Beyond the TESSERACT: Trustworthy Dataset Curation for Sound Evaluations of Android Malware Classifiers

Theo Chow[†‡], Mario D'Onghia[‡], Lorenz Linhardt[⋆∘], Zeliang Kan[†⋄],
Daniel Arp[§], Lorenzo Cavallaro[‡], Fabio Pierazzi[‡]
[†]King's College London, [⋆]Technische Universität Berlin,
[∘]BIFOLD – Berlin Institute for the Foundations of Learning and Data, [⋄]HiddenLayer,
[§]Technische Universität Wien, [‡]University College London

Abstract—**The reliability of machine learning critically depends on dataset quality. While machine learning applied to computer vision and natural language processing benefits from high-quality benchmark datasets, cyber security often falls behind, as quality ties to the ability of accessing hard-to-obtain realistic data that may evolve over time. Android is, however, positioned uniquely in this ecosystem due to AndroZoo and other sources, which provide large-scale, continuously updated, and timestamped repositories of benign and malicious apps.**

Since their release, such data sources provided access to populations of Android apps that researchers can sample from to evaluate learning-based methods in realistic settings, i.e., over temporal frames to account for apps evolution (natural distribution shift) and test datasets that reflect in-the-wild class ratios. Surprisingly, we observe that despite this abundance of data, performance discrepancies of learning-based Android malware classifiers still persist even after satisfying such realistic requirements, which challenges our ability to understand what the state-of-the-art in this field is. In this work, we identify five novel factors that influence such discrepancies: we show how such factors have been largely overlooked and the impact they have on providing sound evaluations. Our findings and recommendations help define a methodology for creating trustworthy datasets towards sound evaluations of Android malware classifiers.

I. INTRODUCTION

The foundation of trustworthy Machine Learning (ML) research lies in the datasets used for evaluation. To ensure that results generalize beyond controlled lab-only settings, datasets must faithfully capture the distributions, dynamics, and challenges of the real world. However, unlike computer vision and natural language processing, where community benchmarks such as ImageNet [29] and MMLU [48] (or datasets in repositories such as UCI [6] and Kaggle [4]) provide stable and widely accepted testbeds, the Android malware domain lacks reliable and standardized benchmarks. This absence is primarily due to both restricted access to real-world attack data [83] and the highly non-stationary nature of adversarial environments [78], in which attacker strategies evolve continuously, hindering the curation of static benchmarks. The introduction of large-scale collections of benign or malicious Android Application Packages (APKs) such as AndroZoo [14] and VirusShare [7] has partially addressed this gap by enabling researchers to curate time-aware Android malware datasets in accordance to the best practices and guidelines introduced in TESSERACT [78] and other works on recommendations to apply ML to security [17], [36], [83]. However, we show that these measures alone are insufficient to guarantee reliable and unbiased assessments.

Let us consider two state-of-the-art Android malware datasets that have been adopted in top-tier security venues: *APIGraph* [97] and *Transcendent* [19]. Both were explicitly constructed to adhere to the spatio-temporal constraints proposed by TESSERACT [78] and reflect recent trustworthy AI guidelines for dataset design and fair evaluations [17], [36]. Despite this, evaluation across five state-of-the-art malware detectors reveals striking discrepancies: Figure 1 shows that detection models consistently achieve higher F_1-Score scores on *APIGraph* than on *Transcendent*, even when evaluated on the same time frame (2014–2018). From an ML perspective, this indicates that sources of dataset bias persist beyond temporal or class-ratio considerations, and that hidden shifts or artifacts may systematically affect evaluation outcomes. These observations highlight the need for a deeper investigation into the nuanced factors that must be addressed to curate truly trustworthy datasets in security domains.

More specifically, we identify five new spatio-temporal factors that can affect dataset sampling and evaluation of a learning-based Android malware detector. After introducing our hypotheses, we empirically evaluate the impact of each factor using five representative state-of-the-art Android malware classifiers: DREBIN [18], DEEPDREBIN [43], MALSCAN [94], RAMDA [60], and HCC [25]. For each factor, we provide practical recommendations to reduce spatial and temporal biases. Finally, we survey existing Android malware datasets used in the literature to evidence how the factors we identify have often been ignored or overlooked in previous work. In fact, we discover that 95% of the datasets we surveyed violate 3 or more of our recommendations, which can lead to biased experimental results.

