The Illusion of Success: Learning-Based Android Malware Detectors (Replicability Study)

MICHAEL TEGEGN✉, The University of British Columbia, Canada
JULIA RUBIN, The University of British Columbia, Canada

Since 2012, hundreds of machine-learning-based classification approaches have been proposed to help separate malware from benign Android applications. These approaches typically collect a large number of applications of both types, split them into training and testing subsets, train a binary classifier on the training subset, and measure accuracy on the testing subset. They typically report very high achieved accuracy, i.e., F1-score of around 95%. Recent work has also highlighted several biases and flaws in the experimental setup and evaluation methodology of such approaches, questioning the trustworthiness of their reported results.

In an effort to better understand the current status of classification-based malware detection, we first conduct a systematic literature review to extract the properties of existing tools and the datasets that they use. We then design a large-scale longitudinal study, where we evaluate the most prominent tools on a range of datasets spanning 13 years (2011-2023), which we systematically collected from the AndroZoo and VirusShare repositories while controlling for the known experimental setup biases.

Our results show lower than reported classification performance, with the F1-score for a tool ranging from high 60 to low 90 percent, depending on a dataset. Moreover, even successful classification often utilizes hidden but semantically weak correlations in the data. In fact, a deliberately naïve and unreliable classifier we designed for this study, which uses application package names as features for classification, performs comparably to and sometimes even better than the state-of-the-art tools when compared under the same setup. These results challenge the claimed detection capabilities of the tools, render comparisons of reported tool accuracy (without empirical evaluation on the exact same set of applications) practically meaningless, and call for the creation of more reliable and explainable semantic malware detection tools.

CCS Concepts: • **Security and privacy → Software security engineering**.

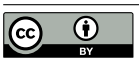
Additional Key Words and Phrases: Android malware detection, machine learning, empirical evaluation

ACM Reference Format:
Michael Tegegn and Julia Rubin. 2026. The Illusion of Success: Learning-Based Android Malware Detectors (Replicability Study). *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA142 (October 2026), 24 pages. https://doi.org/10.1145/3832233

1 Introduction

Android remains the most popular mobile operating system and consistently accounts for over 65% market share [66]. This popularity also makes it an effective medium for malicious actors, with millions of new Android malware applications observed in the wild each year [44], many of which infiltrate the official app store [23]. Since the introduction of Drebin in 2014 [18], classification-based malware detectors, which learn features differentiating malicious and benign applications from a labeled dataset, have become a viable alternative to signature- and heuristic-based approaches used earlier. Even with the emergence of Large Language Models (LLMs), classification remains a valuable Android malware detection paradigm due to its advantages in terms of cost, scalability,

Authors' Contact Information: Michael Tegegn, The University of British Columbia, Vancouver, Canada, mtegegn@ece.ubc.ca; Julia Rubin, The University of British Columbia, Vancouver, Canada, mjulia@ece.ubc.ca.



ACM 2994-970X/2026/10-ARTISSTA142
https://doi.org/10.1145/3832233

and interpretability [86]: although LLMs demonstrate remarkable semantic reasoning capabilities, real-world mobile applications can span millions of tokens and analyzing an entire application can incur prohibitive token and computational costs. Therefore, LLMs could be better utilized after initial detection, once classification has determined what warrants closer attention.

Papers introducing classification-based malware detection tools often report exceptional performance, with an F1-score for malware detection of around 95% [55, 68]. Yet, prior work has also identified several biases in the evaluation of these tools [17, 57, 62], such as sample duplication (presence of identical or near-identical samples), label inaccuracy (incorrectly labeled or greyware samples), temporal bias (information from "the future" leaking into the training data), evaluation data imbalance (using an unrealistic ratio of benign and malware samples for evaluation), and representativeness/sampling bias (differences in distribution of the malware and benign classes). These biases are studied mostly individually or just a few at a time, on relatively small datasets (fewer than 20K samples), leaving room for exploring the true classification performance of the detectors on large-scale unbiased datasets.

This work aims to fill this gap. To chart the landscape of Android malware detection and evaluation techniques, we start by conducting a comprehensive literature review. Such review is needed as prior surveys are either already outdated [50, 64] or performed on a small scale [12, 20]. We thus organize and classify existing work and further use the most prominent of the identified techniques for our empirical evaluation. Our aim is to answer the following research questions:

RQ1: Which techniques have been applied to design and evaluate malware detection tools?
RQ2: To what extent can we reproduce tool-reported results on large-scale longitudinal datasets and what are the possible reasons for discrepancies?

To answer **RQ1**, in early 2025, we shortlisted 71 top publication venues spanning the fields of Security, Machine Learning, and Software Engineering. We then systematically queried five academic digital libraries and search engines, resulting in a total of 1,446 unique papers. After a series of filtering steps, we identified 158 relevant papers that propose ML-based malware detection tools. For each paper, we recorded the type of information extracted from an application (e.g., permissions, APIs, call graphs). We refer to this information as *application features*. We also recorded the method by which features are *extracted* (e.g., via static or dynamic analysis), the way an application is *represented* for an ML model (e.g., as a vector or image encoding), the type of ML *model* used (e.g., linear or DNN), the *baseline* tools used for comparison (e.g., Drebin), and *malware applications* used for evaluation (e.g., from AndroZoo).

We categorize the features into three main types: *discrete features*, such as APIs (52%) and permissions (46%); *graph-based features*, such as those built using information in call (18%) and control flow graphs (6%); and *raw-code features*, where tools use raw byte code (11%) or code embedding (3%) directly. The vast majority of the tools extract features using static analysis (82%), with dynamic and hybrid analysis accounting for 9% each.

To feed features into an ML algorithm, they have to be represented in a particular format, which we refer to as *app representation*. App representation does not directly correspond to the type of features used; for example, information from a call graph can be represented in an aggregated form, as a count of edges or node fan-out information. For the analyzed tools, vector-based feature encoding is the most common format, with binary vectors (31%), frequency vectors (9%), or a mix of both (11%) used by the majority of tools. Some tools use graph embeddings (19%), represent the code of an app via text embedding (11%), or encode the code as an image (10%). More niche approaches (9% in total) include Markov chains (3%) and more.

For the classification, the analyzed tools use a diverse set of ML models, including DNNs (35%), linear models (33%), tree-based models (27%), K-nearest neighbor models (11%), Bayesian models (9%), and Multilayer Perceptron models (3%).

For evaluation, the most commonly used baselines for comparison are Drebin (22%), MamaDroid (20%), and MalScan (20%). We find that some papers do not perform comparisons with existing tools but rather compare different hyperparameter settings of the tool they present. We also find that while the most commonly utilized malware dataset is the Drebin dataset (37%), which is fairly old by now, several papers use popular application repositories, such as AndroZoo (32%) and VirusShare (32%). Yet, most papers employ their own sampling procedure, which introduces biases and makes it impossible to compare results across papers.

To answer **RQ2**, we select six state-of-the-art tools that utilize features from each distinct type: discrete, graph-based, and raw-code. Specifically, we use Drebin [18] and RevealDroid [34] for discrete features; MamaDroid [55] and MalScan [77] for graph-based features; and InterpretDL [42] and DetectBERT [67] for raw-code features.

We then construct a longitudinal set of datasets, systematically sampling applications from AndroZoo and VirusShare. We stratify samples by years while accounting for sample duplication, label inaccuracy, and temporal biases. For our AndroZoo sets, we additionally account for representativeness/sampling bias by selecting only Google Play applications for both benign and malware classes. Since the VirusShare repository does not include benign samples, for our VirusShare datasets, we had to match malware with benign applications coming from other sources, i.e., AndroZoo. Finally, to account for the data imbalance bias, we build testing datasets with both the common 50:50 and the more recommended 90:10 benign-to-malware ratios.

In total, we collected 115,558 (malware and benign) samples spanning a period of 13 years (2011-2023) from AndroZoo and 35,754 malware samples spanning a period of 11 years (2012-2022) from VirusShare. We could not include samples for 2024 and onward as, at the time of collection (2025), AndroZoo contained fewer than 100 deduplicated malware apps for each year. We divided each dataset into several walk-forward train-test splits, where each split spans four years – the earlier two years for training and the later two for testing – resulting in 18 total splits (10 for AndroZoo and 8 for VirusShare). We ran the tools on each of the 18 splits and analyzed their classification performance in terms of precision, recall, and F1-score.

Our results show that the classification performance of a tool on different splits varies substantially. For example, for the AndroZoo dataset, F1-score ranges between 65.1% and 93.5%, depending on the time period under consideration. This means that any comparison of reported tool accuracy (without empirical evaluation on the exact same set of applications) is practically meaningless. In fact, we observe that any given tool can outperform others on a specific metric in at least one train-test split.

We also observe a correlation between the performance of different tools on the same split. This result suggests that classification performance is largely driven by the inherent properties of the dataset rather than the feature sets used. In fact, a deliberately naïve and unreliable classifier we designed for this study, which uses application package names as classification features, achieves similar and sometimes even higher accuracy than the evaluated state-of-the-art tools, further highlighting the dependence of results on spurious and semantically weak correlations rather than semantically meaningful features.

For the VirusShare dataset, unsurprisingly, using different data sources for benign and malware samples improves tool accuracy compared with that achieved on data from the same data source (as in the AndroZoo case). Furthermore, enforcing the data imbalance scenario in both cases leads to a large drop in F1-score (19.7% on average).

Table 1. Prior Literature Reviews on Android Malware Detection.

Survey	Year	Topic	Years Covered	# Papers	Features	Represent.	Models	Datasets	Baseline
Alqahtani [11]	2019	Android malware + ML	-	9	○	○	●	○	○
Liu et al. [50]	2020	Android malware + ML	2014-2019	~25	●	○	●	○	○
Qiu et al. [64]	2020	Android malware + Deep Learning	- 2019	31	●	●	●	○	○
Pan et al. [61]	2020	Android malware + Static Analysis	2014-2020	98	●	●	●	●	○
Li et al. [47]	2021	All malware + Attacks and Defenses	- 2021	10 (Android)	◐	○	◐	○	○
Muzaffar et al. [58]	2022	Android malware + ML	2013-2020	70	●	○	●	◐	○
Liu et al. [51]	2022	Android malware + Deep Learning	- 2021	-	●	●	●	●	○
Deldar et al. [28]	2023	All malware + Zero-day ML	-	18 (Android)	●	●	●	●	○
Altaha et al. [12]	2024	Android malware + Supervised ML	2017-2024	40	●	○	●	○	○
Bilot et al. [19]	2024	All malware + Graph learning	-	20 (Android)	●	●	●	●	○
Brosolo et al. [20]	2024	PE/Android malware + Vision-based	2018-2024	65 (PE+Android)	●	●	●	◐	○
Maniriho et al. [54]	2024	All malware + Deep Learning	-	32 (Android)	○	○	○	●	○
Survey in our work:		**Android malware + ML**	**2011-2025 (Q1)**	**158 (Android)**	●	●	●	●	●

Overall, our study highlights the false promise of tool-reported results and the difficulty in comparing results across papers. It calls for more advanced detectors that utilize semantic information rather than spurious correlations embedded in data and perform well even on challenging data splits. Furthermore, it highlights the need for concrete, unbiased, and continuously updated datasets, which can drive reliable comparison across tools.

The remainder of the paper is structured as follows. Section 2 outlines our literature review. In Section 3, we discuss our tool evaluation methodology, including dataset construction and tool selection procedures, and present the evaluation results. Section 4 discusses lessons learned and implications of our work. Limitations and threats to validity are described in Section 5. Section 6 discusses the relevant related work and, finally, Section 7 concludes the paper.

2 Literature Review: ML-based Android Malware Detection

Several ML-based tools have been proposed for Android malware detection, with the earliest works dating back to 2012 [63, 76]. Such tools typically construct a dataset of Android applications and extract app features using static, dynamic, or hybrid analysis. They then produce a single app-level representation (a.k.a. encoding), train a classifier, and report the classification result.

