A Multi-view Android Malware Detection Model Through Multi-objective Optimization

Philipe Fransozi, Jhonatan Geremias, Eduardo K. Viegas
Graduate Program in Computer Science
Pontifical Catholic University of Parana, Brazil
{philipe.hfransozi, jgeremias, eduardo.viegas}@ppgia.pucpr.br

***Abstract*—Over the past few years, several highly accurate machine learning (ML) techniques have been proposed for Android malware detection. Unfortunately, proposed schemes are rarely used in production, a situation usually caused by their limited generalization capabilities, leading to low accuracies when deployed. This paper proposes a new multi-view Android malware detection model, implemented in two stages. First, we extract multiple feature sets from an analyzed Android application package. The feature sets provide a complementary Android app behavioral vector for the classification task, enhancing the system's generalization. Secondly, we conduct a multi-objective optimization to select the optimal feature subset from each view for subsequent ensemble-based classification. Our proposal's insight is to proactively select each feature subset that simultaneously improves accuracy and reduces processing requirements in a multi-view setting. Experiments on our new dataset, comprising over 40 thousand Android app samples, demonstrated the feasibility of our proposal. Our scheme can improve true-positive rates by an average of 4.4 while demanding only up to 65% of inference processing costs.**

***Index Terms*—Android, Malware Detection, Machine Learning, Multi-objective Optimization**

I. Introduction

Android is the world's most used mobile operating system (OS), with over 3 billion active users, accounting for almost three-quarters of the current market share [1]. Unfortunately, the number of malicious Android applications, namely malware, also increases over time. According to a cybersecurity report [2], in 2023 alone, the number of identified Android malware samples grew by 52%, with several even making their way into official app stores. In practice, current approaches used to detect Android malware have failed to secure users effectively.

Android malware detection generally follows a dynamic or static implementation [3]. On the one hand, dynamic-based techniques search for malicious footprints during the targeted application execution. Consequently, they usually require significant effort for the detection task, with some apps even hiding their behavior while being monitored to evade detection. On the other hand, static-based approaches conduct their detection task by analyzing the associated app files. As a result, they are often preferred in the literature due to their easier-to-conduct nature and the promising results reported [4].

Static-based Android malware detection is typically performed by analyzing the Android Android Application Package (APK) files. These techniques may include analyzing the app's requested permissions (*manifest.xml*), as well as the binary source code (*.dex*), to extract associated opcodes and conducted API calls [5]. Over the last few years, several approaches have been proposed in the literature for the classification task, where authors typically resort to Machine Learning (ML)-based techniques. In such cases, the extracted app behavioral feature vector is used as input to an ML model, which classifies it as either *goodware* or *malware*.

Despite the promising results reported in the literature, such as high classification accuracies, current ML-based techniques are rarely used in production [6]. This is because the behavior of Android malware is typically characterized by several malicious footprints, which often necessitates the analysis of multiple files for the classification task. For instance, an analyzed APK may request several highly sensitive app permissions while still using them correctly when its source code is analyzed [7]. If the classification is based solely on the requested permissions, it may incorrectly classify it as malware. Surprisingly, the vast majority of current approaches in the literature often rely on analyzing a single APK file for the classification task [8].

Combining multiple feature sets for the Android classification task is the subject of several works in the literature [5]. In general, proposed schemes resort to Deep Neural Network (DNN)-based techniques, which usually enhance system accuracy, but they come with significant trade-offs regarding memory and processing requirements [9]. Nevertheless, selecting the optimal features to consider in a multi-view setting is challenging. This is because each used classifier must take into account the resulting pool classification performance during the feature selection process.

Contribution. In light of this, this paper proposes a new multi-objective optimization model for multi-view classification of Android malware, implemented in two stages. First, we extract multiple feature sets from an analyzed Android APK. Each feature set is extracted based on an associated APK file, providing a complementary Android app behavioral vector for the classification task. Second, we conduct a multi-objective optimization to select the optimal feature subset from each feature set for subsequent ensemble-based classification. Our proposal's insight is to select the optimal features that can simultaneously improve both accuracy and reduce inference processing requirements in a multi-view setting. As a result, our proposed scheme can leverage complementary feature sets for Android malware classification with lower processing trade-offs.