In summary, we make the following contributions:

- We discover five novel spatio-temporal bias factors (§II) beyond TESSERACT [78] and existing literature [17], [36], [83], which can lead to misleading results even if following all current recommendations for realistic evaluations, as shown in Figure 1.

- We systematically analyze the impact of these factors on five state-of-the-art Android classifiers and two Android malware datasets that reflect current best practices. Where necessary, we augment our analysis by sampling additional data from AndroZoo to further generalize our findings beyond the *Transcendent* and *APIGraph* datasets.
- We present actionable recommendations to address each identified factor including an evaluation metric (§V) and sampling strategy (§VIII) for evaluating and curating trustworthy Android malware dataset.
- We conduct a prevalence study of the identified bias factors for popular Android malware datasets, highlighting that these are often overlooked (§IX).

To ensure reproducibility, and to foster bias-free evaluations in Android malware classification, we release our code and data publicly at https://github.com/s2labres/hypercube-ml, including STAS and HYPERCUBE.

II. PRELIMINARY OBSERVATIONS

In this section, we investigate the possible causes behind the striking performance difference observed in Figure 1. We reinforce that this occurs regardless of the feature representation or machine learning classifier used and that the two datasets, *APIGraph* [97] (D_A) and *Transcendent* [19] (D_T), adhere to all spatial and temporal constraints outlined in TESSERACT [78] and align with recommendations from Arp et al. [17]. Moreover, they overlap in 2014–2018; hence, we restrict D_A to this period to match the time frame of D_T, which allows for a direct comparison between the two datasets.

Family Overlap. Prior work [26] showed that the appearance of new malware families is a major cause of drift in D_T. Hence, we investigate the distribution differences between the two datasets by computing the "family overlap," i.e., the percentage of malware samples in the test set (from 2015 to 2018) belonging to families that existed in the training set (2014). A more formal definition of family overlap can be found in Appendix §C. Figure 2 reports the family overlap, which shows that new malware families steadily replace older ones in D_T, while malware family overlap does not show a clear trend in D_A. Considering that both datasets were sampled within the same time frame, we postulate that certain factors differed during the curation of the datasets, resulting in different distributions of malware samples.

Novel Spatio-Temporal Bias Factors. To identify possible sources of bias explaining the performance difference in Figure 1, we examine the high-level characteristics of D_A and D_T in Table I. We notice three major differences (Timestamp Types, VT Thresholds, and App Markets) and postulate that these may have contributed to the observed performance difference.

Specifically, *App Markets* distribution affects the sampling source of both benign and malicious APKs (§VI), while a higher VirusTotal Threshold (VTT) may lead to malware that is "easier to detect" (§VII). Similarly, *Timestamp Types* can influence both sampling and evaluation by inducing dissimilar temporal distribution of APKs (§IV).

Table I: High-level differences between D_A and D_T.

Characteristic	D_A [97]	D_T [19]
Time Frame	2012-2018	2014-2018
Timestamp	*VT_first_dates*	AZ *dex_dates*
VT Threshold	15	4
App Markets	GooglePlay (goodware), VirusShare (malware)	GooglePlay (91%), Anzhi (7%), AppChina (2%)
Dataset Size	320,001	259,230
Dataset Size (2014-18)	241,611	259,230
Num. Families	484	492

Although *Dataset Size* is similar between D_A and D_T, neither work clearly motivates its chosen size; in fact, none of the datasets surveyed in our prevalence analysis (§IX) provide an explicit justification. We therefore study how dataset size may influence the statistical representativeness of a dataset (§VIII).

Finally, despite sharing the same *Time Frame*, we argue that existing metrics such as Area Under Time (AUT) [78] capture only a temporal snapshot of performance and its possible that one may get "lucky" with the training and testing split, resulting in a misleading performance for a given timeframe—which we refer to as *Temporal Luck* (§V).

In the remainder of this paper, we analyze each factor by formulating hypothesis and evaluating their impact on both distribution and detection performance. From our findings, we determine whether each factor introduced biases during the curation and evaluation of D_A and D_T. We then provide practical recommendations on how to prevent these bias factors. We further quantify each factor's prevalence in the scientific community by surveying Android malware datasets in previous work and assessing them with respect to our recommendations (§IX).