Table 1 summarizes recent literature reviews covering Android malware detection tools. Most systematic reviews only include papers up to 2021 [47, 51, 58]. Newer works are conducted on a smaller scale [12] or focus on specific ML techniques, such as Graph-learning [19], Computer Vision [20], and Deep Learning [51, 54]. Unifying all papers from existing reviews would miss important contributions, such as RevealDroid [34], MalScan [77], HomDroid [78], AMCDroid [52], and IntDroid [94]. Thus, to better understand the current tool landscape and motivate the design decisions of our empirical study, we first perform a systematic literature review.

2.1 Review Procedure

Paper Collection. Our paper collection procedure is summarized in the top half of Figure 1. To limit our analysis to high-quality papers, we first collected the list of top conferences and journals from CORE [2] and Google Scholar Metrics [4] spanning the fields of Security, Machine Learning, and Software Engineering. For CORE, we restricted our search to A/A* venues under the three fields. For Google Scholar Metrics, we selected the top 20 venues listed under the three corresponding topics: *Software Systems*, *Computer Security & Cryptography*, and *Artificial Intelligence*. This resulted in a total of 71 venues; the complete list can be found in our online appendix [70].

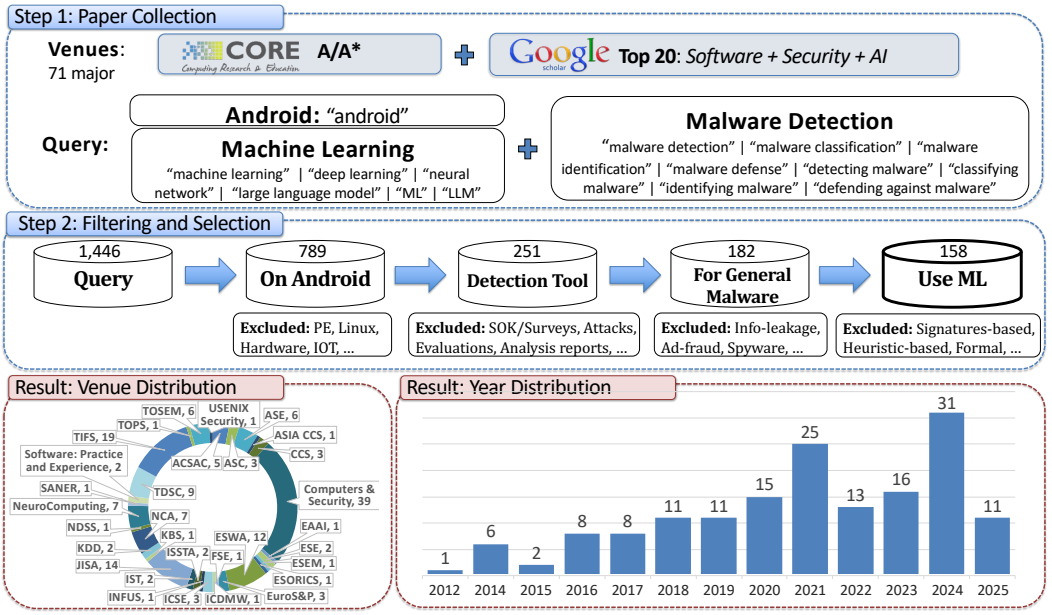


Fig. 1. Literature Review Procedure and Filtering Results.

Since our goal is to identify and characterize Android malware detection tools that utilize ML techniques, we designed a query by expanding terms with possible synonyms for completeness, and using specific word combinations for specificity (to limit false positives). The complete query used is shown at the top of Figure 1. The "+" and "|" signs indicate AND and OR operators, respectively. We used the AND operator with the concepts *Android*, *Machine Learning*, and *Malware Detection*, and the OR operator with diversified terminologies for each concept. Afterwards, we identified the digital libraries that host proceedings of the selected venues for querying: namely, IEEE Xplore [5], ACM [1], ScienceDirect [6], and Springer [7]. For venues that are either not indexed or only partially indexed by digital libraries, we used Google Scholar [3] as a primary search engine.

During paper collection, we used the appropriate digital library for each venue, splitting the query into multiple variants whenever necessary. Moreover, to avoid accidental misses, we used alternative venue names (e.g., *Conference on Neural Information Processing Systems*, *NeurIPS*, *NIPS*) for our Google Scholar search and later removed duplicates. We performed our search in the first quarter of 2025, which resulted in a total of 1,446 papers.

Paper Filtering and Selection. After collecting query results, we performed manual filtering to identify the relevant papers. To start, the first author of this paper and an additional member of our research group performed a pilot run reviewing 20 randomly selected papers to decide on the inclusion/exclusion criteria. Each reviewer independently went through the abstract, introduction, and conclusions of the papers, classifying them as relevant or irrelevant. Papers that propose an original ML-based malware detection approach, i.e., not a re-implementation, are marked as relevant and included for our study. In the case where a paper is marked as irrelevant, the reviewers also recorded the reason for exclusion. After the pilot run, the reviewers discussed the filtering results and standardized the exclusion criteria. Then, the remaining papers were split between the reviewers to perform further independent filtering.

Results of the filtering procedure are shown in the bottom half of Figure 1. We excluded works that (a) study a different software system (e.g., PE and Linux), (b) do not present a detection tool (e.g.,

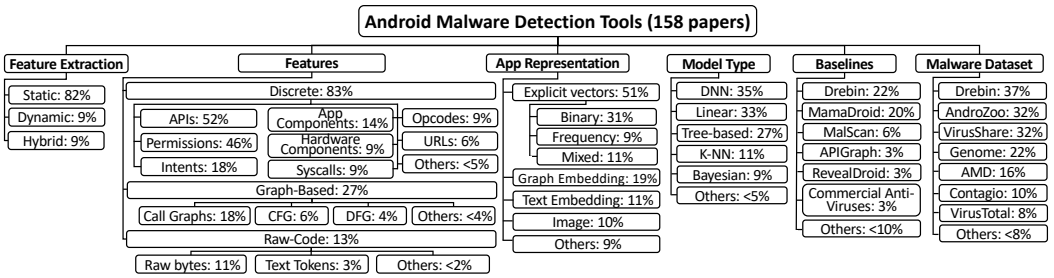


Fig. 2. Statistics of ML-based Android Malware Detection Tools.

are rather SoK/surveys and attacks), (c) present a malware detector but not for general malware (e.g., focus specifically on info leakage, ad fraud, etc.), and (d) present a malware detector but do not use machine learning (e.g., perform signature-based or heuristic-based detection). At the end of our filtering process, we selected for further analysis 158 papers that perform ML-based Android malware detection.

Paper Characterization. We split the 158 papers between the two reviewers, who analyzed and recorded the detection technique and evaluation methodology presented in each paper. Specifically, the reviewers recorded (a) the feature extraction procedure, i.e., whether or not the applications are run to extract features (static, dynamic, or hybrid analysis); (b) the type of extracted features (e.g., permissions); (c) the app representation used to encode the extracted features (e.g., vector vs. image encoding); (d) the type of ML model used (e.g., linear vs. neural net); (e) the baseline tools used for comparison (e.g., Drebin); and (f) the source of applications used for evaluation (e.g., AndroZoo).

2.2 Results

Figure 2 shows the aggregated statistics of the identified papers, which we discuss below. Detailed information about each reviewed paper is available in our online appendix [70].

Features. We refer to the type of information extracted from an application (e.g., permissions, APIs, call graphs) as an *application feature*. We classify features into three types: (a) discrete, e.g., permissions and APIs; (b) graph-based, e.g., control flow and data flow graphs; and (c) raw-code, e.g., code bytes and text tokens. Each type of information extracted from an app can induce more than one feature. For example, if a tool constructs a call graph and then uses it to obtain a count of outgoing edges for each API, we say the tool uses both graph-based and discrete API features. As another concrete example, Wang et al. [72] use the permission specification text, as is, from the Android Manifest file. In this case, we say the tool uses both text token and permissions features.

Among the discrete features, APIs and permissions are, in fact, used by the majority of the tools: 52% and 46%, respectively. Other discrete features include Android Intents (18% of the tools), app components (14%), hardware components (9%), and URLs (6%). Dynamic tools also use system calls (9%), opcode traces (8%), and web traffic info (7%), such as IP addresses and headers.

Several tools utilize *graphs* to capture more semantically-rich information. Call graphs (18%), control flow graphs (6%), and data flow graphs (4%) are the most popular. For example, MamaDroid [55], APIGraph [88], and MalScan [77] all use program graphs and APIs as their graph-based and discrete features, respectively. However, they utilize different types of graphs (call graphs, API dependency graphs) and different information about APIs (API count, graph centrality statistics).

A few tools do not extract any semantic information, using *raw code* (binary or decompiled) directly. Some of these tools (11%) read APKs as *raw bytes* while others (3%) consider *text tokens*.

Feature Extraction. The majority of tools (82%) use *statically* extracted features. Tools attribute their reliance on static features to their efficiency and scalability [18, 34]. Additional approaches

use *dynamic* analysis (9%), which involves running an Android application on a real device or a sandboxed emulator and recording its behavior. For example, both DL-Droid [13] and EnDroid [32] run an APK on a real device or an emulator, respectively, to extract features such as API calls and event logs (DL-Droid) and system calls and network statistics (EnDroid). Some tools rely on features extracted using both static and dynamic, i.e., *hybrid* analysis (9%). For example, DroidDetector [85] combines dynamic app behavior statistics, such as file and network access operations, with statically extracted permissions and API calls. Instead of feature extraction, some tools use malware datasets which already include both statically and dynamically extracted features, namely CICAndMal [45] and KronoDroid [38].

App Representation. Features have to be represented in a format that can be inputted into an ML model. Most tools (51%) use a fixed-dimensional *vector-based* feature representation. In such a representation, each position in the vector contains a binary or continuous value, which directly quantifies a certain concept. A mix of binary and continuous values is also possible. A large number of these tools (31%) use explicit binary vectors [8, 16, 18, 46, 81], e.g., where each vector position corresponds to a specific permission, API, or another discrete feature. Frequency (9%) and mixed vectors (11%) are typically used to store the number of occurrences of a specific feature.

Graph embeddings, used by 19% of tools, e.g., [40, 65, 79] are typically generated by Graph Neural Networks (GNNs) for graph-based features. Yet, some tools choose to represent graph-based features in a different way, e.g., by a frequency-based vector with edge counts.

Text embeddings are used by 11% of the tools to represent raw-code features. For example, DetectBERT [67] uses a customized BERT-like representation model to represent app bytecode.

Images are also used to represent raw-code features, specifically the raw-byte ones (10% of the tools). For example, Iadarola et al. [42] treat code bytes as a sequence of unsigned integers corresponding to grayscale pixels. Yadav et al. [82] further extend this technique by splitting bytes into three channels, corresponding to RGB colors. The generated images are further processed using techniques from computer vision, e.g., Convolutional Neural Nets (CNNs).

Finally, a few tools (9%) use other representation structures, such as transition matrices for Markov Chains [55] and structural entropy values [22].

Model Type. The most commonly used models are various types of DNNs (35%) and linear models (33%). Tree-based (27%), K-nearest neighbors (11%), Bayesian (9%), and Multilayer Perceptrons (3%) models were common in the earlier years [18, 83, 84]. Later, DNNs of various types became popular, including GNNs [65, 79], CNNs [42, 82], and RNNs [30].

Baselines. The majority of papers (60%) evaluate results by comparing with at least one baseline tool. The most commonly used baselines are Drebin (22%) and MamaDroid (20%), which, despite being fairly old, continue to be utilized for comparison even in recent works. Some papers (23%) do not compare with existing baselines but rather evaluate different algorithms (e.g., SVM, RF, DNN) on the same feature set. The remaining papers either report their results without any comparison (12%) or compare against results in other papers, without running comparative experiments themselves (5%). As the sampling strategy differs even for concrete datasets under the same name (e.g., AndroZoo), such comparisons are not trustworthy.

Malware Datasets. The most popular malware dataset, used by 37% of tools, is the Drebin dataset, which comprises 5,560 malware applications from 179 malware families [18]. The next two most popular sources of malware samples, utilized by 32% of tools each, are AndroZoo [10] and VirusShare [74]. AndroZoo is a popular repository of benign and malware applications containing over 24 million APKs at the time of writing. VirusShare is another popular repository that hosts malware samples that target various platforms including Android, PE, and Linux binaries. As both are application repositories, a consistent sampling strategy should be defined for each.