In summary, the main contributions of our work are:

- A new publicly available multi-view Android malware dataset with 3 complementary feature sets. The dataset was built through the analysis of ≈ 40 thousand Android samples;
- A new multi-view Android malware detection model implemented through a multi-objective optimization strategy. Our proposed scheme can improve true-positive accuracy by an average of 4.4 while also demanding only up to 65% of processing costs;

Roadmap. The remainder of this paper is organized as follows. Section II further describes ML-based Android malware detection. Section III describes the related works. Section IV presents our proposal, while Section V evaluates its performance. Section VI concludes our work.

II. Background

This section further discusses the detection of Android malware through ML-based techniques. More specifically, we first describe how ML can be applied to detect Android malware. Then, we delve into the challenges associated with their application.

A. Static-based Android Malware Detection

The detection of Android malware through ML-based techniques is typically implemented following a four-phase process [3]. First, the *Data Extraction* module obtains the associated APK files. For example, through the extraction of the evaluated Android app requested permissions (*manifest.xml*). Then, the behavior of the obtained file is extracted by a *Feature Extraction* module, creating a feature vector. The format of the feature vector varies according to the evaluated file. For example, the permission file typically generates a feature vector consisting of a list of permissions requested or not requested by the evaluated app. The resulting feature vector is classified by a *Classification* module, which, using a previously trained ML model, categorizes it as either *goodware* or *malware*. Finally, *malware*-classified samples are signaled through an *Alert* module.

The successful application of ML-based detection of Android malware relies on utilizing a representative training dataset [6]. This is because the ML model is built by analyzing the behavior available in the training dataset. Hence, it must provide a significant variability representation of the available training samples. Surprisingly, most of the literature relies on outdated training datasets, often containing hundreds to a few thousand malware samples.

B. Challenges of ML-based Malware Detection

Over the past few years, several works have proposed highly accurate ML-based techniques for Android malware detection [7]. However, despite the promising results, proposed schemes are rarely used in production environments. In practice, designing a reliable ML-based scheme requires a large number of training samples from the dataset. This is because the built classifier must be able to adequately generalize the behavior from the training data to that observed during production deployment.

Providing a well-generalized ML model is challenging, especially when a single-event view is used. Android malware can typically only be identified through the evaluation of multiple complementary behaviors [5]. For example, a sensitive permission can only be considered malware-related if the associated source code uses it for malicious intent. In such a scenario, the designed ML-based technique must be able to account for multiple views during the classification task, a requirement that usually comes at the expense of more computational requirements.

III. Related Work

Android malware detection through ML-based techniques has been a widely explored topic in the literature over the last years [3]. The proposed approaches aim to provide high accuracy for a given test dataset. For instance, D. O. Sahin *et al.* [10] makes use of a feature selection technique on a permission-based ML model for malware detection. Their approach improves classification accuracy but uses an outdated dataset and few samples. S. Seraj *et al.* [11] proposes an updated dataset with more malware samples. Their approach uses a DNN technique and improves accuracy compared to the literature. However, the authors overlook how complementary views can improve their system generalization. A. Pektas *et al.* [12] proposes using opcode sequences for malware detection. Their approach improves accuracy when combined with feature selection. Similarly, the authors overlook how multi-view can be used to improve reliability. Another approach proposed by A. Darwaish *et al.* [5] translates the source code binary to an image for the classification task. The author's proposal increases accuracy but also overlooks the application of multiple feature sets.

Combining multiple feature sets for Android malware detection have also been the subject of several works over the past years. S. Millar *et al.* [13] proposes using Ppcodes, permissions, and API packages for the classification task. Their proposal improves classification accuracy but overlooks how views can be optimized to be combined. V. Ravi *et al.* [9] combines multiple views through a DNN-based approach. Their scheme improves classification accuracy but incurs a significant tradeoff in processing costs. A similar approach is proposed by A. Kyadige *et al.* [14], which combines multiple views through a DNN-based scheme. Their approach improves classification accuracy but overlooks how views can be combined optimally.

