underset{\{x_i,\ldots,x_N\},\{\alpha_i,\ldots,\alpha_N\}}{\arg\min} \quad obj_{error}(fit(h, x_i, \alpha_i), \mathcal{D}_{test}) \tag{3}$$

Table 2. Top 100 extracted static-based API Call features for each analyzed Android APK file based on their information gain.

Description	Quantity
API Calls for Object and Basic Types Manipulation	36
API Calls for UI and Layouts	25
API Calls for Data Manipulation	19
API Calls for System Resources	9
API Calls for Data Structures and Collections	6
API Calls for Communication and Integration	5

Table 3. Top 100 extracted static-based Opcode features for each analyzed Android APK file based on their information gain.

Description	Quantity
Opcodes related to object manipulation	25
Opcodes related to arithmetic and logic operations	17
Opcodes related to conditional and branching instructions	13
Opcodes related to data movement	10
Opcodes related to memory and constants	10
Opcodes related to type conversions and comparisons	9
Opcodes related to method invocation	9
Opcodes related to return instructions	4
Opcodes related to synchronization and exceptions	3

Table 4. Top 100 extracted static-based Permission features for each analyzed Android APK file based on their information gain.

Description	Quantity
Permissions related to miscellaneous system features	17
Permissions related to system features and management	16
Permissions related to network and internet access	14
Permissions related to user data and accounts	12
Permissions related to phone and communication services	11
Permissions related to hardware and sensors	10
Permissions related to device storage and filesystems	8
Permissions related to notifications and user interactions	7
Permissions related to location services	5

where fit denotes a function that train the ensemble h using the selected views subspaces x_i and classifiers hyperparameters α_i. Therefore, our scheme aims to find each view's feature subspace and classifier hyperparameters that optimize the resulting ensemble inference processing time and accuracy. Our proposed model can consider the application of multi-view Android malware detection and the resulting ensemble accuracy. Our insight is to select each view feature subspace while measuring the resulting ensemble precision and processing costs. How we solve Eq. 3 is described in Section 6.

4.3 Discussion

Our proposed multi-view Android malware detection model addresses the single-view limitation in current approaches by leveraging a multi-objective optimization framework. To achieve such a goal, we analyze multiple complementary aspects of Android APK files, such as permissions, Opcodes, and API calls, to improve generalization and reliability in real-world settings. This multi-view approach not only enhances classification accuracy but also ensures processing costs remain feasible for deployment on resource-constrained devices. Our model applies a genetic search algorithm for multi-objective optimization, which iteratively refines feature selection and classifier hyperparameters to balance accuracy with processing costs. The use of a simple majority voting mechanism to combine classification results across views further contributes to a lightweight design, allowing for minimal inference cost while maintaining high detection reliability. As a consequence, our proposed scheme provides a reliable and lightweight approach to Android malware detection.

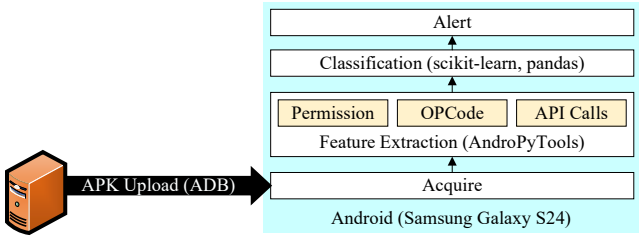


Figure 3. Prototype overview.

5 Prototype

We implemented our proposed multi-view classification scheme (see Fig. 1) as a Android application. Figure 3 illustrates the overview of our implemented prototype.

We consider a proposal prototype implemented as an application executed on an Android device on Samsung Galaxy S24. The application receives as input the to-be-analyzed APK through a Android Debug Bridge (ADB) interface. The received APK file is analyzed through AndroPyTool [Martín *et al.*, 2019], where we consider three complementary views, namely *Permission*, *OPCode*, and *API Calls*. In practice, we considered three views as follows:

- ***API Calls***. A total of $63,460$ features that comprise the number of every API call conducted by the analyzed APK source code binary (*dex*);
- ***Opcode.*** A total of 224 features counting the number that each Opcode occurred on the APK source code binary (*dex*);
- ***Permissions.*** A total of $19,083$ features that assesses which permissions were requested by the evaluated APK (*manifest.xml*);

Each view is further summarized in Tables 2, 3, and 4. Here, we only list the top 100 features for each view based on their information gain (latter discussed in Section 6.3). As a result, our proposed prototype implementation enables the evaluation of our multi-view scheme as implemented in a real Android device. In addition, we also assess the behavior of the analyzed APK file through a multi-view representation, as depicted by the *Permission*, *OPCode*, and *API Calls* views. The resulting views are preprocessed using Pandas API $v.2.2.3$. Finally, we classify the APK through the classifiers implemented through scikit-learn $v.1.5.2$.

