information

Article

A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection

Vasileios Kouliaridis [1,†] and Georgios Kambourakis [2,*,†]

[1] Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Samos, Greece; bkouliaridis@aegean.gr
[2] European Commission, Joint Research Centre, 21027 Ispra, Italy
* Correspondence: georgios.kampourakis@ec.europa.eu or gkamb@aegean.gr
† These authors contributed equally to this work.

Abstract: Year after year, mobile malware attacks grow in both sophistication and diffusion. As the open source Android platform continues to dominate the market, malware writers consider it as their preferred target. Almost strictly, state-of-the-art mobile malware detection solutions in the literature capitalize on machine learning to detect pieces of malware. Nevertheless, our findings clearly indicate that the majority of existing works utilize different metrics and models and employ diverse datasets and classification features stemming from disparate analysis techniques, i.e., static, dynamic, or hybrid. This complicates the cross-comparison of the various proposed detection schemes and may also raise doubts about the derived results. To address this problem, spanning a period of the last seven years, this work attempts to schematize the so far ML-powered malware detection approaches and techniques by organizing them under four axes, namely, the age of the selected dataset, the analysis type used, the employed ML techniques, and the chosen performance metrics. Moreover, based on these axes, we introduce a converging scheme which can guide future Android malware detection techniques and provide a solid baseline to machine learning practices in this field.

Keywords: android malware; mobile malware detection; machine learning and classification

Citation: Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. *Information* **2021**, *12*, 185. https://doi.org/10.3390/info12050185

Academic Editor: Christoforos Ntantogian

Received: 1 April 2021
Accepted: 23 April 2021
Published: 25 April 2021

1. Introduction

According to a recent report from McAfee [1], 2020 was the year of mobile sneak attacks. Namely, cybercriminals and state-sponsored actors are persistently seeking ingenious ways to acquire user data. Likewise, malware writers continue to come up with new ways of hiding their attacks and frauds, making it increasingly difficult to identify and neutralize. On the positive side, new mobile malware detection techniques are also evolving to counter these threats. Indeed, machine learning (ML) has long proved its decisive role in this ecosystem given that the vast majority of mobile malware detection solutions proposed in the literature so far are based on some kind of ML-driven scheme.

Traditionally, ML is exploited in anomaly-based detection methods. Anomaly-based detection comprises two basic phases, namely the training and the detection or testing one. That is, such solutions employ ML to detect malicious behavior, i.e., deviation from a model built during the training phase. Anomaly-based detection can be further categorized depending on the type of analysis, i.e., static, dynamic, and hybrid. Static analysis is performed in a non-runtime environment, which analyzes an app's internal structure. Dynamic analysis, on the other hand, adopts the opposite approach, taking place during the app's normal operation.

As shown in Table 1, assorted app features can be extracted depending on the analysis type, either static, dynamic, or hybrid. Each of these features has its advantages and limitations. Namely, features stemming from the static analysis have proven efficient against older malware apps [2], but tend to be ineffective against code obfuscation and encryption techniques [3].

Table 1. Feature extraction options per analysis method.

Analysis Type	Feature Extraction Method	Features Extracted
Static	Manifest analysis	Package name, Permissions, Intents, Activities, Services, Providers
	Code analysis	API calls, Information flow, Taint tracking, Opcodes, Native code, Cleartext analysis
Dynamic	Network traffic analysis	URLs, IPs, Network Protocols, Certificates, Non-encrypted data
	Code instrumentation	Java classes, intents, network traffic
	System calls analysis	System calls
	System resources analysis	CPU, Memory, and Battery usage, Process reports, Network usage
	User interaction analysis	Buttons, Icons, Actions/Events

When feeding additional features extracted through dynamic analysis to malware detection models, they can typically cope significantly better with the newest and more challenging pieces of malware [4]. However, hybrid analysis systems are inherently more complex, due to the several extra components mandated by dynamic analysis, such as a virtual or real platform, and a user event and input emulator to exercise the app. On top of that, some sophisticated malicious apps can recognize when being executed in emulated environments and avoid detection [5].

Classification is the process of categorizing data into classes. This process starts with predicting the class of given data points. The classes are often referred to as target, label, or categories. From an ML model's perspective, classification requires a training dataset with multiple instances from which the chosen ML model learns. Much like app analysis methods, each ML model also has its pros and cons based on the supplied data [6]. As detailed in Section 3, the majority of mobile malware detection works in the literature demonstrate a different ML algorithm as the best performer for mobile malware detection. For this reason, several performance optimization techniques have been used throughout the literature to further enhance classification performance. These techniques include:

- Feature ranking and selection by calculating feature importance scores.
- Dimensionality reduction transforms features into a lower dimension to reduce bias and noise.
- Ensemble models combine the output of multiple base models to improve the overall classification performance and can be used in conjunction with any of the previous two techniques.

Given the growing impact of ML-aided mobile malware detection schemes, a deeper literature review is needed, considering all state-of-the-art works available and exploring the details behind each efficient detection model. Unfortunately, while there are many contributions in the literature leveraging ML for mobile malware detection on the Android platform, most of them rely on diverse metrics, classification models, and performance improvement techniques. The absence of a common baseline in this field can cause confusion, lead to half-true or even incorrect generalizations, and even mislead future research. In an effort to mitigate these issues, the work at hand contributes to the following goals:

- Provides a detailed mapping of the contemporary ML techniques regarding Android malware detection proposed in the literature during the last 7 years, namely from 2017 to 2021.
- Categorizes each contribution based on four distinct criteria, i.e., the chosen metrics, dataset age, classification models, and performance improvement techniques.
- Introduces a converging, i.e., decision-making scheme to guide future work in this ecosystem.