Feature selection has been proposed to improve classification in a multi-view setting. Y. Wu *et al.* [15] uses reinforcement learning to conduct the feature selection task. Their approach improves accuracy but overlooks the application of multiple views. M. Azad *et al.* [16] proposes the application of particle swarm for feature selection on a DNN classifier. Their approach improves accuracy but overlooks the application

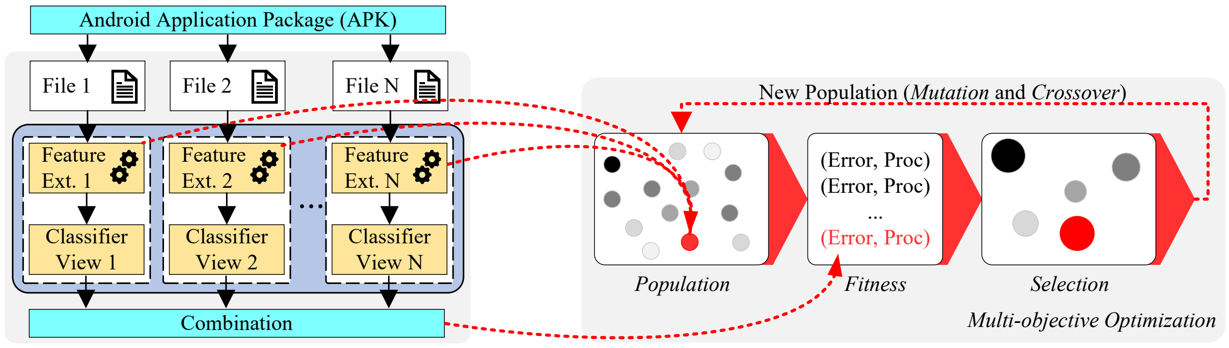


Fig. 1: A multi-view Android detection model through multi-objective optimization. *Left-side* shows the classification pipeline used by our model. *Right-side* shows the feature selection process to combine multiple views.

of multiple views. Similarly, H. Hawks *et al.* proposes a feature selection approach combined with an ensemble of ML classifiers for the detection task. Their approach can increase classification accuracy but neglects the application of multiple views. Therefore, current approaches in the literature usually fail to consider how multi-view can be explored to perform reliable Android malware detection.

IV. A Multi-view Android Malware Detection Model

To address the aforementioned challenge of Android malware detection in a multi-view setting, our proposed scheme is implemented through a multi-objective optimization approach. Figure 1 illustrates the operation of our proposed model.

The proposal considers a multi-view ML-based classification pipeline that operates following a static-based Android app analysis. To achieve such a goal, the Android APK is analyzed through a multi-view process, in which multiple files are used for the classification task (Fig.1, *Files 1* to *N*). As an example, by analyzing the requested permissions (*manifest.xml*) and Opcodes (*dex*). Our main assumption is that multi-view can be used to improve the Android malware detection generalization and reliability. The behavior of each file is extracted by an associated feature extraction module (Fig.1, *Feature Ext. 1* to *N*). Similarly, the resulting feature vectors are classified by an associated classifier, producing a classification outcome for each view (Fig.1, *Classifier View 1* to *N*). To combine the multiple classifications, we make use of a combination module that performs a majority voting to produce the final classification outcome (Fig.1, *Combination*).

To address the challenge of combining multiple classifications based on complementary views, we make use of a multi-objective optimization approach (Fig. 1, *Multi-objective Optimization*). Our goal is to optimize the features used by each view based on their combined classification accuracy and processing costs. Our main insight is that feature selection can be used to enhance malware detection accuracy while also providing a multi-view classification process.

The following subsections further describe our proposed model, including its implementation components.

A. Multi-view Classification

Current approaches in the literature for Android malware detection usually rely on a single view for the classification task. As a consequence, resulting schemes often fail to provide the necessary level of generalization capabilities to be used in production environments. To address such a challenge, our proposed model operates following a multi-view classification process implemented through an ensemble of classifiers.