6 Evaluation

In this section, we investigate the performance of our proposed scheme. More specifically, we aim to answer the following Research Questions (RQs):

- ***RQ1***: *What is the accuracy performance of traditional single-view ML-based Android malware detection?*
- ***RQ2***: *How does our proposed multi-objective optimization improve classification accuracy?*
- ***RQ3***: *What is the classification impact of each used view?*
- ***RQ4***: *What are the processing tradeoffs of our scheme when implemented on a smartphone?*

The following subsections further describe our model-building procedure and its performance.

6.1 A Realistic Android Malware Dataset

To reliably evaluate the performance of our proposed scheme in a multi-view setting, we constructed a new Android malware dataset tailored to the challenges of real-world detection scenarios. For this purpose, we collected approximately $40,000$ Android APK files from the AndroZoo repository [Allix *et al*., 2016], spanning a representative period from 2022 to 2024. The dataset was carefully balanced to include $20,000$ samples labeled as *goodware* and $20,000$ as *malware*, ensuring a fair and comprehensive evaluation.

Each collected APK file was subjected to a labeling process using the VirusTotal API [Virustotal, 2024], a widely recognized tool for malware analysis. A file was classified as *malware* if at least two antivirus solutions in VirusTotal flagged it as malicious. Conversely, files not meeting this criterion were labeled as *goodware*. This labeling strategy provides a pragmatic balance between accuracy and scalability, leveraging the consensus of multiple antivirus engines to enhance reliability while managing the practical constraints of handling a large dataset. As a result, the built dataset represents our evaluation framework, reflecting the diversity of modern Android applications and malware. In practice, it enables the evaluation of our scheme's ability to generalize and reliably detect threats in a multi-view classification setting.

The resulting dataset was divided into three distinct subsets: *training*, *validation*, and *testing*, comprising 40%, 30%, and 30% of the total APKs, respectively. The split was performed through random selection without replacement to ensure no overlap between the datasets and to maintain data integrity. The *training* dataset was utilized to construct the single classifiers employed in our proposed scheme. The *testing* dataset was used in the multi-objective optimization process, guiding the selection of feature subsets and classifier hyperparameters based on accuracy and processing costs. Meanwhile, the *validation* dataset was reserved for evaluating the final system's accuracy, ensuring an unbiased assessment of the model's generalization capabilities. This structured partitioning ensures that our dataset provides a comprehensive representation of Android APK behaviors, making it useful for assessing the robustness and effectiveness of ML-based malware detection techniques in multi-view settings.

6.2 Model Building

To evaluate our proposed scheme, we selected five widely used classifiers for Android malware detection: Decision Tree (DT), Random Forest (RF), k-Nearest Neighbor (kNN), Adaboost (Ada), and Bagging (Bag). The DT classifier was implemented using the *gini* index as a node quality measure, with no restrictions on the maximum tree depth, allowing it to fully grow as required by the data. The RF classifier utilized an ensemble of 100 decision trees, each configured identically to the single DT, to improve classification performance through majority voting. The kNN classifier was configured with 5 neighbors, employing the Euclidean distance metric for similarity calculations. For kNN, the dataset was normalized using the *min-max* scaling procedure to ensure fair distance computations. The Ada classifier was implemented with 100 estimators and a learning rate of 1.0. Lastly, the Bag was evaluated with 100 estimators and sample selection with resample. All classifiers were implemented using the *scikit-learn* API $v0.24$.

Table 5. Accuracy performance of selected ML-based Android Malware detection schemes

View	Classifier	Accuracy (%)		
		TP	***TN***	***F1***
API Calls	DT	76.33	74.33	0.75
	RF	77.38	83.96	0.80
	kNN	75.55	74.11	0.75
	Adaboost	69.51	68.91	0.69
	Bagging	77.58	83.36	0.79
Opcode	DT	76.66	72.83	0.75
	RF	75.50	84.91	0.80
	kNN	75.00	73.08	0.74
	Adaboost	64.96	70.00	0.66
	Bagging	76.18	83.26	0.78
Permissions	DT	62.25	74.00	0.66
	RF	65.50	73.26	0.69
	kNN	67.60	60.68	0.65
	Adaboost	54.26	68.53	0.58
	Bagging	54.26	68.53	0.67
Ours	DT	77.45	78.53	0.77
	RF	78.96	84.53	0.81
	kNN	79.63	78.93	0.79
	Adaboost	80.96	50.63	0.70
	Bagging	79.80	82.78	0.81