The other popular evaluation datasets are MalGenome [92] (22%), AMD [75] (16%), and Contagio Minidump (10%). MalGenome is arguably the most studied malware dataset. It contains 1,260 malware samples from 49 malware families in years 2010 and 2011, together with manual analysis reports for each sample. AMD contains 24,553 malware samples collected from 2010 to 2016, automatically categorized into 135 varieties and 71 malware families. Contagio Minidump [26] is a collection of ~1,200 malware samples, with analysis reports. The widely used version of Contagio Minidump only contains samples from 2011 to 2013; yet the corresponding blog post contains updates from 2018.

In addition to the limited and outdated set of samples, prior work showed that Drebin, MalGenome, AMD, and Contagio datasets all contain various biases [43, 89], making them unreliable evaluation targets and weak candidates for a replicability study. The application repositories AndroZoo and VirusShare contain a large number of samples and are thus utilized by a large fraction of the tools. Yet, tools utilizing their own sampling strategies lead to evaluation inconsistencies, even when the data source is mentioned. We thus conduct a large-scale replicability study with stratified longitudinal data, for a more reliable tool comparison.

Answer to RQ1: Among the 158 surveyed tools, discrete features (such as APIs and permissions) and graph-based features (such as call graphs and control flow graphs) are used the most. Raw-code features are still less common at the time of analysis. The vast majority of features are extracted through static analysis, with dynamic and hybrid analysis being less popular. Features are mostly represented by explicit vectors (binary, frequency-based, or a mix of the two). Other common approaches are graph embeddings, text embeddings, and images. For classification, tools mostly use DNNs, tree-based models, K-nearest neighbors, and Bayesian models. Only 60% of papers report performing evaluation experiments to compare with baseline tools. Yet, papers often apply a proprietary sampling strategy from large popular repositories, such as AndroZoo and VirusShare, or use outdated datasets, such as Drebin [18], making results incomparable across papers.

3 Empirical Evaluation

To answer **RQ2**, we select six state-of-the-art tools that utilize features from each of the three categories: discrete, graph-based, and raw-code and run them on a carefully constructed longitudinal set of datasets, systematically sampled from AndroZoo and VirusShare repositories. To account for the data imbalance bias, we build testing datasets with both the common 50:50 and representative 90:10 benign-to-malware ratios. In what follows, we present our evaluation setup and results.

3.1 Experimental Setup

For the evaluation setup, we first describe our tool selection procedure and the selected tools. Then, we present our deliberately naïve reference classifier, which we refer to as SIM. Next, we present our dataset construction procedure, including data collection, debiasing, and split-construction. Finally, we explain the list of metrics used to analyze the results.

Tools. We focus on the 27/158 tools with publicly available implementations – either from the original authors or from a third-party. We further exclude two tools that use dynamically extracted features since most malicious servers are non-functional for the malware used in our study. Table 2 shows the remaining 25 tools we analyze in our study, together with the type of features they use (columns 3-5), the source files used for feature extraction (columns 6-8), and the learning setup corresponding to the feature representation and model type (columns 9 and 10). The last column indicates which of the tools are runnable off-the-shelf without requiring significant re-engineering.

We found that 15 of the 25 do not run. Of the remaining 10 tools, we select two commonly used tools for each feature type: For the set of tools that utilize only discrete features, we select

Table 2. Tools with Publicly Available Implementation.

		Ft. Type			Sources			Learning Setup		
Tool	**Year**	**Discrete**	**Graph-based**	**Raw-code**	**Manifest**	**Bytecode**	**Native code**	**Representation**	**Model Type**	**Runnable**
Drebin [18]	**2014**	✓	✗	✗	✓	✓	✗	**Binary Vector**	**Linear**	✓
HMMDetector [22]	2016	✓	✗	✗	✗	✓	✗	Vector - other	Tree-based (DT+RF)	✗
AndroDialysis [31]	2017	✓	✗	✗	✓	✗	✗	Binary Vector	Bayesian	✗
RevealDroid [34]	**2018**	✓	✗	✗	✗	✓	✓	**Frequency Vector**	**Linear+CART**	✓
LoopMC [53]	2018	✓	✗	✗	✗	✓	✗	Frequency Vector	Tree-based (DT)	✗
PermPair [16]	2019	✓	✗	✗	✓	✗	✗	Binary Vector	Threshold	✗
Sec-SVM [29]	2019	✓	✗	✗	✓	✓	✗	Binary Vector	Linear	✓
Chen et al. [24]	2023	✓	✗	✗	✓	✓	✗	Binary Vector	DNN	✗
PAD [46]	2024	✓	✗	✗	✗	✗	✗	Binary Vector	DNN	✗
Meta-MAMC [48]	2024	✓	✗	✗	✓	✓	✗	Vector-other	DNN	✗
DroidSIFT [87]	2014	✓	✓	✗	✗	✓	✗	Graph	Bayesian	✗
ICCDetector [81]	2016	✓	✓	✗	✓	✓	✗	Frequency Vector	Linear	✗
MamaDroid [55, 59]	**2017,2019**	✓	✓	✗	✗	✓	✗	**Markov Chains**	**Tree-based (RF)**	✓
MalScan [77]	**2019**	✓	✓	✗	✗	✓	✗	**Graph**	**RF+KNN**	✓
APIGraph [88]	2020	✓	✓	✗	✗	✓	✗	Vector-other	Tool-specific	✓
S3Feature [60]	2022	✓	✓	✗	✗	✓	✗	Binary Vector	Tree-based (RF)	✗
AMCDroid [52]	2023	✓	✓	✗	✓	✓	✗	Graph Embedding	DNN	✓
MsDroid [40]	2023	✓	✓	✗	✓	✓	✗	Graph Embedding	DNN	✗
GHGDroid [65]	2024	✓	✓	✗	✗	✓	✗	Graph Embedding	DNN	✗
MaskDroid [90]	2024	✓	✓	✗	✗	✓	✗	Graph Embedding	DNN	✗
GNNDroid [79]	2025	✓	✓	✗	✗	✓	✓	Graph Embedding	DNN	✗
BasicBlock [8]	2016	✗	✗	✓	✗	✓	✗	Binary Vector	Linear+Tree-based	✓
InterpretDL [42]	**2021**	✗	✗	✓	✗	✓	✗	**Image**	**DNN**	✓
DetectBERT [67]	**2024**	✗	✗	✓	✗	✓	✗	**Text Embedding**	**DNN**	✓
TIML [68]	2024	✗	✓	✓	✗	✓	✗	Images+Vector	Linear+DNN	✗

Drebin [18] and RevealDroid [34]. Drebin is the most popular tool as per our study summarized in Figure 2, which uses light-weight static analysis and a simple model architecture to build a binary classifier. It first extracts 8 different types of features from the app's bytecode and manifest file: permissions requested in the manifest; permissions actually used by the app through APIs; main app components, such as activities and services; used hardware components, such as Bluetooth and camera; used intent filters, which represent Android or user-defined actions such as battery-related events; API calls guarded by permissions; suspicious API calls, such as `getDeviceId`; and network addresses specified in the app. It then binary-encodes these features based on their presence or absence, and trains a linear Support Vector Machine (SVM) as a binary classifier. Despite its simple architecture, Drebin achieves a very high accuracy on most malware detection datasets. For all experiments using Drebin, we use 1000 top ranked features based on the mutual information gain [49] feature selection metric. RevealDroid is also a light-weight malware detection system that relies on APIs and native system call counts extracted from the bytecode and native files for detection. In particular, RevealDroid extracts features corresponding to package-level API usage, method-level API usage, statically resolvable reflection calls, and system calls from all bundled native files to create a combined frequency feature set. It uses a linear SVM as a classifier to perform malware detection.

For the graph-based features, we select MamaDroid [55] and MalScan [77], as these are the two most commonly utilized baselines. MamaDroid relies on abstracted API sequences to represent the holistic behavior of applications. It then models the invocation relationship of applications as Markov chains where each abstracted API is a state, and the transition probabilities are used as features for classification using a Random Forest classifier. For our experiments, we use the re-implementation made available by Molina-Coronado et al. [57] selecting package name as the

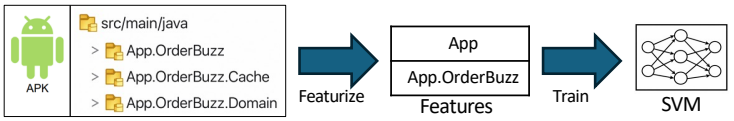


Fig. 3. SIM: Naïve Classifier based on Package Name Similarity.

abstraction level. MalScan utilizes function call graphs to perform social-network-based centrality analysis of sensitive APIs for encoding applications. In particular, it represents apps as a fixed length vector where each position corresponds to the different graph centrality measures (degree, Katz, closeness, and harmonic) of a pre-identified list of sensitive APIs. It then uses different types of classifiers (1-NN, 3-NN, and Random Forest) to perform classification. For our experiments, we directly use their implementation available on GitHub [77].

Among the set of tools that utilize raw-code features, we select DetectBERT [67] and InterpretDL [42]. DetectBERT is a recent work which uses text embeddings extracted through language modeling of bytecode instructions for detection. It first relies on a pre-trained language model to generate embeddings for all classes found in each application. The generated class embeddings are then combined using concepts from multiple instance learning [56] to construct a single app embedding. It later uses a fully-connected final layer to perform malware classification. For implementation, we directly use the source code provided by the authors' GitHub [67]. We also use the pre-trained language model the authors provided upon request to generate the class embeddings. InterpretDL is an image-based detection tool that directly encodes the sequence of bytes found in Android bytecode as greyscale images. These images are then used as input to a CNN to perform malware classification, which are later visualized to identify image regions that contribute to the prediction. We make use of the source code provided by the authors to run our experiments.

Naïve Classifier (SIM). Android malware detection tools utilize either handcrafted features derived from domain expertise or latent representations generated from complex learning approaches such as pre-trained embedding models. To estimate the importance of tool-specific features and contextualize the relevance of our replication results, we augment our list of tools with a naïve classifier that intentionally utilizes non-semantic features. Our tool, referred to as SIM, relies on simple package name similarity to perform malware detection. The design of SIM is shown in Figure 3. It uses package names of defined classes as features to represent each application. In particular, for each application, we first extract the list of defined classes with their full package names to use as features. For example, if the APK contains the class `App.OrderBuzz.Cache`, the package name `App.OrderBuzz` is used as a feature. The package names themselves are non-semantic features since they hold no relation to the actual code contained in the classes.

These package names are extracted by running the dexdump command on all classes.dex files. The set of package names per APK is encoded using one-hot encoding where 1 implies the presence of the package name and 0 implies the absence. Similar to Drebin experiments, we use the 1000 top ranked features based on mutual information gain feature selection metric to train an SVM model for classification.

Datasets. We select two of the most commonly used app repositories in ML-based Android malware detection: AndroZoo [10] and VirusShare [74], utilized by 32% of the tools each. We intentionally avoid using the Drebin and MalGenome datasets as, despite being popular, they are outdated and contain a severely limited number of malware samples (5,560 for Drebin and 1,260 for MalGenome). Additionally, several prior works [43, 89] have already shown different biases contained in the datasets. The left-hand side of Figure 4 outlines our dataset construction procedure for samples we selected from the AndroZoo and VirusShare data sources.