The operation of our scheme is illustrated in Figure 1. It starts with analyzing a to-be-classified Android APK. In such a case, the APK files used for the feature extraction process are obtained. In practice, our proposed scheme considers three views, namely *Permission* (manifest.xml), *Opcode* (dex), and *API Calls* (dex) (further described in Section V-A). Therefore, the behavior of each analyzed file is extracted by an associated feature extractor to compound a feature vector. Each resulting feature vector is classified by an associated classifier (Fig. 1, *Classifier View 1* to *N*). Finally, to combine the multiple decisions, our scheme uses a *Combination* module that performs a majority voting procedure.

As a result, our proposed model operates following a multi-view process to improve the resulting system generalization and reliability. Our scheme combines the decisions with a simple majority voting procedure, hence keeping inference processing costs minimal. The following subsection further describes how we improve the combination of multiple views.

B. Multi-objective Optimization

To combine multiple views for the Android malware detection task, we make use of a multi-objective optimization approach. In practice, we consider the simultaneous optimization of inference processing time and the resulting ensemble accuracy. To achieve such a goal, we implement the multi-objective optimization as a wrapper-based feature selection through a multi-objective genetic search algorithm (Fig. 1, *Fitness*, *Selection*, *Mutation*, and Crossover). The implementation and used parameters are further described in Section V-C).

Let N be the number of used views and classifiers h. Where each view $x_i \subseteq \mathbb{R}^n$, such that each view x_i is a unique feature space. Our multi-objective optimization task aims at finding a feature subset space for each view such that it minimizes processing time and error rate through the following equations:

$$obj_{proc}(h, x) = \sum_{i=1}^{N} processingTime(h_i(x_i)) \tag{1}$$

$$obj_{error}(h, x) = max(\sum_{i=1}^{N} h_i(x_i)^{goodware}, \sum_{i=1}^{N} h_i(x_i)^{malware}) \tag{2}$$

where obj_{proc} measures the total classifier h inference processing time for all views x_i, and obj_{error} measures the ensemble error rate based on the maximum confidence votes from the ensemble. The obj_{error} measures the ensemble classification outcome according to the maximum sum of the individual classification confidence values for each class, where $h_i(x_i)^{goodware}$ denotes the classifier h_i confidence for $goodware$, and $h_i(x_i)^{malware}$, the classifier h_i for $malware$. As a result, our multi-objective optimization approach aims to solve the following equation:

$$\begin{aligned} &\arg\min_{\tilde{x_i},\ldots,\tilde{x_N}} \sum_{j=1}^{\mathcal{D}} obj_{proc}(h, x_j) \\ \text{and} \\ &\arg\min_{\tilde{x_i},\ldots,\tilde{x_N}} 1 - \sum_{j=1}^{\mathcal{D}} accuracy(x_j, obj_{error}(h, x_j)) \end{aligned} \tag{3}$$

where $\mathcal{D}$ is a testing dataset, and $\tilde{x}_i$ is a feature subspace of view x_i. Therefore, our scheme aims to find each view's feature subspace that optimizes the resulting ensemble inference processing time and accuracy. At the latter, we aim to minimize the inverse of the obtained system accuracy $(1 - accuracy)$. As a result, our proposed model can consider the application of multi-view Android malware detection while also considering the resulting ensemble accuracy. Our insight is to select each view feature subspace while measuring the resulting ensemble precision and processing costs. The implementation of Eq. 3 is described in Section V.

V. 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 are the processing tradeoffs of our scheme?*

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

TABLE I: Extracted static-based features for each analyzed Android APK file on our dataset.

View	Quantity	Description
API Calls	22174	API Calls to Android Support packages
	10595	API Calls to Android Widget packages
	3367	API Calls to Android View packages
	2710	API Calls to Android App packages
	1751	API Calls to Java Util packages
	1257	API Calls to Java Time packages
	1254	API Calls to Java Lang packages
	582	API Calls to w3c.dom packages
	284	API Calls to xml.sax packages
	188	API Calls to json packages
	19298	API Calls to other packages
Opcode	50	Opcodes related to math operations
	49	Opcodes related to read and put values, and array operations
	33	Opcodes related to bitwise and, xor, or, shifts operations
	26	Opcodes related to move, jump, call operations
	66	Opcodes related to other operations
Permissions	1332	Permissions related to read, write, set and access
	586	Permissions related to network, internet, bluetooth, nfc, phone and calls
	116	Permissions related to bind
	82	Permissions related to location
	16967	Permissions related to other features