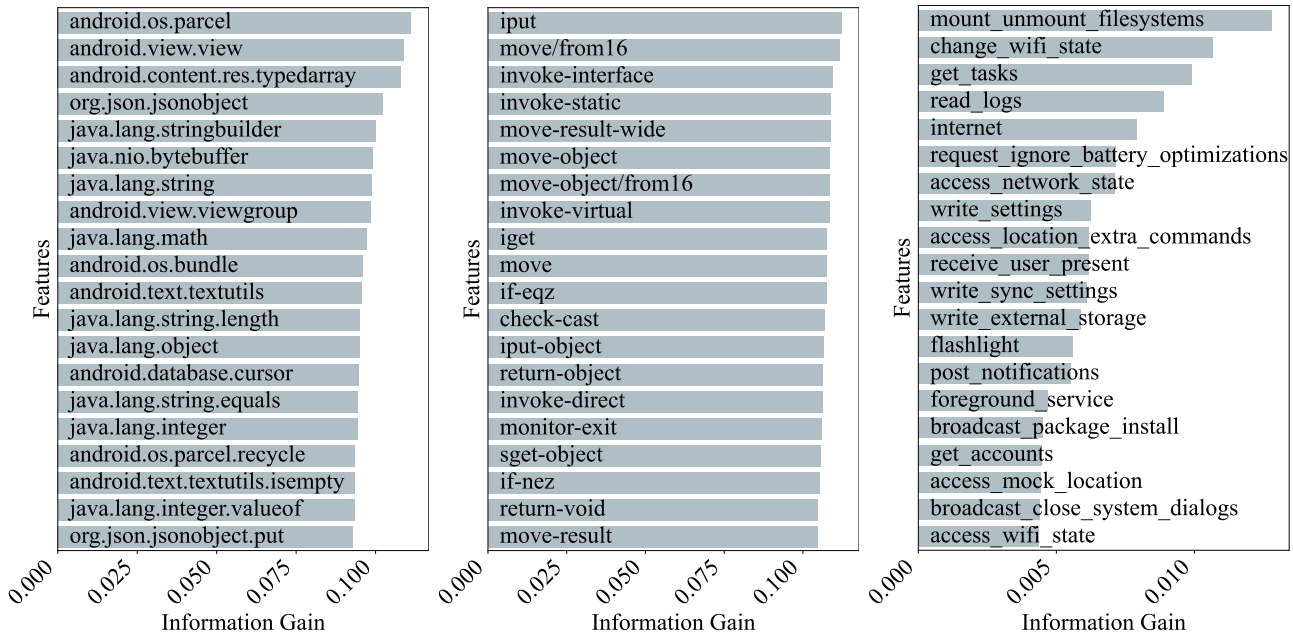
The performance of the classifiers was assessed using their True Positive (TP) and True Negative (TN) rates. The TP rate represents the proportion of *malware* samples correctly identified as *malware*, while the TN rate represents the proportion of *goodware* samples correctly classified as *goodware*. In addition, we measure the F-Measure according to the harmonic mean of precision and recall values while considering *malware* samples as positive and *goodware* samples as negative, as shown in Eq. 6.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

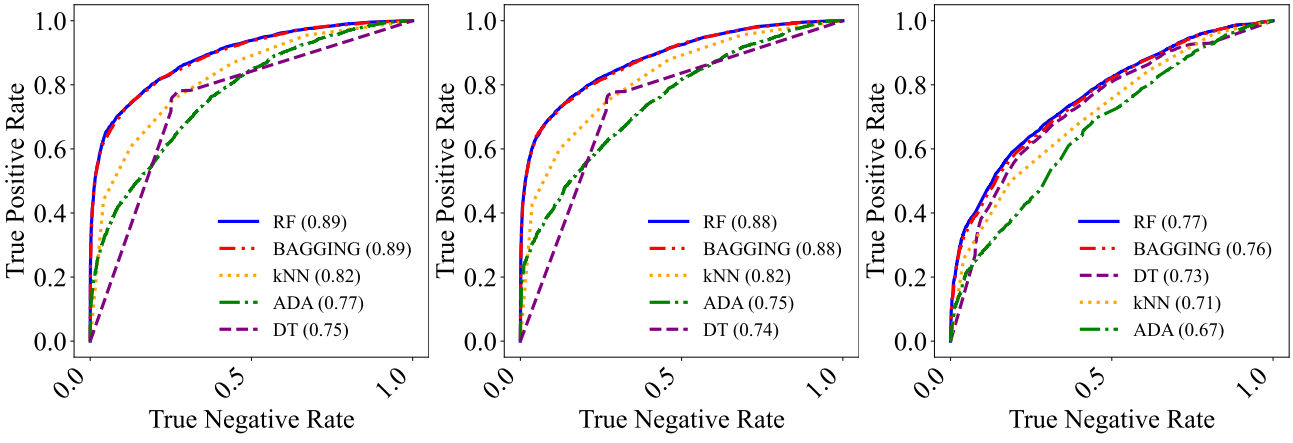
$$F\text{-}Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \tag{6}$$