The remainder of this paper is organized in the following manner. The next section discusses similar surveys in the literature. Section 3 details ML-driven Android malware detection schemes proposed in the literature so far and categorizes each work based on a quartet of criteria. Section 4 provides a discussion on the main findings and introduces the proposed scheme. Section 5 draws a conclusion.

2. Relevant Surveys

As of today, the topic of Android malware detection has received plenty of attention in the literature. However, few works focus on the ML methodologies employed and, to the best of our knowledge, none of them provides a clear classification of mobile malware detection systems based on the metrics and ML techniques used. Focusing on a period spanning from 2017 to 2021, this section chronologically identifies such literature contributions and places them vis-à-vis the current work.

Yan et al. [7] offered a thorough survey on dynamic mobile malware detection approaches, summarizing a number of criteria and performance evaluation metrics for mobile malware detection. Additionally, the authors analyzed and compared the until then existing mobile malware detection systems based on the analysis methods and evaluation results. Finally, the authors pointed out open issues in the field and future research directions.

Odusami et al. [8] surveyed mobile malware detection techniques in an effort to identify gaps and provide insight for effective measures against unknown Android malware. Their work showed that approaches which rely on ML to detect malicious apps were more promising and produced higher detection accuracy as opposed to signature-based techniques.

Kouliaridis et al. [9] provided a holistic review of works on the topic of mobile malware detection and categorized each of them under a unique classification scheme. Precisely, the latter groups each work based on its target platform, feature selection method, and detection techniques, namely signature-based or anomaly-based detection.

Liu et al. [10] presented a comprehensive survey of Android malware detection approaches that utilize ML techniques. The authors analyzed and summarized several key topics, including sample acquisition, data preprocessing, feature selection, ML models, algorithms, and detection performance. Finally, they elaborated on the limitations of ML approaches and offered insights for potential future directions.

Gibert et al. [11] surveyed popular ML techniques for malware detection and in particular, deep learning techniques. The authors explained research challenges and limitations of legacy ML techniques and scrutinized recent trends and developments in the field with a focus on deep learning schemes. They categorized the surveyed works into three groups, namely static, dynamic, and hybrid.

As shown in Table 2, none of the above works offers a complete classification of each approach based on the features listed in Section 1, namely, metrics, classification models, dataset, and performance improvement techniques. Furthermore, none of them concentrate on performance improvement techniques for ML-based detection systems. The current work aspires to fill this gap and additionally introduce an overarching, decision-making scheme that will potentially guide future ML-based methodologies on deciding which analysis method, ML performance optimization technique, and metric is most fitting based on the employed dataset.

Table 2. Important topics addressed by the related works. PEM: performance evaluation metrics, DT: detection techniques, ML: machine learning, AM: analysis methods, FE: features and feature extraction, DL: deep learning, ML PI: ML performance improvement.

Work	Year	Performance	PEM	DT	ML	AM	FE	DL	Datasets	ML PI
[7]	2017	+	+	-	-	-	-	-	-	-
[8]	2018	-	+	+	+	-	-	-	-	-
[9]	2020	+	-	+	-	+	+	-	-	-
[10]	2020	+	-	-	+	-	+	-	+	-
[11]	2020	-	-	-	+	-	-	+	-	-
Current	2021	+	+	+	+	+	+	-	+	+

3. Literature Survey

This section provides a detailed review of major published works devoted to the detection of Android malware in the last seven years. Table 3 categorizes each work in chronological order based on the following four criteria, while Table 4 offers a condensed view of the common criteria across all contributions.

- The analysis type, namely static, dynamic, or hybrid.
- The feature extraction method, namely Manifest Analysis (MA), source Code Analysis (CA), Network Traffic Analysis (NTA), Code Instrumentation (CI), System Calls Analysis (SCA), System Resources Analysis (SRA), and User Interaction Analysis (UIA).
- The features collected, as it has been listed in Table 1.
- The classification approach, i.e., base models and possible performance improvement techniques, including Feature importance (FI) metrics, Dimensionality Reduction (DR), and Ensemble Learning (EL).

Shabtai et al. [12] contributed a system that detects malicious behavior through network traffic analysis. This is done by logging user-specific network traffic patterns per examined app and subsequently identifying deviations that can be flagged as malicious. To evaluate their model, they employed the C4.5 algorithm, achieving an accuracy of up to 94%.

Canfora et al. [13] suggested an Android malware detection scheme that analyzes op-code frequency histograms; this is accomplished by observing the frequency of occurrences of each group of op-codes. Precisely, their detection model capitalizes on a vector of features obtained from eight Dalvik op-codes. These op-codes are usually used to alter the app's control flow. Six classification models were employed during the evaluation, namely LadTree, NBTree, RandomForest, RandomTree and RepTree. The proposed model were applied separately to the eight features and the three groups of features. The first group includes the move and the jump features, the second involves two well-known distance metrics, namely Manhattan and Euclidean distance, and the last embraces all the four features. The proposed method was evaluated on the Drebin dataset using several classifiers, namely J48, LadTree, NBTree, Random Forest, Random Tree and RepTree, and achieved an accuracy of 95%.