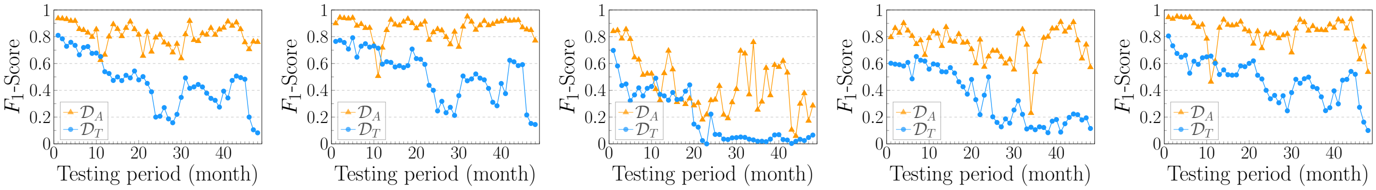
III. EXPERIMENTAL SETTINGS

Before introducing and analyzing the impact of the factors we identify, we provide a brief overview of the datasets, classifiers, and metrics used in the remainder of the paper.

Datasets. To investigate the five identified factors, we use D_A [97] and D_T [19] (see §II). Where relevant, we generalize our findings using metadata from AndroZoo [1], reports from VirusTotal [8], and family labels unified with Euphony [49]. We sample across multiple markets, recreate both D_A and D_T with different VTTs at different time points, and curate datasets with varying strategies for determining sampling size. Further details are provided in §VI, §VII, §VIII.

Classifiers. All our experiments are carried out on five representative Android malware classifiers:

- **DREBIN** [18]: Linear SVM using binary features from static analysis (e.g., APIs, URLs, Activities).
- **DEEPDREBIN** [43]: MLP using the DREBIN features.



(a) DREBIN [18] (b) DEEPDREBIN [43] (c) MALSCAN [94] (d) RAMDA [60] (e) HCC [25]

Figure 1: **Motivational example.** Performance of five SotA classifiers on two datasets from the same time frame, with the F_1-Scores on D_A [97] constantly higher than the ones on D_T [19]. This paper investigates dataset bias factors and flawed evaluation strategies that may cause this discrepancy.

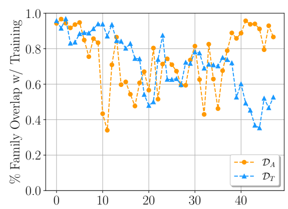


Figure 2: Monthly family overlap of D_A and D_T at test time (2015–2018) with respect to families in the training set (2014).

- **MALSCAN** [94]: Graph-based approach using degree centrality with a Random Forest classifier.
- **RAMDA** [60]: DNN combining a VAE and an MLP, focusing on sensitive APIs, Permissions, and Intents.
- **HCC** [25]: Encoder-guided embedding using malware family labels for enhanced class separability.

Appendix B provides more details on these classifiers. Notice that this heterogeneity allows for robust and representation-agnostic insights. Since HCC shows similar trends yet consistently outperforms the other classifiers, we show performance plots only for HCC throughout the remainder of the paper but include all the other performance figures in the Appendix.

Metrics. We assess each factor's impact on classification performance using the standard performance metrics F_1-Score and *AUT* for temporal evaluation [78]. Additionally, we report the *Family Overlap* measure introduced in §II to quantify the distribution shift of malware families independent of the feature representation or classifier used (see Appendix §C).

IV. FACTOR 1: TIMESTAMP TYPES

A. Hypothesis

Since TESSERACT's advocacy for time-aware evaluation [13], [75], [78], the integration of temporal information has become an essential consideration in constructing realistic datasets. In particular, we note that timestamps do not just impact the temporal order of samples but the sampling process itself. To illustrate this: for example, in D_A, samples with VirusTotal First-Submission Dates (*VT_first_dates*) before 2012 and after 2018 were discarded; and in D_T, the AndroZoo population was filtered by excluding samples with *dex_date* greater than 2018 or smaller than 2014. As such timestamp-based filtering directly shapes the resulting dataset population, we analyze the impact how commonly used timestamp types affect the construction of time-based datasets.