This evaluation ensures a comprehensive assessment of the



(a) API Calls View **(b)** Opcode View **(c)** Permissions View

Figure 4. Top 20 features on each view's as ranked based on their information gain. For model training purposes, we consider the top 100 features based on their information gain.



(a) API Calls View **(b)** Opcode View **(c)** Permissions View

Figure 5. Classification performance of selected classifiers in a single-view operation on our dataset.

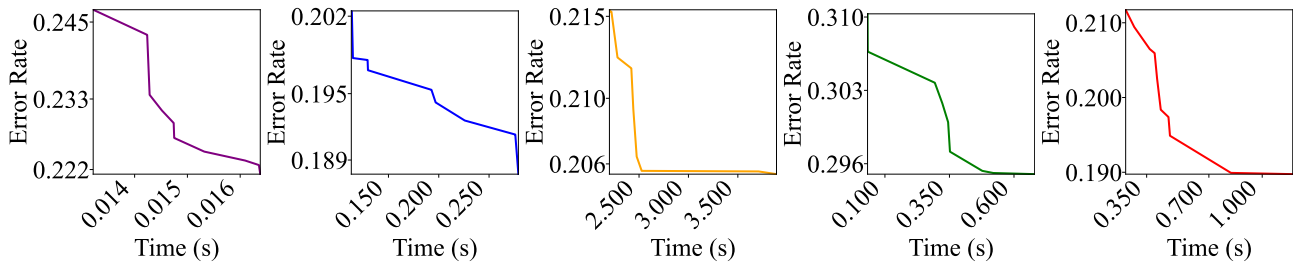
classifiers' ability to detect malware and maintain reliability in distinguishing benign applications.

6.3 Multi-view Android Malware Detection

Our first experiment is designed to address *RQ1* by evaluating the accuracy performance of traditional ML-based Android malware detection techniques. To this end, we assessed the accuracy of the selected classifiers (as described in Section 6.2) within a single-view classification setting. In practice, we train the selected classifiers considering a single-view setting using the *training* dataset and evaluate their performance on the *validation* dataset.