Jang et al. [14] developed Andro-AutoPsy, an anti-malware system based on Android malware similarity matching. To train the proposed model, the authors gathered malware information from antivirus mobile threat reports, malware repositories, and community sites via web crawling. They chose five footprints as features, namely the serial number of a digital certificate, API call sequence, permissions, intents, and system commands. According to the authors, Andro-AutoPsy can detect zero-day malware. Andro-AutoPsy was evaluated with nearly 1K malware apps obtained from the VirusShare [15] and Contagio mobile datasets, [16] and more than 109K benign samples collected from Google Play [17].

Yerima et al. [18] proposed an ensemble classification model capitalizing on critical Android and Java API calls stemming from the source code, as well as on app permissions extracted from the manifest file. In all the experiments, McAfee's internal (private) dataset

was utilized. During the evaluation phase, several classifiers were employed, namely Naive Bayes, Simple Logistic, Decision Tree and Random Tree, scoring an AUC of up to 99.3% and accuracy of 97.5%.

Coronado-De-Alba et al. [19] presented a mobile malware detection approach based on a meta-ensemble algorithm. They conducted static analysis on a corpus of 1531 malware apps collected from the Drebin dataset [20] and 765 benign apps, to get permissions and intents. The authors employed the RandomForest and RandomCommittee algorithms, achieving an accuracy of up to 97.5% with the use of the former.

Milosevic et al. [21] put forward a detection method that concentrates on the extraction of non-trivial and beneficial to detection malicious patterns. This has been achieved by examining the usefulness of source code in terms of detection, as well as the permissions set of features when combined with either classification or commonly used clustering techniques, respectively. In their experiments, the M0Droid corpus [22] was considered. Several classifiers were used during the evaluation process, such as the C4.5, Random forest, Naive Bayes, Support Vector Machine (SVM), JRip, and Logistic Regression. Their results showed an accuracy of up to 95.6%.

By using a static app analysis approach, Idrees et al. [23] contributed an Android malware detection method based on ensemble learning to augment the performance of base classification models. They collected Android apps from a variety of corpora, including Contagio dump, MalGenome [24], theZoo [25], Malshare [26], and Virushare [15]. For each app, they extracted permissions and intents. The features with the highest feature importance score, calculated using the information gain (IG) algorithm, were selected to train the model. The authors employed the Naive Bayes, Decision Tree, Decision Table, Random Forest, and Multilayer perceptron (MLP) classifiers. Their evaluation tests demonstrated a best AUC and accuracy score of up to 99.8%.

DroidNative has been introduced by Alam et al. [27] as a solution for detecting Android malware. This scheme considers both bytecode and native code analysis, and according to the authors, DroidNative is the first to build cross-platform (x86 and ARM) semantic-based signatures for Android. Specifically, during app analysis, bytecode is passed to an Android Runtime (ART) [28] compiler to generate native binary code. This code is then disassembled and translated into Malware Analysis Intermediate Language (MAIL) code. To evaluate their approach, the authors collected over 5490 malware from the Drebin and Contagio mobile corpora. Their results showed a detection rate of up to 93.57% and an AUC score ranging from 97.86% to 99.56%.

Kouliaridis et al. [29] proposed Mal-warehouse, an open-source tool performing data collection-as-a-service for Android malware behavioral patterns. To this end, they developed an open-source tool called "MIET" to extract statistical data in terms of memory, CPU, battery, and network utilization over a period of time from Android devices when running an app. Mal-warehouse is also equipped with a detection module, which the authors evaluated via the use of a series of base classifiers, namely k-NN, Random Forest, SVM, Naive Bayes and AdaBoost, and achieved a top AUC score of 85.4%.

Tao et al. [30] introduced MalPat, an automated malware detection system that scans for malicious patterns in Android apps. Specifically, MalPat detects malicious patterns by analyzing API calls. The authors collected 31,195 benign apps and 15,336 malware samples. A repeated process was followed to evaluate MalPat using the Random Forest model, in which they randomly selected a percentage of both malicious and benign datasets as the training set, and the remaining part was taken as the testing set. In their evaluations, MalPat achieved a 98.24% F1 score using the SVM classifier.

A mobile malware detection scheme that relies on information flow analysis has been suggested by Shen et al. [31]. They introduced the notion of complex-flow as a new representation approach for information flows. According to them, complex-flow is a set of simple flows that share a common portion of code. For example, "if an app is able to read contacts, store them and then send them over the Internet, then these two flows would be (contact, storage) and (contact, network)". According to the authors, their approach can

detect malicious information flows based on the app's behavior along with the flow. That is, for each examined app, their approach compares the app's behavior motif, obtained from the complex-flows representation of the app, to decide whether it is malicious or not. This is done by means of two-class SVM classification. Precisely, to test the performance of their method, the authors used four different datasets, totaling 8598 apps. Their model achieved the best accuracy of 94.5%.