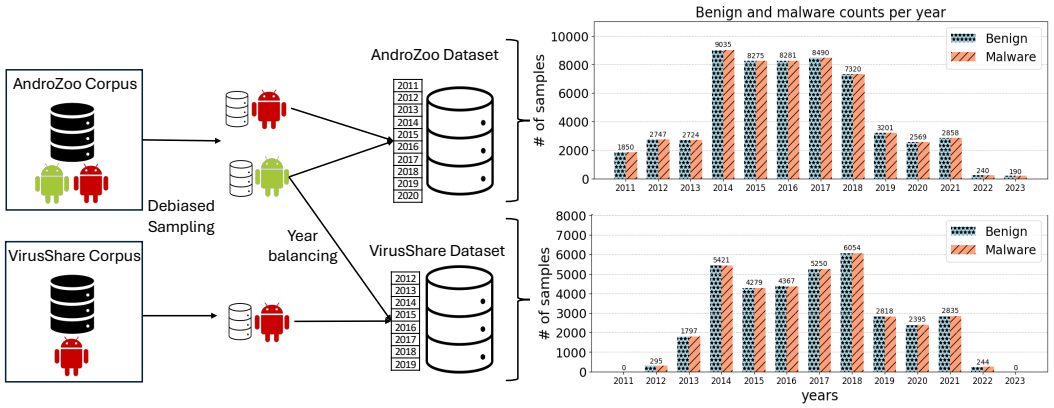


Fig. 4. Dataset Construction Summary.

AndroZoo is a popular repository of benign and malware applications containing over 24 million APKs at the time of writing. To estimate true classifier accuracy, it is important that the experimental setup is free from the known pitfalls mentioned in previous works [17, 62, 89]. Hence, using the metadata provided by the website, we systematically collected a large scale de-biased dataset covering the years 2011 to 2023. The rigorous de-biasing procedure we followed to account for the known experimental pitfalls is described later in this section.

VirusShare [74] is also a popular repository that hosts malware samples for various platforms including Android, PE, and Linux binaries. To sample from this data source, we first downloaded all Android-specific zip files on the website accounting for 4.4 terabytes of compressed data. We then followed the same de-biasing approach as for AndroZoo to construct the final concrete dataset.

Dataset Debiasing Procedure. While ML-based detection shows great promise, several works [17, 57, 62] have outlined biases at different stages of the machine learning workflow used to train and evaluate these models. This section presents a summary of the biases mentioned by the literature, and how we configured our dataset to mitigate them.

(a) Sample Duplication: Sample duplication refers to the case where a subset of samples appear multiple times in the dataset. Several works [57, 89] have shown that sample duplication exists in most of the commonly used datasets and it significantly affects trained models: models perform worse in deduplicated versions of the datasets. Deduplication is commonly performed by computing digest hashes of APKs or bytecode and retaining a single sample for a given hash in the dataset. The most commonly used digest hashes include SHA-1, SHA-256, and MD5 which generate a fixed length cryptographic hash for any type of data which effectively serves as a unique identifier. Other ways to perform deduplication include using featurized vectors or other APK identifiers such as the package name to perform unique sampling. AndroZoo samples are already indexed based on APK hashes, which means that no APK-level duplicates exist in the repository. However, some apps contain multiple versions appearing under the same package which can lead to biased classification results. Therefore, we additionally perform package-name-based de-duplication, sampling the latest available app for a given package name. We use the same approach to de-duplicate the VirusShare dataset.

(b) Label Inaccuracy: Label inaccuracy refers to the presence of incorrectly labeled samples in the dataset. This pitfall is particularly relevant in Android malware detection as datasets are commonly constructed by using antivirus scan results which themselves may change over time [93]. Therefore, to mitigate the prevalence of inaccurate labels, prior works resort to using the latest available scan results from multiple antiviruses. These works further suggest avoiding greyware, i.e., samples

that are not truly malicious but also not fully benign, by using a reasonable threshold of antivirus detections to categorize samples as malware. AndroZoo metadata contains information about the number of detections on VirusTotal [71], a website that hosts over 70 commercial antivirus scanners in addition to other metadata relevant to malware analysis. However, the metadata contains results from scans performed when the samples are initially uploaded to AndroZoo, which makes most entries outdated since VirusTotal antiviruses constantly update their database. As a result, for both AndroZoo and VirusShare datasets we fetched the latest detection statistics (as of January 2025) from VirusTotal by querying the website using the hashes of the downloaded samples. Subsequently, to avoid the presence of greyware, we filtered out all benign samples that have been flagged by at least one antivirus, and all malware samples that have less than five antivirus detections, similar to other works [91].

We chose VirusTotal for labeling mainly due to its scalability as there are 151,312 applications across the two datasets requiring us to use automated labeling tools. Another reason why we chose VirusTotal is for its reliability on older samples. Prior research [93] has shown that antivirus engines become increasingly stable after 1 year due to emerging reports and continuous time-aware monitoring, and that a threshold-based labeling strategy is highly effective to ensure label accuracy. As our samples are at least two years old and we utilize the latest VT reports with threshold-based labeling, we expect the labels to be accurate.

(c) Representative/Sampling Bias: Sampling bias is when a dataset does not represent the data distribution observed in the wild leading to spurious correlations or superficially inflated results [17]. One cause of this bias is data imbalance / spatial bias [62] which involves the unrealistic assumption of the ratio of benign-malware samples in the dataset. Another cause is data source mismatch in which incompatible data sources are used for benign and malware samples. Overall, while this is a very common pitfall that occurs in several applications of machine learning, prior works have suggested that it can be mitigated by not sampling applications from different sources, having sufficient data, and using a reasonable benign-malware ratio. Since AndroZoo contains APKs from various markets and repositories such as Google Play [36], AppChina [15], and Anzhi [14], we focus only on Google Play applications for both the malware and benign class. This reduces the risk of data-source-related spurious correlations between samples when performing classification tasks. To ensure both classes are well represented and enable experiments with different benign-to-malware ratios during evaluation, we balance the benign and malware samples for each year. We do this by downsampling the majority class to have the same number of samples as the minority class for each year. For the VirusShare dataset, using the same source for benign and malware samples is not possible as the dataset only contains malware samples. Therefore, similar to detection tools, we pair the collected malware dataset with the benign samples from AndroZoo. Specifically, for each yearly subset of VirusShare malware samples, we randomly sample the same number of AndroZoo benign samples from the same year.

(d) Temporal Bias: Temporal bias [62] refers to the case where future information which should only be part of the testing data is also present in the training data leading to an unrealistic evaluation scenario. In particular, as malware samples evolve over time, experiments should ensure that no time-related information that is unavailable in practice leaks into the training data. This is different from traditional data snooping, where testing samples are indirectly used to tune model hyper-parameters. Rather, it corresponds to the train-test split construction itself and its time-awareness. To prevent classifiers from having future knowledge (a.k.a. temporal inconsistency) we perform experiments on temporally consistent train-test splits.

The top right of Figure 4 shows the year-by-year sample distribution of the resulting AndroZoo dataset after the de-biasing procedure making up a total of 115,558 samples spread across 13 years.

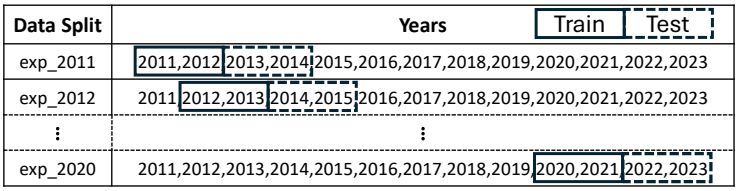


Fig. 5. Walk-forward Time-aware Dataset Splits.

For all years, the sample count per year is limited by the number of malware samples AndroZoo contains for that year. Consequently, the final dataset contains fewer samples for earliest and latest years. The bottom right of the figure shows the sample distribution of the VirusShare dataset for each year making up a total of 71,508 samples.

Train-Test Splits. In addition to the time-awareness constraint, we perform evaluations on multiple splits per dataset to measure the change in metrics at different time periods. Practically, this corresponds to different deployment times of the same tool given a dataset source. Figure 5 shows how the datasets are split into multiple temporally consistent subsets that make up a single experiment. Each subset spans four years, where samples from the first two years are used for training, and samples from the later two years are used for testing. The two-year testing period represents a realistic time window for which a malware detection tool is expected to remain operational. This is particularly the case since accurate labels needed for training require time to collect as antivirus software can update its prediction even two years after an app first becomes available [73]. The two-year training period is selected after experimentation with different cut-offs (e.g., one year, three years, all prior years) which all gave comparable, but slightly worse results.

In this work, we refer to experiments with the starting year of the time window, e.g., S_{2011} refers to the experiment where samples from 2011-2012 are used for training, and ones from 2013-2014 are used for testing.

As class imbalance poses a practical challenge in Android malware detection, we evaluate the tools in balanced and imbalanced versions of each split. We consider an imbalance scenario where benign-to-malware composition in the testing data is 90:10, which is considered a realistic estimate of malware prevalence in the wild [62]. The imbalance scenario is achieved by downsampling test malware apps, without changing the number of benign apps.

Evaluation Metrics. Similar to prior work, we report precision, recall, and F1-score on the test set for all our experiments, with malware as the positive class. In addition, to quantify how well an individual app statistic (e.g., APK size) separates malware from benign apps, we compute the Average Precision (AP) metric: the weighted mean of precision values obtained by varying the classification threshold over that statistic, with the increase in recall at each threshold used as weight (equivalently, the area under the precision-recall curve). A high AP means that a simple threshold-based classifier on this statistic alone would effectively separate the two classes.

3.2 Results

Figures 6 and 7 show the classification performance of tools on the walk-forward splits of AndroZoo and VirusShare datasets, respectively. The top and bottom rows in each figure correspond to the balanced and imbalanced scenarios considered in our study. The horizontal axis represents the train-test split, and the vertical axis encodes the value of each computed metric, namely precision, recall, and F1-score. Each scatter line corresponds to an evaluated tool. To ensure fair comparison of results in each split, we only include tools that successfully run and extract features for at least 90% of the samples. This explains the omission of RevealDroid for the last two splits in both

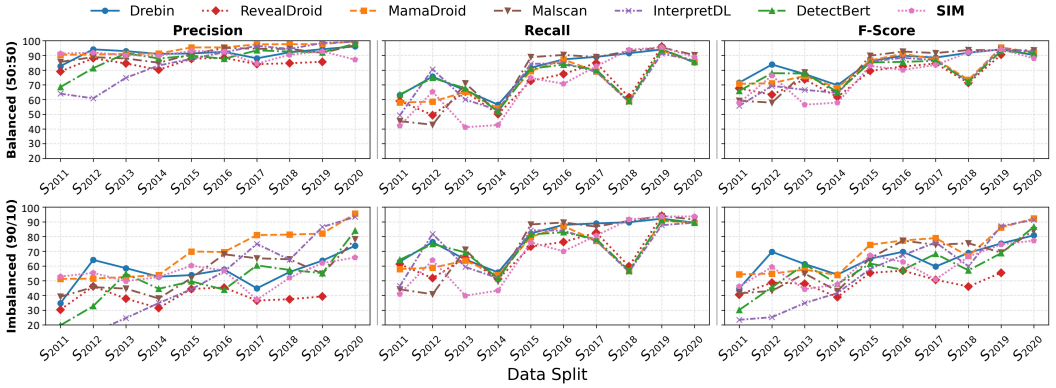


Fig. 6. Precision, Recall, and F1-score Results on AndroZoo Dataset.

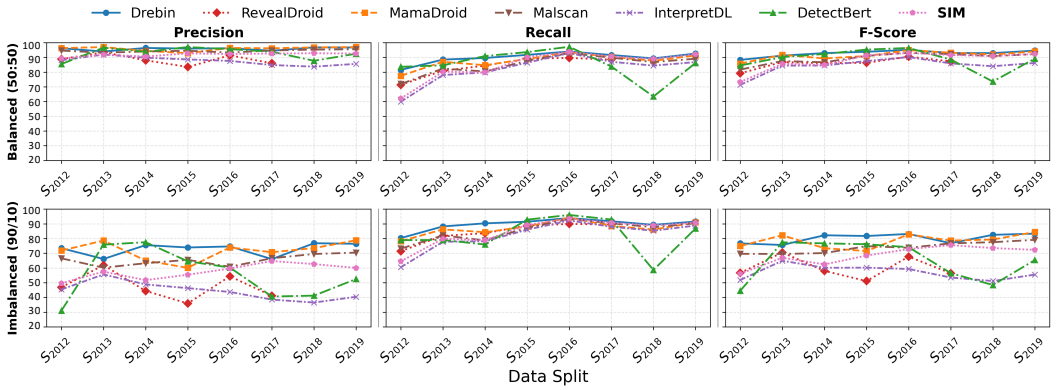


Fig. 7. Precision, Recall, and F1-score Results on VirusShare Dataset.

datasets. Furthermore, most of our results discussion is on AndroZoo dataset since it does not contain the sampling bias that was unavoidable for VirusShare.