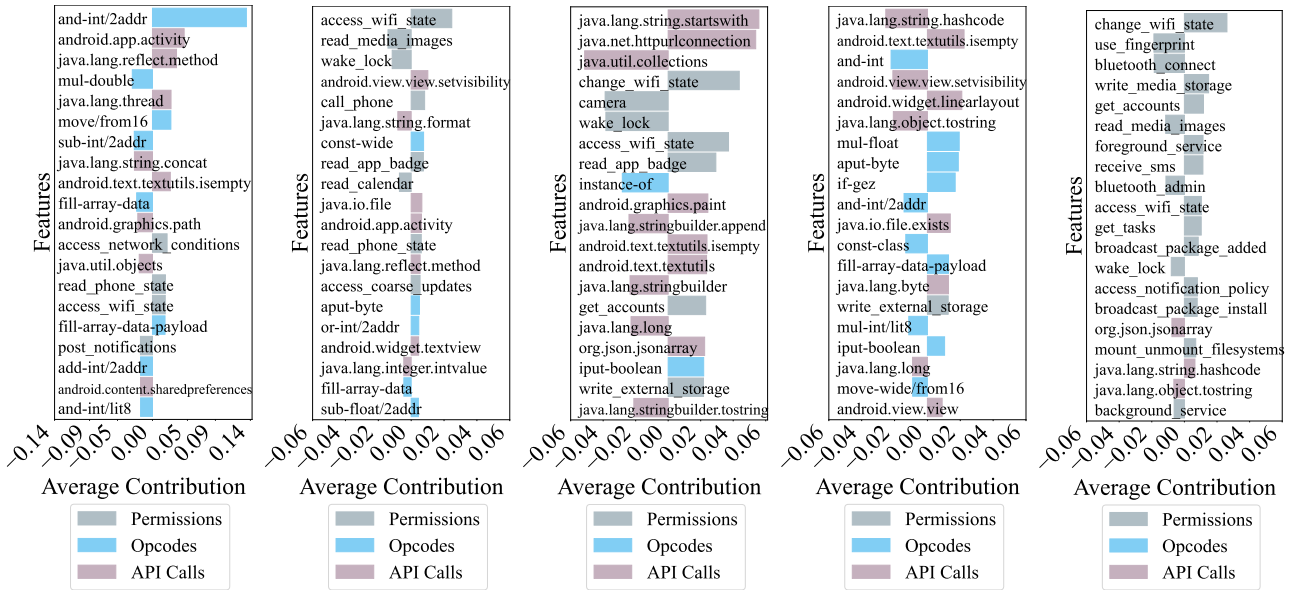
Before the training phase, we perform a filter-based feature selection to remove irrelevant features from each view, ensuring the model operates with the most informative attributes. To achieve this, we compute the mutual information of each feature across all views relative to the dataset labels within the training dataset. This process allows us to quantify the relevance of each individual feature concerning the label. We then rank these features and retain each view's top 100 features, focusing on those with the highest mutual information. These top 100 features are used throughout the remaining experiments for both the traditional single-view and our proposal. Figure 4 illustrates each view's information gain for the top 20 features. The results highlight variability in feature informativeness across views, with the highest information gain reaching up to 0.12, 0.11, and 0.12 for the *API Calls*, *Opcode*, and *Permissions* views, respectively. This variability shows the value of multi-view analysis in capturing diverse aspects of Android malware behavior.

Table 5 presents the classification accuracy achieved by the selected techniques when applied to individual views. It is possible to note that the classifiers exhibit relatively low accuracy rates on average. For example, the RF classifier achieved F1-Scores of 0.80, 0.80, and 0.69 for the *API Calls*, *Opcode*, and *Permissions* views, respectively. Similar findings were found when other classifiers were considered. As a result,



(a) Decision Tree (DT) **(b)** Random Forest (RF) **(c)** k-Nearest Neighbor (kNN) **(d)** Adaboost (Ada) **(e)** Bagging (Bag)

Figure 6. Pareto curve of our proposed model for each evaluated classifier in a multi-view setting.



(a) Decision Tree (DT) **(b)** Random Forest (RF) **(c)** k-Nearest Neighbor (kNN) **(d)** Adaboost (Ada) **(e)** Bagging (Bag)

Figure 7. Explainable predictions using LIME on our proposed model for each evaluated classifier in a multi-view setting. Note how each view impact on the resulting model's decision boundary. Positive values reflect the impact of correctly classifying malware, while negative values reflect the impact of correctly classifying goodware.

ML-based Android malware detection techniques relying on single-view classification struggle to achieve the necessary level of reliability for practical deployment in production environments.

Figure 5 shows the Receiver Operating Characteristic (ROC) curve of the selected single-view classifiers. Similarly, most selected single-view approaches provide relatively small Area Under the Curve (AUC) scores regardless of the used classification view. As an example, the RF classifier achieved AUC scores of 0.89, 0.88, and 0.77 for the *API Calls*, *Opcode*, and *Permissions* views, respectively. Therefore, regardless of the used classifier single-view approaches cannot meet high-reliability rates.

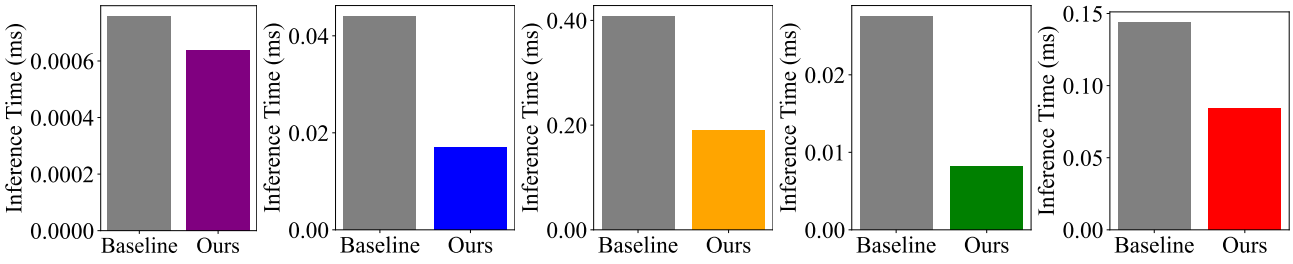
Our second experiment is designed to address *RQ2* by evaluating the classification accuracy of our proposed model. To achieve this objective, we first assess the performance of our scheme implemented with the multi-objective optimization framework described in Section 4.2. The scheme employs a wrapper-based feature selection approach, leveraging the *Non-dominated Sorting Genetic Algorithm* (NSGA-III) [Deb *et al.*, 2002]. The optimization process is executed using the *pymoo* API, which facilitates efficient handling of multi-objective optimization tasks. This setup enables simultaneous optimization of classification accuracy and inference processing costs, ensuring a balance between performance and computational efficiency.