Wang et al. [32] proposed a malware detection method that employs network traffic analysis and uses the c4.5 algorithm. According to the authors, c4.5 is capable of identifying Android malware with very high accuracy. The authors tested their model on the Drebin dataset [20]. The obtained results showed that the proposed model performs well when compared with state-of-the-art approaches and attains a detection rate of up to 97.89% with the aforementioned algorithm.

Kouliaridis et al. [4] proposed a plain heterogeneous ensemble malware detection method. The ensemble model is created by averaging the output of several base models based on either static or hybrid analysis. The features extracted pertain to permissions, intents and API calls, Java classes, network traffic, and inter-process communications. The performance of this method is evaluated against several corpora, namely Drebin [20], VirusShare [15], and AndroZoo [33]. The authors evaluated their model using more than a handful of classifiers, namely Logistic Regression, Naive Bayes, Random Forest, k-NN, AdaBoost, Stochastic Gradient Descent (SGD), and SVM. Additionally, the authors used the most challenging dataset, i.e., AndroZoo, and achieved accuracy and AUC score of 97.8% and 97.7%, respectively. Finally, feature importance was calculated for each dataset and feature.

Potha et al. [34] examined the effect of an ensemble model when external instances of different sizes and types are used. This novel ensemble model works by combining the output of several base models, namely Logistic Regression, MLP, and SGD. Their results demonstrated that ensemble models based on a larger and possibly homogeneous size of external instances are exceptionally effective alternatives to ensemble models which comprise smaller sizes, and feasibly more heterogeneous external instances. Additionally, they examined the effect of using either the entire feature set or a random subspace of features of instances, and showcased that the latter aids an extrinsic ensemble model to further augment its performance. The authors reported 99.4%, 99.3%, and 99.7% AUC and 98.3%, 98.7%, and 99.1% accuracy on the AndroZoo, VirusShare, and Drebin datasets, respectively.

DL-Droid, a deep learning system that detects malicious Android apps with dynamic analysis using stateful input generation, has been proposed by Alzaylaee et al. [35]. They collected more than 31,000 apps of which more than 11,000 being malware. DL-Droid runs on an automated platform, which is able to perform both static and dynamic analysis. The evaluation was carried out using a real Android device, and the reported required time for analyzing each app was approximately 190 sec. Using the Random Forest classifier, DL-Droid achieved a detection rate of up to 97.8% when only considering features stemming from dynamic analysis and 99.6% when also adding features derived from static analysis.

Taheri et al. [36] developed four malware detection methods based on Hamming distance. Their models aim to discern similarities between samples which are first nearest neighbors (FNN), all nearest neighbors (ANN), weighted all nearest neighbors (WANN), and k-medoid based nearest neighbors (KMNN). The authors extracted permissions, intents and API Calls from three datasets, namely Drebin [20], Contagio mobile [16], and MalGenome [24]. Using a Random Forest Regressor feature selection algorithm, the authors selected 300 important features. The evaluation was carried out using several classifiers, namely SVM, Decision Tree, Random Forest, and MLP, and achieved an accuracy between 90% and 99%.

Millar et al. [37] presented DANdroid, a mobile malware detection model which uses deep learning to classify apps. DANdroid capitalizes on a triad of features, namely Opcodes, permissions, and API calls. Their model was evaluated with apps from the

Drebin dataset, obfuscated with five techniques, which in turn produced a total of nearly 70K apps. Their results demonstrated a F-score of up to 97.3% using the CNN algorithm.

Cai et al. [38] proposed JOWMDroid, an Android malware detection scheme based on feature weighting, with the joint optimization of weight-mapping and classifier parameters. Eight feature categories were extracted from Android apps, and then the most important features were selected using the IG algorithm. The proposed model calculates weights per feature with three base models, and then five weight-mapping models are designed to map the initial weights to the final ones. Finally, the parameters of the weight-mapping model and the base model are jointly optimized by the differential evolution algorithm. The authors collected malware from two datasets, namely Drebin and AMD. They used several classifiers to assess their approach, i.e., SVM, Random Forest, and Logistic Regression, scoring the best accuracy of 98.1%.

The effect of two well-known dimensionality reduction techniques, namely PCA and t-SNE, when applied on base models as well as ensembles has been examined by Kouliaridis et al. [2]. It was demonstrated that both these transformations are able to considerably increase the performance of each base model as well as the constructed ensembles. Static analysis was employed to extract permissions and intents from 1000 apps in the AndroZoo dataset. The authors evaluated their model using several classifiers, namely AdaBoost, k-NN, Logistic Regression, Naive Bayes, MLP, SGD, Random Forest and SVM, and achieved a 95.1% and 91.7% AUC and accuracy scores, respectively.

Table 3. Outline of the surveyed works.