This section discusses key observations from our experiments, along the following variability dimensions: across AndroZoo vs. VirusShare datasets in a balanced scenario; across balanced vs. imbalanced scenario; across benign vs. malware classes; across different train-test time splits; across tools; and across years, i.e., first vs. second year in testing data.

Variability Across Datasets. As shown in Figures 6 and 7, in the balanced scenario, almost all tools achieve higher scores for VirusShare dataset vs. AndroZoo dataset. This difference is statistically significant ($p<0.05$ in paired t-test) in four of the six tools when considering all splits and all six tools when excluding the split with the fewest samples (S_{2019}). The results highlight the impact of sampling bias, with benign applications in VirusShare experiments coming from AndroZoo.

Interestingly, this sampling bias introduces non-semantic differences that can facilitate easy separation of the benign and malware classes, e.g., using app size as a feature. The first two rows for each dataset in Table 3 show the APK sizes for the two classes in each split. For AndroZoo, benign samples are 17.3 MB on average while malware samples are 14.3MB. However, for VirusShare, benign samples, paired from AndroZoo, are still around 17.5MB on average while malware samples are 7.3MB. The third row for each dataset in the table shows the AP of a threshold-based classifier that relies on APK size as its threshold. This result demonstrates that such a classifier consistently achieves higher results on the VirusShare dataset compared to the AndroZoo: an average of 72.3% on VirusShare and 51.6% on AndroZoo.

Table 3. Per-split Average APK Size and Average Precision of a Threshold-based Detector using APK Size.

Dataset	Stat.	S_{2011}	S_{2012}	S_{2013}	S_{2014}	S_{2015}	S_{2016}	S_{2017}	S_{2018}	S_{2019}	S_{2020}	Avg.
AndroZoo	Avg. Benign Size (MB)	5.3	7.0	8.8	11.2	14.9	18.1	21.5	25.9	29.6	30.9	17.3
	Avg. Malware Size (MB)	6.4	8.7	10.6	10.8	11.9	13.1	14.4	18.5	23.7	25.4	14.3
	Average Precision (%)	44.1	45.4	46.5	52.0	61.2	62.0	52.6	50.4	53.8	48.3	**51.6**
VirusShare	Avg. Benign Size (MB)	-	7.3	8.8	11.3	15.7	19.1	22.1	26.2	29.8	-	17.5
	Avg. Malware Size (MB)	-	4.6	4.6	6.3	6.1	7.4	8.7	8.7	12.0	-	7.3
	Average Precision (%)	-	61.1	74.1	75.9	77.4	76.5	64.7	71.5	76.9	-	**72.3**

Table 4. Average F1-score Drop per Tool in Balanced vs. Imbalanced Test Scenarios (%).

Dataset	Drebin	RevealDroid	MamaDroid	MalScan	InterpretDL	DetectBERT	Avg.
AndroZoo	19.4	**26.0**	**11.5**	18.6	21.2	21.6	**19.7**
VirusShare	**12.5**	26.0	13.1	15.5	**27.2**	23.9	**19.7**

Table 5. Tool-average Per-class Accuracy for each Split (%).

Dataset	Class	S_{2011}	S_{2012}	S_{2013}	S_{2014}	S_{2015}	S_{2016}	S_{2017}	S_{2018}	S_{2019}	S_{2020}	Avg.
AndroZoo	Malware	56.3 ± 7.4	63.6 ± 15.6	65.9 ± 3.6	52.8 ± 2.3	81.3 ± 5.3	85.1 ± 4.4	83.5 ± 4.8	70.4 ± 17.0	94.0 ± 1.7	86.6 ± 2.1	73.9
	Benign	83.4 ± 9.9	84.6 ± 18.5	90.0 ± 5.6	91.7 ± 2.9	91.3 ± 2.7	92.4 ± 3.0	92.7 ± 5.7	94.2 ± 3.7	92.6 ± 5.9	98.1 ± 1.3	91.1
	Diff	**+27.1**	**+21.0**	**+24.1**	**+38.9**	**+10.1**	**+7.3**	**+9.2**	**+23.8**	**-1.3**	**+11.5**	**+17.2**
VirusShare	Malware	-	73.7 ± 7.9	82.6 ± 4.3	82.8 ± 4.8	89.5 ± 2.5	93.0 ± 2.3	90.0 ± 2.5	79.7 ± 14.4	89.1 ± 3.0	-	85.0
	Benign	-	92.1 ± 6.2	95.2 ± 1.7	93.9 ± 3.5	91.7 ± 5.2	92.2 ± 3.5	88.9 ± 5.5	91.7 ± 6.1	93.3 ± 5.0	-	92.4
	Diff	-	**+18.5**	**+12.6**	**+11.1**	**+2.2**	**-0.8**	**-1.1**	**+12.0**	**+4.3**	-	**+7.4**

Variability Across Balanced vs. Imbalanced Scenarios. The top and bottom halves in each of Figures 6 and 7 show the classification results for the balanced (50:50) and the imbalanced (90:10) benign-to-malware ratio, respectively. The precision results show the significance of accurate benign classification when enforcing the imbalance scenario, with tools that perform well on benign apps being the least affected in the final reported F1-score. There is no difference in recall across the two scenarios as the malware class is downsampled to create imbalanced variants, and recall values are retained due to random sampling.

The high impact on precision directly stems from the lower number of malware samples in the imbalance scenario: as we downsampled malware apps without changing the number of benign apps, the relative proportion of true positives (TP) decreases compared to false positives (FP). This leads to a decrease in the precision value (P), which is calculated as $\frac{TP}{TP+FP}$. The decrease is notable in Figures 6 and 7, implying a large fraction of false-positive results in practical settings.

Table 4 shows the average drop in F1-score of each tool when comparing the balanced and imbalanced scenarios across all splits of the two datasets. For example, the first value in the table indicates that Drebin achieves 19.4% lower F1-score in the imbalanced scenario compared to the balanced one, on average, in the AndroZoo dataset. While all models exhibit a significant drop in F1-score (19.7% on average) when evaluated on the imbalanced scenario, the least affected tool from the AndroZoo dataset is MamaDroid, which shows a decrease of 11.5%. Our results further emphasize the importance of reporting the test class imbalance ratio in tool evaluation [62], as this often overlooked data substantially affects the outputs of the commonly used metrics.

Variability Across Classes. Figures 6 and 7 also show a clear difference in classification results between benign and malware classes. For the balanced dataset splits, tools consistently exhibit higher precision than recall across the splits due to a smaller number of false positives compared to false negatives. In fact, in the AndroZoo dataset, apart from S_{2011} and S_{2012} splits, tools achieve a precision of above 90% indicating the relative ease of benign classification. This is in sharp contrast to recall which drops to around 50% for splits spanning earlier years.

Table 5 further clarifies the difference between the two classes through per-class accuracy metrics; malware accuracy is the same as recall and benign accuracy is commonly referred to as *specificity*. The first two rows for each dataset correspond to benign and malware accuracies, and the third row

Metric	Dataset	Stat.	S_{2011}	S_{2012}	S_{2013}	S_{2014}	S_{2015}	S_{2016}	S_{2017}	S_{2018}	S_{2019}	S_{2020}	Avg.
Precision	AndroZoo	Min	64.0	60.8	74.8	80.2	87.7	87.7	84.1	84.7	85.6	96.1	
		Max	90.4	94.1	92.9	91.3	95.4	95.4	97.5	97.6	98.4	99.5	
		Mean	**78.4**	83.9	87.1	86.4	90.4	91.8	92.3	92.6	93.3	**97.9**	
		SD	**10.2**	**12.1**	**6.7**	**4.4**	**2.9**	**3.3**	**5.2**	**4.4**	**4.7**	**1.4**	**5.5**
	VirusShare	Min	-	59.9	78.0	79.8	86.4	89.6	83.8	63.4	86.2	-	
		Max	-	83.6	88.5	90.9	93.6	97.2	91.6	89.3	92.6	-	
		Mean	-	**74.3**	83.6	85.0	89.6	**93.4**	88.5	82.3	89.2	-	
		SD	-	**8.6**	**3.9**	**4.6**	**2.6**	**2.4**	**2.8**	**10.7**	**2.8**	-	**4.8**
Recall	AndroZoo	Min	45.3	42.9	60.0	50.1	72.7	77.4	78.8	58.3	91.2	85.1	
		Max	63.4	80.6	71.0	56.5	88.8	90.3	89.0	92.9	96.0	90.2	
		Mean	56.3	63.6	65.9	**52.8**	81.3	85.1	83.5	70.4	**94.0**	86.6	
		SD	**7.4**	**15.6**	**3.6**	**2.3**	**5.3**	**4.4**	**4.8**	**17.0**	**1.7**	**2.1**	**6.4**
	VirusShare	Min	-	71.3	84.4	84.5	86.3	90.2	85.9	73.6	86.2	-	
		Max	-	88.3	91.7	92.9	95.3	96.4	93.2	92.9	94.6	-	
		Mean	-	**81.8**	88.6	88.6	90.8	**93.4**	90.0	86.7	91.2	-	
		SD	-	**6.0**	**2.9**	**3.4**	**3.5**	**2.6**	**3.2**	**8.1**	**3.5**	-	**4.2**
F1-score	AndroZoo	Min	55.7	57.9	66.6	61.7	79.5	82.5	84.5	71.3	90.5	90.2	
		Max	71.5	83.8	78.5	69.7	89.8	92.6	91.6	93.6	95.4	93.5	
		Mean	**65.1**	70.6	75.0	65.5	85.5	88.3	87.5	79.0	**93.5**	91.9	
		SD	**6.4**	**9.4**	**4.4**	**2.9**	**3.4**	**3.7**	**2.4**	**10.6**	**1.7**	**1.2**	**4.6**
	VirusShare	Min	-	85.5	91.8	88.0	83.5	87.6	84.8	83.7	85.6	-	
		Max	-	96.3	97.0	94.4	97.0	96.4	96.2	96.8	97.0	-	
		Mean	-	**90.4**	**94.0**	91.8	91.5	92.8	91.4	91.3	92.6	-	
		SD	-	**4.1**	**2.3**	**2.7**	**4.8**	**3.2**	**4.8**	**5.5**	**4.4**	-	**4.0**

Table 6. Per-split Min, Max, Mean, and Standard Deviation of Precision, Recall, and F1-score across Tools (%).

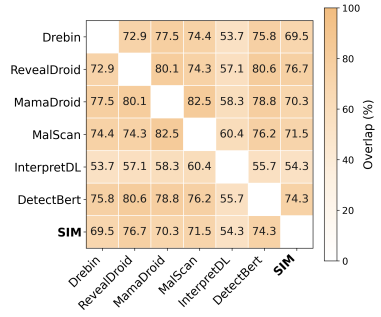


Fig. 8. Jaccard Similarity of Misclassified Samples for Pairs of Tools in AndroZoo's Worst Performing Split (S_{2014}).

explicitly computes the diff calculated as benign accuracy minus malware accuracy. For most of the AndroZoo splits, there is a significant difference of 17.2% on average, which can be as high as 38.9% in S_{2014} split. While the difference is not as pronounced in the VirusShare dataset (7.4% on average), the trend remains that splits with lower F1-score (e.g., S_{2012}) still have decent (>90%) benign accuracy while the malware accuracy is much lower (<85%). This implies that most classifiers default to the benign class for samples not resembling training data. This aligns with expectations of behavioral changes in malware samples observed in the wild, i.e., new strands of malware appearing, may mislead trained classifiers in temporally consistent evaluations.

Variability Across Splits. As shown in Figure 6 of the AndroZoo dataset, there is a significant variation in recall (and F1-score) across the splits, with most tools performing relatively better in the later years (2015-2020) than earlier years (2011-2014). Specifically, all tools have less than 60% recall for S_{2014} split, while they achieve more than 91% for the S_{2019} split.