To implement our proposed multi-objective optimization model (see Alg. 1) we vary each of the selected classifiers hyper-parameters as follows:

- Decision Tree (DT). Maximum tree depth is varied from 1 to 50;
- Random Forest (RF). The number of trees is varied from 5 to 100;
- k-Nearest Neighbor (kNN). The number of neighbors is varied from 1 to 10;
- Adaboost (Ada). The number of estimators is varied from 5 to 100;
- Bagging (Bag). The number of estimators is varied from 5 to 100;

Here, each evaluation uses the same classifier applied for each view. The goal is to compare the improvement of our proposal when considering a single-view setting $vs.$ a multi-view implementation (see Fig. 1). As an example, we assess the performance of our proposal when using the RF classifier for both the *API Calls*, *Opcode*, and *Permissions* views. Recalling that, we combine each decision through a simple majority voting process. The results of this evaluation provide insights into the effectiveness of our proposed multi-view classification model in addressing the inherent limitations of single-view approaches.

The NSGA-III implementation uses a 100 population size,



(a) Decision Tree (DT) **(b)** Random Forest (RF) **(c)** k-Nearest Neighbor (kNN) **(d)** Adaboost (Ada) **(e)** Bagging (Bag)

Figure 8. Average inference processing time of our prototype executed in a smartphone device (see Fig. 3).

100 generations, a crossover of 0.3, and a mutation probability of 0.1. The multi-objective optimization aims at decreasing the obj_{proc} (Eq. 1) and obj_{error} (Eq. 2) as measured on the validation dataset. As a result, the procedure aims at solving Eq. 3, by selecting for each individual classifier the best hyper-parameters (α), and feature subspace (x_i), that decreases the obj_{proc} and obj_{error} when combined in an ensemble. To this goal, we evaluate our scheme without varying the underlying used classifier view (Fig. 1, *Classifier View 1* to *N*). This characteristic allows us to compare the performance of our proposal $vs.$ the traditional scheme evaluated previously.

Figure 6 presents the Pareto curve of our proposed model for each selected classifier, illustrating the trade-off between inference processing costs and the system's error rate. The curve highlights a direct relationship between these two objectives, where reducing one often increases the other. In practical deployment scenarios, operators must carefully select the operational point that aligns best with their application's requirements, balancing performance and resource constraints. For our evaluation, we selected the operation point nearest to zero on both objectives, representing the optimal balance between minimizing processing costs and error rate within the tested configurations.

Using the selected operation points, we analyzed the accuracy performance of our proposed model. Table 5 summarizes the classification accuracy of our scheme for each evaluated classifier. The results demonstrate that our approach enhances accuracy across all classifiers compared to single-view implementations. For instance, when using the RF classifier, our scheme improved the F1 score by 0.01, 0.01, and 0.11 compared to single-view setups employing the *API Calls*, *Opcode*, and *Permissions* views, respectively. On average, our approach increased TP rates by 18.6%, 5.7%, and 4.1% when compared to the single-view strategy as implemented with the API Calls, Opcodes, and permission views respectively. These findings highlight the effectiveness of our multi-objective optimization strategy in boosting classification accuracy, making it a robust solution for Android malware detection in multi-view settings.

To answer *RQ3*, we further investigate how each view can impact the decision of our proposed multi-view model. To achieve such a goal, we make use of LIME technique [Ribeiro *et al.*, 2016], and evaluate the influence of each feature on the decision boundary for each selected classifier. The goal is to investigate how selected views can influence the classification outcome of the resulting classifier. Figure 7 shows the explanations for each classifier when operating in a multi-view setting for a sample *malware* event. It is possible to note that for every classifier, the utilization of multiple views can affect the final outcome. As an example, the RF classifier makes use of *API Calls*, *Opcode*, and *Permissions* views to deem the evaluated event as *malware*. In this case, the decision can only be reliably conducted because the classifier makes use of the multi-view implementation.