Work	Year	Analysis	Method(s)	Feature(s)	Dataset(s)	ML Technique(s)
[12]	2014	Dynamic	NTA	Network traffic	N/A	Base models
[13]	2015	Static	CA	Opcodes	Drebin	Base models, DR
[14]	2015	Static	MA, CA	Package name, Permissions, API calls, Intents, Opcodes	Contagio Mobile, VirusShare	Base models
[18]	2015	Static	CA	Permissions, API Calls	McAfee	EL
[19]	2016	Static	CA	Permissions, Intents	Drebin	EL
[21]	2017	Static	CA	Permissions, Source code	M0Droid	EL
[23]	2017	Static	CA	Permissions, Intents	Contagio, MalGenome, theZoo, Malshare, VirusShare	FI, EL
[27]	2017	Static	CA	Native code	Contagio Mobile, Drebin	Base models
[29]	2018	Dynamic	SRA	CPU, Memory, and Battery usage, Process reports, Network usage	N/A	Base models
[30]	2018	Static	CA	API calls	N/A	Base models
[31]	2018	Static	CA	Information flow	N/A	Base models
[32]	2019	Dynamic	NTA	Network traffic	Drebin	Base models
[4]	2020	Hybrid	MA, CA, CI	Permissions, Intents, API calls, Java classes, inter-process communication, network traffic	Drebin, VirusShare, AndroZoo	Base models, FI, EL
[34]	2020	Static	MA	Permissions, Intents	Drebin, VirusShare, AndroZoo	Base models, EL
[35]	2020	Hybrid	MA, CA, UIA	Permissions, Intents, API Calls, Actions/Events	McAfee	Base models, FI
[36]	2020	Static	MA, CA	Permissions, Intents, API Calls	Drebin, Contagio mobile, MalGenome	Base models, FI
[37]	2020	Static	MA, CA	Permissions, Opcodes, API Calls	Drebin	Base models
[38]	2021	Static	MA, CA	Permissions, Intents, Features, Components, API Calls, Intents, Shell commands	Drebin, AMD	Base models plus weighted-mapping, FI
[2]	2021	Static	MA	Permissions, Intents	AndroZoo	Base models, EL, DR

Table 4. Summary of key characteristics observed across the surveyed works.

Category	Option	No. of Works
Analysis type	Static	14
	Dynamic	3
	Hybrid	2
Feature extraction method	Source Code analysis	14
	Manifest analysis	8
	Network traffic analysis	2
	Code instrumentation	1
	System resources analysis	1
	User interaction analysis	1
Dataset age	2010 to 2014	11
	2015 to 2016	5
	2017 to 2020	3
ML techniques	Base models	15
	Ensemble learning	7
	Feature importance	5
	Dimensionality reduction	2
Metrics	Accuracy as a metric	13
	AUC as a metric	7
	Other metric	4

4. Discussion

This section wraps up a number of key findings based on the surveyed works in Section 3. Precisely, as shown in Tables 3 and 4, most contributions, i.e., 14 out of 19, rely on static analysis alone, while only 3 and 2 take advantage of dynamic and hybrid analysis, respectively. Additionally, the following important observations can be made about performance optimization techniques:

- Ensemble models are considered by 7 works, i.e., roughly the one-third.
- Feature importance scores are calculated in 5 works, i.e., about the one-fourth.
- Dimensionality reduction techniques are used in only 2 works.

Ensemble models have started to appear in the relevant literature after 2015. Specifically, such models are used in 6 out of 11 works employing static analysis from 2015 to 2020, and in all of the works in the same year span, which make use of a regularly updated malware dataset, namely VirusShare or AndroZoo.

As shown in Table 4, source code analysis is the most common analysis technique exploited in the surveyed literature. Moreover, the most widespread classification features among the surveyed works are Permissions, Intents, and API Calls, used in 12, 9, and 7 works, respectively.

As shown in Table 4, when focusing on the datasets employed, it is deduced that numerous works rely on outdated (and no more updated) datasets. Specifically, Drebin, dated back to 2012, is the most used dataset, utilized in almost half of the surveyed works. On the other hand, Contagio Mobile and MalGenome are used in 4 and 2 works, respectively. However, the former is dated back to 2010, while the latter to 2012. Recall that previous work has shown that feature importance changes across datasets of different age [6,39].

Additionally, previous work has demonstrated that when extracting multiple feature categories from a large collection of apps, the number of features substantially increases [4], as does the computational cost and risk of overfitting due to the resulting model complexity. Therefore, when evaluating a mobile malware detection approach, the choice of the dataset should play a key role in choosing the classification models and performance-enhancing techniques, such as ensemble learning. On the positive side, works dated after

2015 seem to also employ newer datasets, such as VirusShare and AndroZoo, which are regularly updated.

Another important factor when assessing an ML-based approach is the primary metric used to evaluate its classification performance. The top used metrics shown in Table 4 reveal that the Accuracy and AUC are the most commonly used. However, by inspecting the contributions included in Section 3, one can easily conclude that a wide variety of metrics has been utilized to measure classification performance, namely detection rate (DR), true positive rate (TPR), Precision, Recall, F1, Accuracy, and Area Under the Curve (AUC). This assortment can cause a series of issues, including (a) inability to compare with state-of-the-art when the same metric is not available, and (b) incorrect metrics can produce inaccurate or over-estimated results, as in the case when using the accuracy metric with imbalanced datasets [4].

Finally, Figure 1 illustrates the most popular base classification models among the surveyed works. The Random forest seems to be the most popular classifier used in 11 works, followed by SVM and Naive Bayes used in 8 and 6 works, respectively.