Table 6 shows more explicitly that the per-split average F1-score in the AndroZoo dataset ranges from 65.1% in the S_{2011} to 93.5% in S_{2019}. The observed variation in results, despite all samples being collected from the same data source, highlights the unreliability of a single-split evaluation setup. Additionally, it implies that a varying degree of distribution shift is present in malware applications over the 13 year period used in our study. While the size of training/testing data varies across splits, the size of the dataset does not seem to have substantial impact on the results. For example, despite having the largest number of training samples (34,620), S_{2014} induces the worst recall and F1-score, for all tools.

Variability Across Tools. All tools report very similar precision across the splits. While there does exist some variation in recall, the general trend in classification performance remains the same. In particular, DetectBERT and MamaDroid have very similar trends across the majority of splits (2013-2020). The similarity becomes even more pronounced in the VirusShare dataset as all tools achieve relatively high scores across all splits. Surprisingly, our experiments show that our naïve classifier SIM performs similar to the state-of-the-art tools in most splits across the two datasets, and sometimes even outperforms all tools, e.g., AndroZoo S_{2018}. In fact, it performs nearly as good as Drebin, which is the best performing tool overall, across the splits (~5% less F1-score on average across all splits). The remarkable success of our naïve classifier shows how superficial tool-reported results can be even in debiased datasets, and the relative ease of achieving high F1-score for datasets like VirusShare.

Table 7. Tool-average Yearly Per-class Accuracy for AndroZoo and VirusShare Datasets (%): recall (TPR) for the malware class and benign recall (TNR), a.k.a. specificity, for the benign class.

Dataset	Class	Stat.	S_{2011}	S_{2012}	S_{2013}	S_{2014}	S_{2015}	S_{2016}	S_{2017}	S_{2018}	S_{2019}	S_{2020}	Avg.
AndroZoo	Malware	Year 1	68.2 ± 17.0	73.7 ± 12.0	68.4 ± 7.0	74.9 ± 3.0	83.5 ± 5.0	87.3 ± 3.0	88.4 ± 2.0	78.4 ± 10.0	94.4 ± 1.0	85.7 ± 3.0	80.3
		Year 2	52.7 ± 8.0	52.7 ± 19.0	63.4 ± 3.0	31.1 ± 3.0	78.7 ± 6.0	80.1 ± 7.0	77.3 ± 8.0	63.1 ± 21.0	86.8 ± 4.0	87.8 ± 1.0	67.4
		Diff	**-15.5**	**-21.0**	**-5.0**	**-43.8**	**-4.8**	**-7.2**	**-11.1**	**-15.3**	**-7.6**	**+2.1**	**-12.9**
	Benign	Year 1	88.1 ± 7.0	86.0 ± 16.0	91.2 ± 5.0	93.0 ± 2.0	93.1 ± 2.0	93.1 ± 3.0	94.1 ± 4.0	94.8 ± 3.0	92.7 ± 5.0	98.2 ± 1.0	92.4
		Year 2	82.0 ± 10.0	83.2 ± 19.0	88.8 ± 6.0	90.4 ± 4.0	89.3 ± 3.0	90.8 ± 4.0	90.8 ± 7.0	93.6 ± 4.0	93.2 ± 4.0	98.1 ± 2.0	90.0
		Diff	**-6.1**	**-2.8**	**-2.4**	**-2.6**	**-3.8**	**-2.3**	**-3.3**	**-1.2**	**+0.5**	**-0.1**	**-2.4**
VirusShare	Malware	Year 1	-	76.4 ± 7.0	82.6 ± 4.0	86.5 ± 6.0	85.3 ± 4.0	95.1 ± 1.0	91.5 ± 4.0	81.5 ± 12.0	89.0 ± 3.0	-	86.0
		Year 2	-	70.2 ± 8.0	82.5 ± 4.0	79.7 ± 4.0	93.1 ± 1.0	88.4 ± 4.0	88.2 ± 2.0	78.3 ± 13.0	89.9 ± 3.0	-	83.8
		Diff	-	**-6.2**	**-0.1**	**-6.8**	**+7.8**	**-6.7**	**-3.3**	**-3.2**	**+0.9**	-	**-2.2**
	Benign	Year 1	-	91.6 ± 5.0	95.5 ± 2.0	95.9 ± 2.0	93.6 ± 3.0	93.1 ± 3.0	91.5 ± 4.0	94.2 ± 4.0	93.5 ± 5.0	-	93.6
		Year 2	-	92.8 ± 6.0	94.9 ± 2.0	92.3 ± 4.0	90.1 ± 7.0	90.4 ± 5.0	85.9 ± 6.0	89.7 ± 7.0	91.6 ± 5.0	-	91.0
		Diff	-	**+1.2**	**-0.6**	**-3.6**	**-3.5**	**-2.7**	**-5.6**	**-4.5**	**-1.9**	-	**-2.6**

The tools themselves produce similar results, as indicated by the low standard deviations in Table 6: less than 5% for F1-score with the exception of the S_{2012} and S_{2018} splits. There is also a general consensus on the most and least challenging splits, i.e., the S_{2011} and S_{2014} are the most challenging for all tools, while S_{2019} is the least challenging. These observations suggest that the primary source of variation across splits lies in the dataset, not in the choice of tools or feature sets.

To further understand the degree of similarity of the tools, we examine the overlap in misclassified samples. For each split, we record the samples misclassified by each tool, which we refer to as *misclassification sets*. Figure 8 shows the overlap in misclassified samples among the six tools in the lowest performing AndroZoo S_{2014} dataset. Each cell encodes the Jaccard similarity of the misclassification sets of the two tools corresponding to the cell. E.g., Drebin and RevealDroid share 72.9% of the misclassified samples as measured in Jaccard similarity. The figure shows that there is a significant overlap in misclassified samples for all pairs of tools, which suggests hardness of classification lies inherently in the dataset. As such, ensemble methods that combine predictions of multiple tools would not be effective to improve the classification results reported in our datasets.

Variability Across Years. As our dataset used two-year time windows to create train-test splits, we analyze how classification results change across year 1 and 2 of the test period by examining the success of the tools for each subset. Table 7 shows the difference in accuracy across the two year testing periods for the AndroZoo and VirusShare datasets for each of the two classes. For each class, the first and second rows show the average and standard deviation of tool accuracy for the first year and second year of the testing data subset, respectively. The table shows that, for the AndroZoo dataset, there is a significant decrease in malware accuracy (a.k.a. recall) with one year gap between training and testing samples: an average of 12.9% but can be as much as 43.8% as in S_{2014}. Surprisingly, for VirusShare splits, there is little difference (max of ~8%) in the average malware recall for the first and second year of the testing period (2.2% on average). An outlier split is S_{2015} in the VirusShare dataset in which tool-average recall is 7.8% higher for the later year. The relatively low difference in the VirusShare dataset suggests that the discriminative features the models learned due to the data source bias outweigh the difficulty in classification coming from app evolution (year-1 vs. year-2 samples).

Answer to RQ2: Most tools perform substantially better on VirusShare compared to AndroZoo, which suggests that the data source difference allows easy separation due to spurious correlations. Moreover, enforcing the data imbalance scenario leads to large drop in F1-score (19.7% on average), which highlights the large fraction of false-positives in practical scenarios. Notably, we also found that there is a substantial difference in tool performance on different splits of the same dataset, that ranges from 65.1% to 93.5% F1-score, even if the splits are constructed using the exact same methodology from the same dataset. Yet, the tools show high similarity in terms of both the achieved results and the misclassified samples, which suggests that tool performance is largely driven by the

nature of the dataset rather than the feature set used. Surprisingly, our intentionally naïve classifier performs similarly to these detection tools, showing that features do not need to be semantically meaningful to achieve high F1-score in some splits.

4 Discussion and Future Work

Reliable Evaluation of ML-based Detectors. Our study highlights the false promise of tool-reported results and the difficulty in comparing results across papers. Specifically, the study shows that tools' accuracy can be severely inconsistent, depending on the data split used for evaluation, even when controlling for the known biases. In fact, for any given tool, one can find a clean dataset setup for which it outperforms other state-of-the-art tools. This calls for a more rigorous tool evaluation methodology, which relies on concrete and unbiased datasets, shared between the tools. Such datasets should be continuously updated, to ensure they stay current with the up-to-date malware. In addition to ensuring reliable comparison between tools, it could be beneficial to include comparisons with naïve baselines, to contextualize the added value of proposed approaches.

Increasing Sample Diversity. Our experiments, as well as prior works, have shown that datasets used for evaluation of ML-based detectors are not sufficiently diverse, skewing evaluation metrics towards the over-represented types of samples. One possible way forward here is to use dataset filtering techniques, such as declustering through stratified sampling [25], treating clusters as subpopulations. Another approach could rely on using artificial data augmentation techniques, which can increase sample diversity and mitigate non-semantic similarity in the dataset. However, such data augmentation approaches should carefully avoid introducing data-source-related bias, e.g., by mixing applications from different markets and online repositories [17].

Spurious Correlations. Android malware datasets, even after debiasing, contain well-generalizing non-robust signals, as can be seen with the success of our naïve classifier. Detecting, quantifying, and explaining the source and impact of such signals could be a productive direction for possible future work. Moreover, developing algorithms that are robust to such correlations, even in the face of low-diversity datasets, would be beneficial, as such signals can be hard to detect.

Interestingly, our results show that some splits appear inherently challenging (e.g., AndroZoo S_{2014} split) and that there is no correlation between tools' success on data splits and their release date. That is, older tools, such as Drebin, perform better in later splits, compared to the splits spanning years close to the tool's release date. This further demonstrates that the main determinant of tool success is the dataset itself rather than the features used by the tools. As such, developing advanced detectors that utilize semantic information rather than spurious correlations embedded in data is needed.

Data Imbalance and Classification Paradigms. In our experiments, misclassifications are common across tools (Figure 8), making tool ensembling impractical. Focusing on feature sets and representations that analyze the app from multiple different angles could help mitigate this issue.

Moreover, we observe that due to the highly imbalanced nature of the available data (the number of available benign samples is orders of magnitude larger than the number of malicious samples from the same type and origin, e.g., less than 1% of 2.5 million mobile applications in Google Play are found to be malicious), the selection of benign samples for training could affect the accuracy and reliability of the detection. Such significant data imbalance cannot be effectively addressed with simple under- or over-sampling, as it will only intensify the unreliable correlations discussed above, calling for data-property-aware sampling techniques [80].

Finally, as malware apps can generally do everything benign software does (and more), the question about the appropriateness of binary classification for malware detection (rather than outlier detection and other forms of classification) also warrants future research.

5 Threats to Validity

For **internal validity**, one threat is the potential configuration errors during our adoption of existing implementation of tools. We mitigated this problem by not making any changes to the main logic of the provided implementations, even adhering to their pre-configured input pipelines. When multiple configurations are viable, we run a grid search on hyperparameters and select the configuration with the best classification performance. Another potential threat is our reliance on the Dex compilation time to date applications, which leads to the exclusion of applications that have intentionally modified Dex dates (e.g., 1980) limiting the representativeness of the resulting datasets. However, we made a conscious decision to use this timestamping strategy as prior work [39] has shown compilation dates to be more reliable, when falling within realistic bounds, than other methods such as first seen times on VirusTotal. Our strict debiasing procedure may also lead to the incorrect exclusion of samples. Package-name-based deduplication can incorrectly exclude repackaged malware, and enforcing zero VirusTotal flags for the benign class can easily exclude true benign samples that have a few false positive flags. Moreover, enforcing no duplicates may result in a distribution different from the in-the-wild scenario. We compromise on these trade-offs to realize our primary goal of creating a fully debiased dataset. For the literature review part of our work, there is a possibility that we could have missed relevant papers due to misinterpretation of their contributions. To mitigate this threat, each paper was marked as relevant/irrelevant independently by two authors and conflicts were resolved after iterative discussions.

For **external validity**, the main concern is that our results do not generalize beyond the datasets and tools considered in our study. To address this, we motivated our experiment design on what is popularly used in the literature by performing an extensive preliminary study on the common evaluation setups and selecting the most representative tools and datasets. We acknowledge that the generalization of our findings can be improved by including tools that expand our feature-set coverage, e.g., dynamic analysis-based tools. However, we note this diversification does not significantly impact the main goal of our work which was demonstrating that successful classification of ML-based detectors can be inconsistent and insignificant rather than performing a detailed comparison of state-of-the-art tools.