Finally, to answer *RQ4* we investigate the inference processing costs of our scheme when compared to the traditional ensemble approach. To achieve such a goal, we assess the inference processing costs without using our proposed multi-objective optimization (all features) $vs.$ the processing costs with the features and classifier hyper-parameters selected by our model (Table 5). Figure 8 shows the processing time of our proposal $vs.$ the traditional approaches. On average, our proposed model required only 14%, 56%, 53%, 72%, and 42% of the processing costs demanded by its all-features counterpart with the DT, RF, kNN, Ada, and Bag classifiers, respectively. Consequently, our proposed scheme can improve the overall system accuracy and significantly decrease the associated inference processing costs.

7 Conclusion

Android malware detection through ML-based techniques has been a widely explored topic in the literature over the past few years. Surprisingly, despite the promising results, current approaches are rarely used in production. This paper addressed such a challenge through a multi-view Android malware classification implemented via multi-objective optimization. Our proposed model selects the best subset of features from each view that improves classification accuracy when combined by an ensemble of classifiers. Experiments conducted on a new dataset attest to our proposal's feasibility, significantly improving accuracy and decreasing inference computational costs. In future work, we plan on extending our proposed model to make use of deep learning classifiers combined with multi-view classification and feature selection.

Declarations

Funding

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants nº 304990/2021-3, 407879/2023-4, and 302937/2023-4.

Author's Contribution

Philipe Fransozi: Conceptualization, Methodology, Investigation, Writing - original draft. **Jhonatan Geremias**: Supervision, Validation. **Eduardo K. Viegas**: Supervision, Methodology, Writing - review, and editing. **Altair O. Santin**: Supervision, Writing - review, and editing.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The source code and dataset are publicly available at https://github.com/pFransozi/lightweight-android-malware-detection.

References

Alani, M. M. and Awad, A. I. (2022). Paired: An explainable lightweight android malware detection system. *IEEE Access*, 10:73214–73228. DOI: 10.1109/access.2022.3189645.

Alani, M. M., Mashatan, A., and Miri, A. (2023). Xmal: A lightweight memory-based explainable obfuscated-malware detector. *Computers amp; Security*, 133:103409. DOI: 10.1016/j.cose.2023.103409.

Allix, K., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). Androzoo: Collecting millions of android apps for the research community. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 468–471.

Azad, M. A., Riaz, F., Aftab, A., Rizvi, S. K. J., Arshad, J., and Atlam, H. F. (2022). Deepsel: A novel feature selection for early identification of malware in mobile applications. *Future Generation Computer Systems*, 129:54–63.

Bhat, P., Behal, S., and Dutta, K. (2023). A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. *Computers & Security*, 130:103277.

Costa, L. d. and Moia, V. (2023). A lightweight and multi-stage approach for android malware detection using non-invasive machine learning techniques. *IEEE Access*, 11:73127–73144. DOI: 10.1109/access.2023.3296606.

Cui, Y., Sun, Y., and Lin, Z. (2023). Droidhook: a novel api-hook based android malware dynamic analysis sandbox. *Automated Software Engineering*, 30(1). DOI: 10.1007/s10515-023-00378-w.

Curry, D. (2024). Android statistics (2024). *https://www.businessofapps.com/data/android-statistics/*.

Darwaish, A. and Nait-Abdesselam, F. (2020). Rgb-based android malware detection and classification using convolutional neural network. In *IEEE Global Communications Conference*.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Gao, C., Du, Y., Ma, F., Lan, Q., Chen, J., and Wu, J. (2024). A new adversarial malware detection method based on enhanced lightweight neural network. *Computers amp; Security*, 147:104078. DOI: 10.1016/j.cose.2024.104078.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2022). Towards multi-view android malware detection through image-based deep learning. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, page 572–577. IEEE. DOI: 10.1109/iwcmc55113.2022.9824985.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2023). Towards a reliable hierarchical android malware detection through image-based cnn. In *2023 IEEE 20th Consumer Communications amp; Networking Conference (CCNC)*, page 242–247. IEEE. DOI: 10.1109/ccnc51644.2023.10060381.

Guerra-Manzanares, A., Bahsi, H., and Nõmm, S. (2021). Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers amp; Security*, 110:102399. DOI: 10.1016/j.cose.2021.102399.