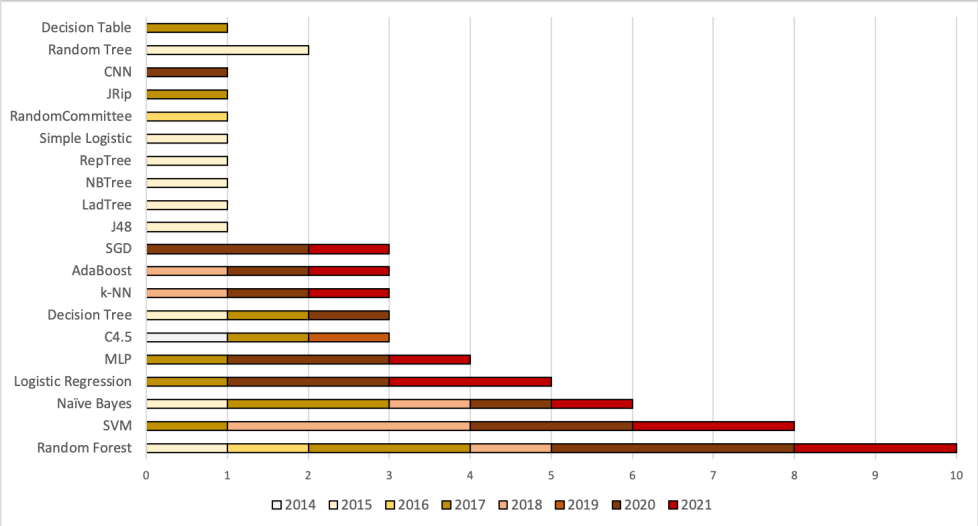

Figure 1. Number of works utilizing each base classification model per year.

In an effort to address the aforementioned issues, we introduce an overarching, converging parameter selection scheme shown in Figure 2. Precisely, the proposed scheme aims to aid future mobile malware detection methodologies, by suggesting a unified baseline for designing more comparable and well-engineered ML-based malware detection solutions. This is achieved by considering all four key parameters into a unified typology. Simply put, the "parameters" term here refers to feature importance across corpora of different age, the increase in performance when using ensemble models instead of base models, the merit of dimensionality reduction techniques in mobile malware detection, and the advantages of each of the classification metrics. Under this mindset, Figure 2 comprises four steps, namely dataset age selection, analysis method selection, ML techniques selection, and performance metrics selection.

Specifically, the proposed scheme guides one in picking optimal ML techniques based on the dataset age and analysis method chosen in the first and second step, respectively. Namely, the first two steps associate the age of the dataset used for evaluation with one of the three analysis methods. The third step indicates the ML classification techniques to be used based on the choice made during the preceding steps. The final step depends on whether the dataset used is balanced in terms of malware and benign apps. This will determine if accuracy is indeed a trustworthy metric. In all cases, however, the AUC metric is preferable, as it constitutes a more conclusive and realistic evaluation of models, even when substantially imbalanced datasets are utilized [40]. Generally, AUC quantifies the effectiveness of each examined approach for all possible score thresholds. As a rule, the

value of AUC is extracted by examining the ranking of scores rather than their exact values produced when a method is applied to a dataset. On top of everything else, AUC does not depend on the equality of distribution between positive and negative classes.