6 Related Work

Different works perform empirical studies to evaluate the true classification performance of malware detectors while studying specific biases or properties related to incorrect evaluation setups. Table 8 lists major contributions towards this direction. Works such as Arp et al. [17, 37] identify the pitfalls commonly observed in security such as sampling bias and label inaccuracy, and perform a prevalence analysis by examining papers published in top-tier security venues. Several other works systematically evaluate the impact of specific biases, such as sample duplication [57, 89], temporal inconsistency [9, 41, 62], class imbalance [57], and label quality [57] on the success of tools. While these works evaluate biased dataset setups and compare them to results from unbiased datasets (control group), our work examines the existence of remaining issues even after dataset debiasing. The most similar work to ours is probably Gao et al. [33], where the authors evaluated 10 detection tools in the presence of obfuscation, concept drift, and adversarial attacks. Similar to other works, their evaluation of obfuscation and adversarial examples is carried out by artificially introducing altered samples in testing data and observing impact on classification performance of the tools. The similarity with our work lies in their evaluation of concept drift caused by software evolution, in which they considered a year-balanced dataset scenario where training data is from 2016 and testing data is from 2017-2020. However, this evaluation considers a single time window, which

Table 8. Prior Empirical Evaluation Studies in Android Malware Detection.

Study	Year	Focus	Experimental Setup			
			Tools	Benign Dataset	Malware Dataset	Split Construction
Pendlebury et al. [62]	2019	Temporal/Spatial bias	DB, MMD	AZ (117K)	AZ (13K)	Spatio-temporal
Cai [21]	2020	Sustainability study	MMD, RvD, DSp, DSv, MFl, Afo	AZ+GP (15K)	AZ+VS (13K)	Cross-validation + Time-aware
Ge et al. [35]	2021	Property Bias	DB, QJ (Opcode)	GP, Other Markets (11K)	Genome, DB, VT (17K)	Time-aware (malware only)
Zhao et al. [89]	2021	Sample duplication	DB	AZ (Size Unclear)	Genome, DB, AMD, RMV (40K)	Cross-validation
Daoudi et al. [27]	2022	Drebin Effectiveness	DB	AZ (700K)	AZ (45K)	Cross-validation
Arp et al. [17]	2022	Bias sources	DB, AMD, OPSEQS	AZ (10K)	AZ (1K)	Random (inferred)
Molina-Coronado et al. [57]	2023	Duplication, Label quality, Imbalance	DB, MMD, AD, BB, DD, DDct, HMM, ICC	AZ (9K)	AZ (18K with greyware)	Cross-validation + Spatio-temporal
Gao et al. [33]	2024	Robustness under obfuscation, drift, and AEs	DB, MMD, RvD, MFl, ALD, Bai21, ImgD, MDMC, MaS, AGr, EFCG	AZ, CICMalDroid (12K)	AZ, CICMalDroid (16K)	Cross-validation + Time-aware
Evaluation in our work	**2026**	**Longitudinal Evaluation**	**DB, MMD, RvD, MaS, IDL, DetB**	**AZ (55K)**	**AZ, VS (100K)**	**Walk-forward**

DB = Drebin, MMD = MamaDroid, AZ = AndroZoo, GP = Google Play, VT = VirusTotal, RvD = RevealDroid, DSp = DroidSpan, DSv = DroidSieve, MFl = MudFlow, Afo = Afonso, QJ = Q. Jerome, RMV = RMVDroid, AD = AndroDialysis, BB = BasicBlocks, DD = DroidDet, DDct = DroidDetector, HMM = HMMDetector, ICC = ICCDetector, ALD = ALDroid, ImgD = ImgDroid, MDMC = MDMC, MaS = MalScan, IDL = InterpretDL, DetB = DetectBERT, AGr = APIGraph, EFCG = EFCG, Bai21 = Bai et al. 2021, AMD = AMD, VS = VirusShare, OPSEQS = OPSEQS

we have shown can be misleading. Moreover, they consider a relatively smaller dataset with 2,160 samples per year, to our average of around 8,900 samples per year.

7 Conclusion

In this paper, we investigated how reliable traditionally debiased evaluation is for assessing the capability of ML-based Android malware detectors. We first performed a systematic literature review and characterized the features, representations, models, and evaluation strategies used by ML-based detectors. We then performed a large-scale evaluation of state-of-the-art tools under multiple debiased dataset instances, demonstrating that results can be severely inconsistent depending on the time period under consideration. In fact, we showed that for any given tool, one can find a clean dataset setup for which it outperforms other state-of-the-art tools, highlighting the unreliability of current single-split evaluation setups. In addition, we showed how an intentionally naïve classifier relying on simple package names is able to achieve results comparable to state-of-the-art tools even on debiased datasets, highlighting the need for more advanced dataset cleaning methods for reliable evaluation. We hope our study encourages placing greater emphasis on the quality of evaluated datasets and adequate comparison of different tools under the same setup.

8 Data Availability

We make all our artifacts (detailed literature review categorization, dataset, and evaluation scripts) available online [70] and also in a long term retention platform [69].

Acknowledgments

Part of this work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and National Cybersecurity Consortium (NCC). We also thank Manan Daga for his help in filtering papers for the literature review described in Section 2.

References

[1] n.d. ACM Digital Library. https://dl.acm.org/.
[2] n.d. CORE ranking (Conference Portal). http://portal.core.edu.au/conf-ranks/.
[3] n.d. Google Scholar. https://scholar.google.com/.
[4] n.d. Google Scholar Metrics: Top venues. https://scholar.google.ca/citations?view_op=top_venues&hl=en.
[5] n.d. IEEE Xplore Digital Library. https://ieeexplore.ieee.org/.
[6] n.d. ScienceDirect. https://www.sciencedirect.com/.
[7] n.d. Springer. https://link.springer.com/.
[8] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 2016. Empirical Assessment of Machine Learning-Based Malware Detectors for Android: Measuring the Gap Between In-the-Lab and In-the-Wild Validation Scenarios. *Empirical Software Engineering* (2016), 183–211. doi:10.1007/s10664-014-9352-6
[9] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are Your Training Datasets Yet Relevant? An Investigation Into the Importance of Timeline in Machine Learning-Based Malware Detection. In *Proc. of the International Symposium on Software Reliability Engineering (ISSRE)*. 51–67. doi:10.1007/978-3-319-15618-7_5
[10] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proc. of the International Conference on Mining Software Repositories (MSR)*. 468–471. doi:10.1145/2901739.2903508
[11] Ebtesam J Alqahtani, Rachid Zagrouba, and Abdullah Almuhaideb. 2019. A Survey on Android Malware Detection Techniques using Machine Learning Algorithms. In *Proc. of the International Conference on Software Defined Systems (SDS)*. 110–117. doi:10.1109/sds.2019.8768729
[12] Safa Altaha, Ahmed Aljughiman, and Sonia Gul. 2024. A Survey on Android Malware Detection Techniques Using Supervised Machine Learning. *IEEE Access* (2024), 173168–173191. doi:10.1109/access.2024.3485706
[13] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. 2020. DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices. *Computers & Security* (2020), 101663. doi:10.1016/j.cose.2019.101663
[14] Anzhi. n.d. Anzhi Market. http://www.anzhi.com/ (Website no longer active).
[15] AppChina. n.d. AppChina. https://www.appchina.com/.
[16] Anshul Arora, Sateesh K Peddoju, and Mauro Conti. 2019. PermPair: Android Malware Detection Using Permission Pairs. *IEEE Transactions on Information Forensics and Security* (2019), 1968–1982. doi:10.1109/tifs.2019.2950134
[17] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Proc. of the USENIX Security Symposium (USENIX Security)*. 3971–3988.
[18] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*. 23–26. doi:10.14722/ndss.2014.23247
[19] Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. 2024. A Survey on Malware Detection with Graph Representation Learning. *ACM Computing Surveys (CSUR)* (2024), 1–36. doi:10.1145/3664649
[20] Matteo Brosolo, Vinod Puthuvath, Asmitha Ka, Rafidha Rehiman, and Mauro Conti. 2024. SoK: Visualization-based Malware Detection Techniques. In *Proc. of the International Conference on Availability, Reliability and Security (ARES)*. 1–13. doi:10.1145/3664476.3664514
[21] Haipeng Cai. 2020. Assessing and Improving Malware Detection Sustainability Through App Evolution Studies. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2020), 1–28. doi:10.1145/3371924
[22] Gerardo Canfora, Francesco Mercaldo, and Corrado Aaron Visaggio. 2016. An HMM and Structural Entropy Based Detector for Android Malware: An Empirical Study. *Computers & Security* (2016), 1–18. doi:10.1016/j.cose.2016.04.009
[23] Michael Cao, Khaled Ahmed, and Julia Rubin. 2022. Rotten Apples Spoil the Bunch: An Anatomy of Google Play Malware. In *Proc. of the International Conference on Software Engineering (ICSE)*. 1919–1931. doi:10.1145/3510003.3510161
[24] Yizheng Chen, Zhoujie Ding, and David Wagner. 2023. Continuous Learning for Android Malware Detection. In *Proc. of the USENIX Security Symposium (USENIX Security)*. 1127–1144.
[25] William G. Cochran. 1977. *Sampling Techniques*. John Wiley.
[26] Contagio. 2018. Contagio Dataset. http://contagiominidump.blogspot.com/.
[27] Nadia Daoudi, Kevin Allix, Tegawendé François Bissyandé, and Jacques Klein. 2022. A Deep Dive Inside DREBIN: An Explorative Analysis Beyond Android Malware Detection Scores. *ACM Transactions on Privacy and Security (TOPS)* (2022), 1–28. doi:10.1145/3503463
[28] Fatemeh Deldar and Mahdi Abadi. 2023. Deep Learning for Zero-day Malware Detection and Classification: A Survey. *ACM Computing Surveys (CSUR)* (2023), 1–37. doi:10.1145/3605775
[29] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing* (2017), 711–724. doi:10.1109/tdsc.2017.2700270

[30] Somayyeh Fallah and Amir Jalaly Bidgoly. 2022. Android Malware Detection using Network Traffic based on Sequential Deep Learning Models. *Software: Practice and Experience* (2022), 1987–2004. doi:10.1002/spe.3112

[31] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. 2017. AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Computers & Security* (2017), 121–134. doi:10.1016/j.cose.2016.11.007

[32] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. 2018. A Novel Dynamic Android Malware Detection System With Ensemble Learning. *IEEE Access* (2018), 30996–31011. doi:10.1109/access.2018.2844349

[33] Cuiying Gao, Gaozhun Huang, Heng Li, Bang Wu, Yueming Wu, and Wei Yuan. 2024. A Comprehensive Study of Learning-based Android Malware Detectors Under Challenging Environments. In *Proc. of the International Conference on Software Engineering (ICSE)*. 1–13. doi:10.1145/3597503.3623320

[34] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2018), 1–29. doi:10.1145/3180155.3182551

[35] Xiuting Ge, Yifan Huang, Zhanwei Hui, Xiaojuan Wang, and Xu Cao. 2021. Impact of Datasets on Machine Learning based Methods in Android Malware Detection: an Empirical Study. In *Proc. of the International Conference on Software Quality, Reliability and Security (QRS)*. 81–92. doi:10.1109/qrs54544.2021.00019

[36] Google. n.d. Google Play Store. https://play.google.com/.