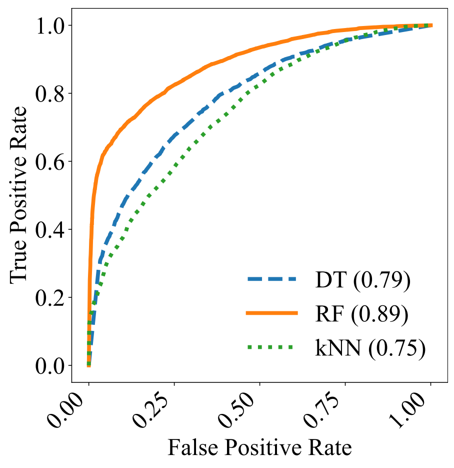
A. A Realistic Android Malware Dataset

To reliably evaluate the performance of our proposed scheme in a multi-view setting, we built a new Android malware dataset. To achieve such a goal, we collected ≈ 40 thousand Android APK files from AndroZoo [17] that were available throughout the year 2022 to 2024, in which 20 thousand are *goodware*, and 20 thousand *malware*. Each collected APK file was labeled through the VirusTotal API [18]. We labeled each APK as *malware* when at least 2 VirusTotal antivirus solution flagged them as such. Otherwise, they were assumed to be *goodware*.

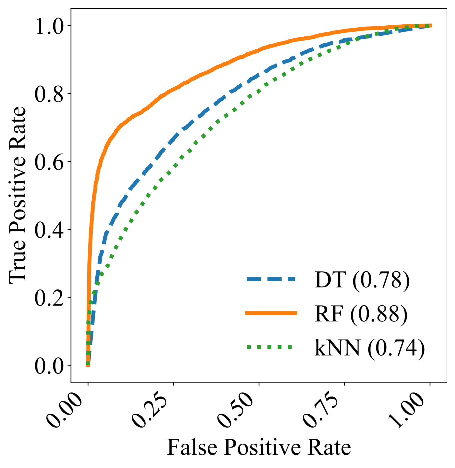
The behavior of each selected APK was extracted through the AndroPyTool [19]. Additionally, we implemented a prototype that standardize the features extracted by AndroPyTool in order to make them suitable for ML tasks. For our experiments, we considered three views as follows:

- ***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*);

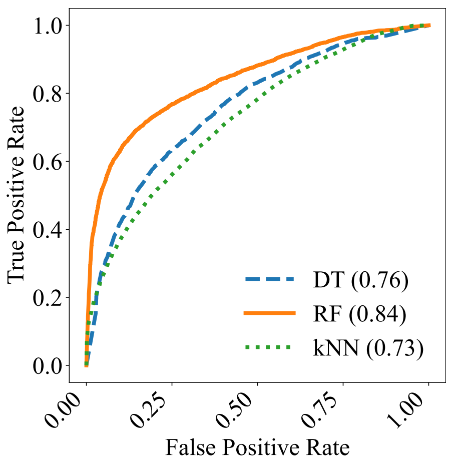
Each view is further summarized in Table I. The resulting dataset was split into three primary datasets, namely *training*, *validation*, and *testing*, each composed of 40%, 30%, and 30% of randomly selected APKs without replacement. The *training* dataset was used for building the selected classifiers, the *testing* dataset was used for the multi-objective optimization



(a) API Calls View



(b) Opcode View



(c) Permissions View

Fig. 2: Classification performance of selected classifiers in a single-view operation on our dataset.

procedure. In contrast, the *validation* dataset was used to measure the final system accuracy. As a result, our built dataset comprises a comprehensive range of APK behaviors that can be used to validate the generalization of ML-based malware detection techniques.

B. Model Building

To evaluate our scheme, we selected three classifiers widely used for Android malware detection, namely Decision Tree (DT), Random Forest (RF), and k-Nearest Neighbor (kNN). The DT classifier was implemented with *gini* as a node quality measure without a maximum tree depth value. The RF was evaluated using 100 decision trees as its base learner, each using the same parameters used by the single DT. Finally, the kNN classifier used 5 neighbors as computed through the Euclidean distance. The used dataset was normalized through the *min-max* procedure for the kNN evaluation. The classifiers were implemented through *scikit-learn* API $v0.24$. To address the large number of features extracted for each view, we also apply Principal Component Analysis (PCA) with 100 components to conduct dimensionality reduction for ML training and testing purposes. The classifiers were evaluated according to their True Positive (TP) and True Negative (TN) rates. The TP denotes the ratio of *malware* samples correctly classified as *malware*, while the TN denotes the ratio of *goodware* samples correctly classified as *goodware*.