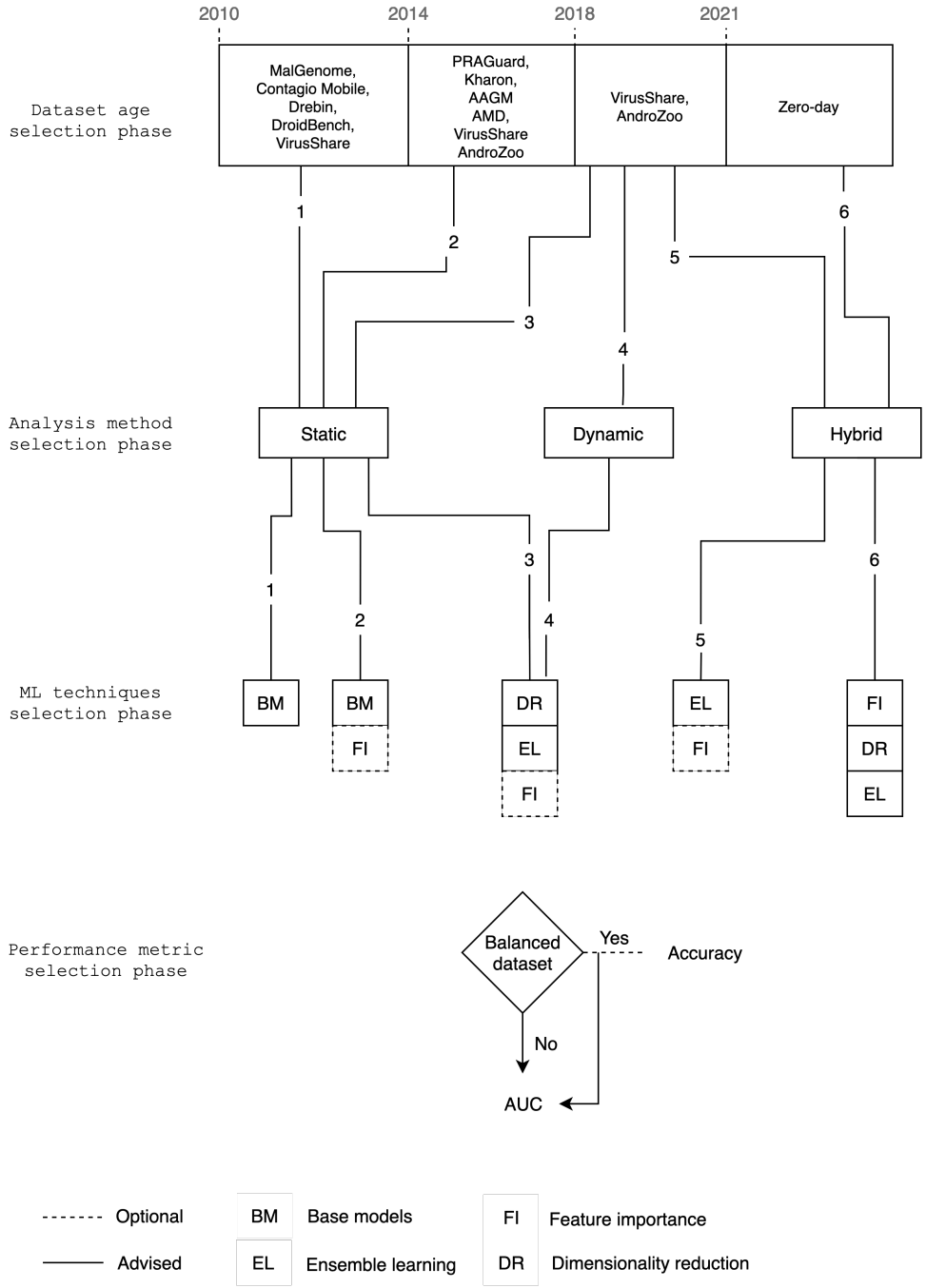

Figure 2. Baseline scheme for mobile malware detection models.

5. Conclusions

This work provides a state-of-the-art survey on ML-powered Android malware detection techniques. To do so, we categorize and succinctly analyze state-of-the-art works in the literature during the last seven years, i.e., from 2014 to 2021, based on the analysis type, feature extraction method, dataset, ML classification techniques, and metrics used in their performance evaluation. Additionally, we elaborate on our findings and research trends, as well as possible issues and future directions. From the results, it becomes obvious that the majority of the approaches embrace a different set of basic parameters, including

the dataset, the analysis (feature collection), and the detection evaluation metrics. To moderate this issue, we proposed a four-step converging scheme to serve as a baseline and springboard for future mobile ML-based Android malware detection approaches. Future work could consider providing an initial evaluation of the proposed converging scheme by juxtaposing findings stemming from its different trajectories in terms of dataset age, analysis method, and ML techniques selected.

Author Contributions: Conceptualization, V.K. and G.K.; writing—original draft preparation, V.K. and G.K.; writing—review and editing, V.K. and G.K.; supervision, G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mobile Threat Report 2020. Available online: https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf (accessed on 10 March 2021).
2. Kouliaridis, V.; Potha, N.; Kambourakis, G. Improving Android Malware Detection Through Dimensionality Reduction Techniques. In *Machine Learning for Networking*; Springer International Publishing: Paris, France, 2021; pp. 57–72. [CrossRef]
3. Bacci, A.; Bartoli, A.; Martinelli, F.; Medvet, E.; Mercaldo, F.; Visaggio, C. *Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis*; Funchal: Madeira, Portugal, 2018; pp. 379–385. [CrossRef]
4. Kouliaridis, V.; Kambourakis, G.; Geneiatakis, D.; Potha, N. Two Anatomists Are Better than One—Dual-Level Android Malware Detection. *Symmetry* **2020**, *12*, 1128. [CrossRef]
5. Petsas, T.; Voyatzis, G.; Athanasopoulos, E.; Polychronakis, M.; Ioannidis, S. Rage against the virtual machine. In Proceedings of the Seventh European Workshop on System Security—EuroSec '14, Amsterdam, The Netherlands, 13 April 2014; ACM Press: New York, NY, USA, 2014. [CrossRef]
6. Roy, S.; DeLoach, J.; Li, Y.; Herndon, N.; Caragea, D.; Ou, X.; Ranganath, V.; Li, H.; Guevara, N. Experimental Study with Real-World Data for Android App Security Analysis Using Machine Learning. In Proceedings of the 31st Annual Computer Security Applications Conference, ACSAC 2015, Los Angeles, CA, USA, 7–11 December 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 81–90. [CrossRef]
7. Yan, P.; Yan, Z. A survey on dynamic mobile malware detection. *Softw. Qual. J.* **2017**, *26*, 891–919. [CrossRef]
8. Odusami, M.; Abayomi-Alli, O.; Misra, S.; Shobayo, O.; Damasevicius, R.; Maskeliunas, R. Android Malware Detection: A Survey. In *Communications in Computer and Information Science*; Springer International Publishing: New York, NY, USA, 2018; pp. 255–266. [CrossRef]
9. Kouliaridis, V.; Barmpatsalou, K.; Kambourakis, G.; Chen, S. A Survey on Mobile Malware Detection Techniques. *IEICE Trans. Inf. aSyst.* **2020**, *E103.D*, 204–211. [CrossRef]
10. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
11. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
12. Shabtai, A.; Tenenboim-Chekina, L.; Mimran, D.; Rokach, L.; Shapira, B.; Elovici, Y. Mobile malware detection through analysis of deviations in application network behavior. *Comput. Secur.* **2014**, *43*, 1–18. [CrossRef]
13. Canfora, G.; Mercaldo, F.; Visaggio, C.A. Mobile Malware Detection using Op-code Frequency Histograms. In Proceedings of the 12th International Conference on Security and Cryptography, SCITEPRESS—Science and and Technology Publications, Colmar, France, 20–22 July 2015. [CrossRef]
14. Jang, J.; Kang, H.; Woo, J.; Mohaisen, A.; Kim, H.K. Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digit. Investig.* **2015**, *14*, 17–35. [CrossRef]
15. Virusshare. Available online: https://virusshare.com/ (accessed on 10 September 2020).
16. Contagio. Available online: http://contagiominidump.blogspot.com/ (accessed on 10 September 2020).
17. Google Play. Available online: https://play.google.com/ (accessed on 10 September 2020).
18. Yerima, S.Y.; Sezer, S.; Muttik, I. High accuracy android malware detection using ensemble learning. *IET Inf. Secur.* **2015**, *9*, 313–320. [CrossRef]
19. Coronado-De-Alba, L.D.; Rodríguez-Mota, A.; Escamilla-Ambrosio, P.J. Feature selection and ensemble of classifiers for Android malware detection. In Proceedings of the 2016 8th IEEE Latin-American Conference on Communications (LATINCOM), Medellin, Colombia, 15–17 November 2016; pp. 1–6.