Hijazi, N., Aloqaily, M., Ouni, B., Karray, F., and Debbah, M. (2023). Harris hawks feature selection in distributed machine learning for secure iot environments. In *ICC 2023 - IEEE International Conference on Communications*. IEEE. DOI: 10.1109/icc45041.2023.10279042.

Kaspersky (2024). Attacks on mobile devices significantly increase in 2023. *https://www.kaspersky.com/about/press-releases/2024_attacks-on-mobile-devices-significantly-increase-in-2023*.

Kyadige, A., Rudd, E. M., and Berlin, K. (2020). Learning from context: A multi-view deep learning architecture for malware detection. In *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE.

Li, C., Lv, Q., Li, N., Wang, Y., Sun, D., and Qiao, Y. (2022). A novel deep framework for dynamic malware detection based on api sequence intrinsic features. *Computers amp; Security*, 116:102686. DOI: 10.1016/j.cose.2022.102686.

Ma, R., Yin, S., Feng, X., Zhu, H., and Sheng, V. S. (2024). A lightweight deep learning-based android malware detection framework. *Expert Systems with Applications*, 255:124633. DOI: 10.1016/j.eswa.2024.124633.

Mahindru, A. and Sangal, A. L. (2020). Mldroid—framework for android malware detection using machine learning techniques. *Neural Computing and Applications*, 33(10):5183–5240. DOI: 10.1007/s00521-020-05309-4.

Martín, A., Lara-Cabrera, R., and Camacho, D. (2019). Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142. DOI: 10.1016/j.inffus.2018.12.006.

Millar, S., McLaughlin, N., Martinez del Rincon, J., and Miller, P. (2021). Multi-view deep learning for zero-day android malware detection. *Journal of Information Security and Applications*, 58:102718. DOI: 10.1016/j.jisa.2020.102718.

Millar, S., McLaughlin, N., Martinez del Rincon, J., Miller, P., and Zhao, Z. (2020). Dandroid: A multi-view discriminative adversarial network for obfuscated an-

droid malware detection. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, CODASPY '20, page 353–364. ACM. DOI: 10.1145/3374664.3375746.

Molina-Coronado, B., Mori, U., Mendiburu, A., and Miguel-Alonso, J. (2023). Towards a fair comparison and realistic evaluation framework of android malware detectors based on static analysis and machine learning. *Computers & Security*, 124:102996.

Odat, E. and Yaseen, Q. M. (2023). A novel machine learning approach for android malware detection based on the co-existence of features. *IEEE Access*, 11:15471–15484. DOI: 10.1109/access.2023.3244656.

Pan, Y., Ge, X., Fang, C., and Fan, Y. (2020a). A systematic literature review of android malware detection using static analysis. *IEEE Access*, 8:116363–116379. DOI: 10.1109/access.2020.3002842.

Pan, Y., Ge, X., Fang, C., and Fan, Y. (2020b). A systematic literature review of android malware detection using static analysis. *IEEE Access*, 8:116363–116379.

Pektaş, A. and Acarman, T. (2020). Learning to detect android malware via opcode sequences. *Neurocomputing*, 396:599–608. DOI: 10.1016/j.neucom.2018.09.102.

Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., and Xiang, Y. (2020). A survey of android malware detection with deep neural models. *ACM Computing Surveys*, 53(6):1–36.

Ravi, V., Alazab, M., Selvaganapathy, S., and Chaganti, R. (2022). A multi-view attention-based deep learning framework for malware detection in smart healthcare systems. *Computer Communications*, 195:73–81.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144.

Seraj, S., Khodambashi, S., Pavlidis, M., and Polatidis, N. (2022). Hamdroid: permission-based harmful android anti-malware detection using neural networks. *Neural Computing and Applications*, 34(18):15165–15174.

Smith, M. R., Johnson, N. T., Ingram, J. B., Carbajal, A. J., Haus, B. I., Domschot, E., Ramyaa, R., Lamb, C. C., Verzi, S. J., and Kegelmeyer, W. P. (2020). Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, CCS '20. ACM.

Virustotal (2024). Virustotal - analyze suspicious files. *https://www.virustotal.com/*.

Wu, Y., Li, M., Zeng, Q., Yang, T., Wang, J., Fang, Z., and Cheng, L. (2023). Droidrl: Feature selection for android malware detection with reinforcement learning. *Computers & Security*, 128:103126.

Şahin, D. O., Kural, O. E., Akleylek, S., and Kılıç, E. (2021). A novel permission-based android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, 35(7):4903–4918.