C. Multi-view Android Malware Detection

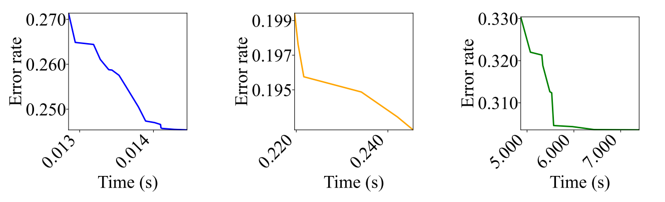
Our first experiment aims at answering *RQ1* and evaluates the accuracy performance of traditional ML-based Android malware detection techniques. To achieve such a goal, we evaluate the accuracy performance of the previously selected classifiers (see Section V-B) in a single-view setting. Table II shows the classification accuracy of the selected techniques when a single-view is considered. It is possible to note that the selected classifiers presented on average a low classification accuracy. For instance, the RF classifier presented TP rates of 83%, 84%, and 78% when making use of the *API Calls*, *Opcode*, and *Permissions* views respectively. As a result, ML-based Android malware detection through single-view classification fails at providing the necessary level of classification reliability for production deployment.

TABLE II: Accuracy performance of selected ML-based Android Malware detection schemes

View	**Classifier**	**Accuracy (%)**		
		TP	***TN***	***F1***
API Calls	DT	70.91	71.02	0.7095
	RF	83.17	77.18	0.8084
	kNN	67.59	66.75	0.6759
Opcode	DT	70.34	70.91	0.7045
	RF	84.32	76.57	0.8116
	kNN	67.91	65.63	0.6782
Permissions	DT	68.43	68.82	0.6849
	RF	78.45	75.14	0.7741
	kNN	70.09	62.58	0.6906
Ours	DT	75.86	75.35	0.7574
	RF	87.22	78.18	0.8324
	kNN	70.77	69.54	0.7061

Our second experiment aims at answering *RQ2* and evaluates the classification accuracy of our proposed model. To achieve such a goal, we first evaluate our proposed scheme making use of the multi-objective optimization scheme (see Section IV-B). We implement our scheme as a wrapper-based feature selection using the *Non-dominated Sorting Genetic Algorithm* (NSGA-III) [20] on top of *pymoo* API. In such a case, the NSGA-III uses a 100 population size, 100 generations, a crossover of 0.3, and a mutation probability of 0.1. The multi-objective feature selection aims at decreasing the obj_{proc} (Eq. 1) and obj_{error} (Eq. 2) as measured on the validation dataset. Notwithstanding, we evaluate our scheme without varying the underlying used classifier view (Fig. 1, *Classifier View 1* to *N*). In practice, our scheme always uses the DT, RF, or kNN, hence changing only the views used. This characteristic allows us to compare the performance of our proposal $vs.$ the traditional scheme evaluated previously.

Figure 3 shows the Pareto curve of our proposed model



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

Fig. 3: Pareto curve of our proposed model according to the used view.

for each selected classifier. It is possible to note a direct relationship between the measured processing inference costs $vs.$ the system's error rate. It is important to note that in practical deployment, the operator must choose the operation point that best fits its application. For our test, we selected the operation point closest to zero on both objectives.

Using the selected operation points, we first investigate the accuracy performance of our proposed model. Table II shows the classification accuracy of our scheme according to the used classifier. Our proposal significantly improves classification accuracy for all the evaluated classifiers. As an example, our scheme implemented using the DT classifier, can improve F1 by 0.05, 0.05, and 0.08 when compared to the single-view implemented with the *API Calls*, *Opcode*, and *Permissions* views respectively. On average, our scheme improved the TP rates by 4.5, and the TN rates by 3.9. Therefore, our scheme, implemented through a multi-objective optimization strategy, can enhance classification accuracy.