20. Arp, D.; Spreitzenbarth, M.; Huebner, M.; Gascon, H.; Rieck, K. Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2014; Volume 12, p. 1128.
21. Milosevic, N.; Dehghantanha, A.; Choo, K.K.R. Machine learning aided Android malware classification. *Comput. Electr. Eng.* **2017**, *61*, 266–274. [CrossRef]
22. Damshenas, M.; Dehghantanha, A.; Choo, K.K.; Mahmud, R. M0Droid: An Android Behavioral-Based Malware Detection Model. *J. Inf. Priv. Secur.* **2015**, *11*, 141–157. [CrossRef]
23. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [CrossRef]
24. Android Malware Genome Project. Available online: http://www.malgenomeproject.org/ (accessed on 11 August 2020).
25. The Zoo Aka Malware DB. Available online: https://thezoo.morirt.com/ (accessed on 11 August 2020).
26. MalShare Project. Available online: https://malshare.com/about.php (accessed on 11 August 2020).
27. Alam, S.; Qu, Z.; Riley, R.; Chen, Y.; Rastogi, V. DroidNative: Automating and optimizing detection of Android native code malware variants. *Comput. Secur.* **2017**, *65*, 230–246. [CrossRef]
28. Android Runtime (ART) and Dalvik. Available online: https://source.android.com/devices/tech/dalvik (accessed on 10 September 2020).
29. Kouliaridis, V.; Barmpatsalou, K.; Kambourakis, G.; Wang, G. Mal-Warehouse: A Data Collection-as-a-Service of Mobile Malware Behavioral Patterns. In Proceedings of the 2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China, 8–12 October 2018; pp. 1503–1508. [CrossRef]
30. Tao, G.; Zheng, Z.; Guo, Z.; Lyu, M.R. MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. *IEEE Trans. Reliab.* **2018**, *67*, 355–369. [CrossRef]
31. Shen, F.; Vecchio, J.D.; Mohaisen, A.; Ko, S.Y.; Ziarek, L. Android Malware Detection Using Complex-Flows. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2430–2437. [CrossRef]
32. Wang, S.; Chen, Z.; Yan, Q.; Yang, B.; Peng, L.; Jia, Z. A mobile malware detection method using behavior features in network traffic. *J. Netw. Comput. Appl.* **2019**, *133*, 15–25. [CrossRef]
33. Allix, K.; Bissyandé, T.F.; Klein, J.; Traon, Y.L. AndroZoo: Collecting Millions of Android Apps for the Research Community. In Proceedings of the 13th International Conference on Mining Software Repositories, Austin, TX, USA, 14–15 May 2016; ACM: New York, NY, USA, 2016; pp. 468–471.
34. Potha, N.; Kouliaridis, V.; Kambourakis, G. An extrinsic random-based ensemble approach for android malware detection. *Connect. Sci.* **2020**, 1–17. [CrossRef]
35. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [CrossRef]
36. Taheri, R.; Ghahramani, M.; Javidan, R.; Shojafar, M.; Pooranian, Z.; Conti, M. Similarity-based Android malware detection using Hamming distance of static binary features. *Future Gener. Comput. Syst.* **2020**, *105*, 230–247. [CrossRef]
37. Millar, S.; McLaughlin, N.; del Rincon, J.M.; Miller, P.; Zhao, Z. *DANdroid: A Multi-View Discriminative Adversarial Network for Obfuscated Android Malware Detection*; Association for Computing Machinery: New York, NY, USA, 2020. [CrossRef]
38. Cai, L.; Li, Y.; Xiong, Z. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Comput. Secur.* **2021**, *100*, 102086. [CrossRef]
39. Kouliaridis, V.; Kambourakis, G.; Peng, T. Feature Importance in Android Malware Detection. In Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 29 December 2020–1 January 2021; pp. 1449–1454. [CrossRef]
40. Fawcett, T. An introduction to ROC analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. [CrossRef]