[37] Alejandro Guerra-Manzanares. 2024. Machine Learning for Android Malware Detection: Mission Accomplished? A Comprehensive Review of Open Challenges and Future Perspectives. *Computers & Security* (2024), 103654. doi:10.1016/j.cose.2023.103654

[38] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. 2021. KronoDroid: Time-Based Hybrid-Featured Dataset for Effective Android Malware Detection and Characterization. *Computers & Security* (2021), 102399. doi:10.1016/j.cose.2021.102399

[39] Alejandro Guerra-Manzanares, Marcin Luckner, and Hayretdin Bahsi. 2022. Concept Drift and Cross-device Behavior: Challenges and Implications for Effective Android Malware Detection. *Computers & Security* (2022), 102757. doi:10.1016/j.cose.2022.102757

[40] Yiling He, Yiping Liu, Lei Wu, Ziqi Yang, Kui Ren, and Zhan Qin. 2022. MsDroid: Identifying Malicious Snippets for Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing* (2022), 2025–2039. doi:10.1109/tdsc.2022.3168285

[41] Haonan Hu, Yue Liu, Yanjie Zhao, Yonghui Liu, Xiaoyu Sun, Chakkrit Tantithamthavorn, and Li Li. 2023. Detecting Temporal Inconsistency in Biased Datasets for Android Malware Detection. In *Proc. of the International Conference on Automated Software Engineering Workshops (ASEW)*. 17–23. doi:10.1109/asew60602.2023.00007

[42] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. 2021. Towards an Interpretable Deep Learning Model for Mobile Malware Detection and Family Identification. *Computers & Security* (2021), 102198. doi:10.1016/j.cose.2021.102198

[43] Paul Irolla and Alexandre Dey. 2018. The Duplication Issue Within the DREBIN Dataset. *Journal of Computer Virology and Hacking Techniques* (2018), 245–249. doi:10.1007/s11416-018-0316-z

[44] Vijaya Kaza. 2026. Keeping Google Play & Android App Ecosystems Safe in 2025. https://blog.google/security/keeping-google-play-android-app-ecosystem-safe-2025/.

[45] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. 2018. Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In *Proc. of the International Carnahan Conference on Security Technology (ICCST)*. 1–7. doi:10.1109/ccst.2018.8585560

[46] Deqiang Li, Shicheng Cui, Yun Li, Jia Xu, Fu Xiao, and Shouhuai Xu. 2023. PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks. *IEEE Transactions on Dependable and Secure Computing* (2023), 920–936. doi:10.1109/tdsc.2023.3265665

[47] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2021. Arms Race in Adversarial Malware Detection: A Survey. *ACM Computing Surveys (CSUR)* (2021), 1–35. doi:10.1145/3484491

[48] Yao Li, Dawei Yuan, Tao Zhang, Haipeng Cai, David Lo, Cuiyun Gao, Xiapu Luo, and He Jiang. 2024. Meta-Learning for Multi-Family Android Malware Classification. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2024), 1–27. doi:10.1145/3664806

[49] Huawen Liu, Jigui Sun, Lei Liu, and Huijie Zhang. 2009. Feature Selection With Dynamic Mutual Information. *Pattern Recognition* (2009), 1330–1339. doi:10.1016/j.patcog.2008.10.028

[50] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. 2020. A Review of Android Malware Detection Approaches based on Machine Learning. *IEEE Access* (2020), 124579–124607. doi:10.1109/access.2020.3006143

[51] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2022. Deep Learning for Android Malware Defenses: A Systematic Literature Review. *ACM Computing Surveys (CSUR)* (2022), 1–36. doi:10.1145/3544968

[52] Zhijie Liu, Liang Feng Zhang, and Yutian Tang. 2023. Enhancing Malware Detection for Android Apps: Detecting Fine-Granularity Malicious Components. In *Proc. of the International Conference on Automated Software Engineering (ASE)*. 1212–1224. doi:10.1109/ase56229.2023.00074

[53] Aravind Machiry, Nilo Redini, Eric Gustafson, Yanick Fratantonio, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. 2018. Using Loops for Malware Classification Resilient to Feature-Unaware Perturbations. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*. 112–123. doi:10.1145/3274694.3274731

[54] Pascal Maniriho, Abdun Naser Mahmood, and Mohammad Jabed Morshed Chowdhury. 2024. A Survey of Recent Advances in Deep Learning Models for Detecting Malware in Desktop and Mobile Platforms. *ACM Computing Surveys (CSUR)* (2024), 1–41. doi:10.1145/3638240

[55] E Mariconti, L Onwuzurike, P Andriotis, E De Cristofaro, G Ross, and G Stringhini. 2017. MamaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*. 1–15. doi:10.14722/ndss.2017.23353

[56] Oded Maron and Tomás Lozano-Pérez. 1997. A Framework for Multiple-Instance Learning. *Advances in Neural Information Processing Systems* (1997), 570–576.

[57] Borja Molina-Coronado, Usue Mori, Alexander Mendiburu, and José Miguel-Alonso. 2023. Towards a Fair Comparison and Realistic Evaluation Framework of Android Malware Detectors Based on Static Analysis and Machine Learning. *Computers & Security* (2023), 102996. doi:10.1016/j.cose.2022.102996

[58] Ali Muzaffar, Hani Ragab Hassen, Michael A Lones, and Hind Zantout. 2022. An In-Depth Review of Machine Learning Based Android Malware Detection. *Computers & Security* (2022), 102833. doi:10.1016/j.cose.2022.102833

[59] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2019. Mamadroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (extended version). *ACM Transactions on Privacy and Security (TOPS)* (2019), 1–34. doi:10.1145/3313391

[60] Fan Ou and Jian Xu. 2022. S3Feature: A Static Sensitive Subgraph-Based Feature for Android Malware Detection. *Computers & Security* (2022), 102513. doi:10.1016/j.cose.2021.102513

[61] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. 2020. A Systematic Literature Review of Android Malware Detection Using Static Analysis. *IEEE Access* (2020), 116363–116379. doi:10.1109/access.2020.3002842

[62] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification Across Space and Time. In *Proc. of the USENIX Security Symposium (USENIX Security)*. 729–746.

[63] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*. 241–252. doi:10.1145/2382196.2382224

[64] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. 2020. A Survey of Android Malware Detection With Deep Neural Models. *ACM Computing Surveys (CSUR)* (2020), 1–36. doi:10.1145/3417978

[65] Lina Shen, Mengqi Fang, and Jian Xu. 2024. GHGDroid: Global Heterogeneous Graph-Based Android Malware Detection. *Computers & Security* (2024), 103846. doi:10.1016/j.cose.2024.103846

[66] StatCounter. 2026. Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide.

[67] Tiezhu Sun, Nadia Daoudi, Kisub Kim, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2024. DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 420–426. doi:10.1145/3674805.3690745

[68] Tiezhu Sun, Nadia Daoudi, Weiguo Pian, Kisub Kim, Kevin Allix, Tegawende F Bissyande, and Jacques Klein. 2025. Temporal-Incremental Learning for Android Malware Detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2025), 1–30. doi:10.1145/3702990

[69] Michael Tegegn and Julia Rubin. 2026. Artifact for "The Illusion of Success: Learning-Based Android Malware Detectors". https://doi.org/10.5281/zenodo.21067702.

[70] Michael Tegegn and Julia Rubin. 2026. Supplementary Material. https://resess.github.io/artifacts/MalwareDetectionEvaluation/.

[71] VirusTotal. n.d. VirusTotal. https://virustotal.com/.

[72] Huanran Wang, Weizhe Zhang, and Hui He. 2022. You Are What the Permissions Told Me! Android Malware Detection based on Hybrid Tactics. *Journal of Information Security and Applications* (2022), 103159. doi:10.1016/j.jisa.2022.103159

[73] Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. 2022. MalWhiteout: Reducing Label Errors in Android Malware Detection. In *Proc. of the International Conference on Automated Software Engineering (ASE)*. 1–13. doi:10.1145/3551349.3560418

[74] Website. n.d. VirusShare - Online malware repository. https://virusshare.com/.

[75] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *Proc. of the International Conference on Detection of Intrusions and Malware, and Vulnerability*

Assessment (DIMVA). 252–276. doi:10.1007/978-3-319-60876-1_12

[76] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. DroidMat: Android Malware Detection Through Manifest and API Calls Tracing. In *Proc. of the Asia Joint Conference on Information Security (AsiaJCIS)*. 62–69. doi:10.1109/asiajcis.2012.18

[77] Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. 2019. MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. In *Proc. of the International Conference on Automated Software Engineering (ASE)*. 139–150. doi:10.1109/ase.2019.00023

[78] Yueming Wu, Deqing Zou, Wei Yang, Xiang Li, and Hai Jin. 2021. HomDroid: Detecting Android Covert Malware by Social-Network Homophily Analysis. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*. 216–229. doi:10.1145/3460319.3464833

[79] Ning Xi, Yuchen Zhang, Pengbin Feng, Siqi Ma, Jianfeng Ma, Yulong Shen, and Yale Yang. 2024. GNNDroid: Graph-Learning Based Malware Detection for Android Apps With Native Code. *IEEE Transactions on Dependable and Secure Computing* (2024), 1460–1476. doi:10.1109/tdsc.2024.3444913

[80] Peiyu Xiong, Michael Tegegn, Jaskeerat Singh Sarin, Shubhraneel Pal, and Julia Rubin. 2024. It Is All about Data: A Survey on the Effects of Data on Adversarial Robustness. *ACM Computing Surveys (CSUR)* (2024), 1–41. doi:10.1145/3627817

[81] Ke Xu, Yingjiu Li, and Robert H Deng. 2016. ICCDetector: ICC-Based Malware Detection on Android. *IEEE Transactions on Information Forensics and Security* (2016), 1252–1264. doi:10.1109/tifs.2016.2523912

[82] Pooja Yadav, Neeraj Menon, Vinayakumar Ravi, Sowmya Vishvanathan, and Tuan D Pham. 2022. EfficientNet Convolutional Neural Networks-Based Android Malware Detection. *Computers & Security* (2022), 102622. doi:10.1016/j.cose.2022.102622

[83] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. 2014. Droidminer: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications. In *Proc. of the European Symposium on Research in Computer Security (ESORICS)*. 163–182. doi:10.1007/978-3-319-11203-9_10

[84] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. 2015. AppContext: Differentiating Malicious and Benign Mobile App Behaviors using Context. In *Proc. of the International Conference on Software Engineering*. 303–313. doi:10.1109/icse.2015.50

[85] Zhenlong Yuan, Yongqiang Lu, and Yibo Xue. 2016. DroidDetector: Android Malware Characterization and Detection Using Deep Learning. *Tsinghua Science and Technology* (2016), 114–123. doi:10.1109/tst.2016.7399288

[86] Xueying Zeng, Youquan Xian, Sihao Liu, Xudong Mou, Yanze Li, Lei Cui, and Bo Li. 2026. MARD: A Multi-Agent Framework for Robust Android Malware Detection. *arXiv* (2026). doi:10.48550/arXiv.2604.25264

[87] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*. 1105–1116. doi:10.1145/2660267.2660359

[88] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-Art Classifiers With API Semantics to Detect Evolved Android Malware. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*. 757–770. doi:10.1145/3372297.3417291

[89] Yanjie Zhao, Li Li, Haoyu Wang, Haipeng Cai, Tegawendé F. Bissyandé, Jacques Klein, and John C. Grundy. 2021. On the Impact of Sample Duplication in Machine-Learning-Based Android Malware Detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2021), 40:1–40:38. doi:10.1145/3446905

[90] Jingnan Zheng, Jiahao Liu, An Zhang, Jun Zeng, Ziqi Yang, Zhenkai Liang, and Tat-Seng Chua. 2024. MaskDroid: Robust Android Malware Detection With Masked Graph Representations. In *Proc. of the International Conference on Automated Software Engineering (ASE)*. 331–343. doi:10.1145/3691620.3695008

[91] Yuyang Zhou, Guang Cheng, Shui Yu, Zongyao Chen, and Yujia Hu. 2024. MTDroid: A Moving Target Defense-Based Android Malware Detector Against Evasion Attacks. *IEEE Transactions on Information Forensics and Security* (2024), 6377–6392. doi:10.1109/tifs.2024.3414339

[92] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*. 95–109. doi:10.1109/sp.2012.16

[93] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *Proc. of the USENIX Security Symposium (USENIX Security)*. 2361–2378.

[94] Deqing Zou, Yueming Wu, Siru Yang, Anki Chauhan, Wei Yang, Jiangying Zhong, Shihan Dou, and Hai Jin. 2021. IntDroid: Android Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–32. doi:10.1145/3442588

Received 2026-01-30; accepted 2026-04-16