Finally, to answer *RQ3* 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 selected by our model (Table II). On average, our proposed model required only 65%, 78%, and 66% of the processing costs demanded by its all-features counterpart with the DT, kNN, and RF classifiers, respectively. Consequently, our proposed scheme can not only improve the overall system accuracy but also significantly decrease the associated inference processing costs.

VI. 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. As future work, we plan on extending our proposed model to make use of deep learning classifiers combined with multi-view classification and feature selection.

References

[1] *Android Statistics (2024)*. [Online]. Available: https://www.businessofapps.com/data/android-statistics/
[2] *Attacks on mobile devices significantly increase in 2023*. [Online]. Available: https://www.kaspersky.com/about/press-releases/2024_attacks-on-mobile-devices-significantly-increase-in-2023
[3] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Computing Surveys*, vol. 53, no. 6, p. 1–36, Dec. 2020.
[4] P. Bhat, S. Behal, and K. Dutta, "A system call-based android malware detection approach with homogeneous amp; heterogeneous ensemble machine learning," *Computers & Security*, vol. 130, p. 103277, Jul. 2023.
[5] A. Darwaish and F. Nait-Abdesselam, "Rgb-based android malware detection and classification using convolutional neural network," in *IEEE Global Communications Conference*, Dec. 2020.
[6] M. R. Smith, N. T. Johnson, J. B. Ingram, A. J. Carbajal, B. I. Haus, E. Domschot, R. Ramyaa, C. C. Lamb, S. J. Verzi, and W. P. Kegelmeyer, "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*, ser. CCS '20. ACM, Nov. 2020.
[7] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Towards a fair comparison and realistic evaluation framework of android malware detectors based on static analysis and machine learning," *Computers amp; Security*, vol. 124, p. 102996, Jan. 2023.
[8] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of android malware detection using static analysis," *IEEE Access*, vol. 8, p. 116363–116379, 2020.
[9] V. Ravi, M. Alazab, S. Selvaganapathy, and R. Chaganti, "A multi-view attention-based deep learning framework for malware detection in smart healthcare systems," *Computer Communications*, vol. 195, p. 73–81, Nov. 2022.
[10] D. O. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel permission-based android malware detection system using feature selection based on linear regression," *Neural Computing and Applications*, vol. 35, no. 7, p. 4903–4918, Mar. 2021.
[11] S. Seraj, S. Khodambashi, M. Pavlidis, and N. Polatidis, "Hamdroid: permission-based harmful android anti-malware detection using neural networks," *Neural Computing and Applications*, vol. 34, no. 18, p. 15165–15174, Jan. 2022.
[12] A. Pektaş and T. Acarman, "Learning to detect android malware via opcode sequences," *Neurocomputing*, vol. 396, p. 599–608, Jul. 2020. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2018.09.102
[13] S. Millar, N. McLaughlin, J. Martinez del Rincon, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *Journal of Information Security and Applications*, vol. 58, p. 102718, May 2021. [Online]. Available: http://dx.doi.org/10.1016/j.jisa.2020.102718
[14] A. Kyadige, E. M. Rudd, and K. Berlin, "Learning from context: A multi-view deep learning architecture for malware detection," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, May 2020.
[15] Y. Wu, M. Li, Q. Zeng, T. Yang, J. Wang, Z. Fang, and L. Cheng, "Droidrl: Feature selection for android malware detection with reinforcement learning," *Computers amp; Security*, vol. 128, p. 103126, 2023.
[16] M. A. Azad, F. Riaz, A. Aftab, S. K. J. Rizvi, J. Arshad, and H. F. Atlam, "Deepsel: A novel feature selection for early identification of malware in mobile applications," *Future Generation Computer Systems*, vol. 129, p. 54–63, 2022.
[17] K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Androzoo: Collecting millions of android apps for the research community," *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pp. 468–471, 2016.
[18] *Virustotal - Analyze suspicious files*. [Online]. Available: https://www.virustotal.com/
[19] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset," *Information Fusion*, vol. 52, p. 128–142, Dec. 2019. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2018.12.006
[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.