For the Android domain, several timestamps are available to practitioners. Among the commonly-used ones, *VT_first_dates* (D_A) indicate when APKs were first submitted to VirusTotal [8], and *dex_dates* correspond to the last modification date of an APK's *classes.dex* file (the executable code of the Android app). Other relevant timestamps include GooglePlay *upload_dates*, namely the date in which an APK was uploaded to GooglePlay, and AndroZoo *crawl_dates*, which indicate the date on which AndroZoo crawled an APK from a market.

All these timestamps provide different temporal information and can be loosely sorted into three categories: *Creation Timestamps*, *Publication Timestamps*, and *Third-party Timestamps*. Creation Timestamps aim to capture the date an APK was created or built. Among the notable timestamps, *dex_dates* are the only one falling in this category. *dex_dates* are known to be unreliable [62] as they are easily modifiable by attackers and removed by default by all Android SDKs released after 2016 [3]. Publication timestamps capture the date in which an APK was uploaded to a public market such as GooglePlay. Since they are market-specific, APKs uploaded to multiple markets will have multiple publication timestamps. Finally, Third-party Timestamps capture the date an APK was included in an app repository managed by professionals/academics, such as AndroZoo and VirusTotal.

Given that timestamps provide different temporal information, it is important to understand the distributions they model. Creation timestamps reflect the evolution of malware families and attack techniques over time but may misrepresent reality since republished malware would be assigned to an earlier point despite reappearing in the present. We obtained confirmation from industrial partners that they do see old malware still circulating years after their original appearance. Publication timestamps instead capture the population of an app market at a specific moment, closely matching the deployment setting of antivirus engines. Third-party Timestamps are less reliable indicators of real-world distributions, as samples can be uploaded before or after market release. Nevertheless, they may be useful proxies when other timestamps are unavailable.

In this section, we first evaluate the ability of Creation Timestamps to model the evolution of malware, knowing that

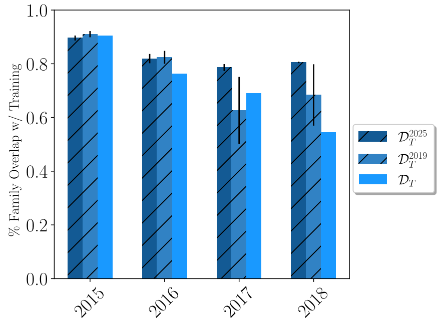


Figure 3: Yearly family overlap of 3 datasets identically and independently sampled from a snapshot of AndroZoo in 2025 and in 2019, using D_T sampling parameters.

old malware may appear at a later time.

Hypothesis 1: Timestamp Type

Due to old malware reappearing, using Creation Timestamps to sample a time-aware dataset in the time frame $[T_1, T_2]$ ($T_1 < T_2$) may lead to different malware distributions, depending on *when* the sampling is performed.

The intuition behind this hypothesis is that as old malware reappears the underlying population may change from the one observed in the past.

We then evaluate the ability of Third-party Timestamps to emulate Publication Timestamps. We would expect APKs to enter third-party services such as AndroZoo and VirusTotal (VT) after their publication on an official app market. In particular, we investigate whether the temporal distribution of a third-party service and a market may be similar but misaligned, which would still allow the Third-party Timestamps to fairly represent that market's distribution.

Hypothesis 2: Timestamp Type

Third-Party Timestamps may fail to reflect the temporal distribution of an app market.

B. Impact

Hypothesis 1. To validate this hypothesis, we first resample D_T in May 2025 three times. We then leverage AndroZoo *crawl_dates* to reconstruct a snapshot of AndroZoo when D_T was sampled in 2019, by filtering out samples that were not present on AndroZoo at that time. For both configurations, we independently sample three datasets, maintaining the same number of malware per month as D_T, using *dex_dates* for sampling and ordering, and VTT=4. We analyze the resulting malware distributions for the two configurations by averaging the family overlap of each. Further details regarding how we recreated D_T can be found in Appendix §A.

Figure 3 reports the family overlap for different samplings. It can be observed that the datasets sampled from AndroZoo in May 2025 (D_T^{2025}) show a higher family overlap in 2017 and 2018, with a significantly smaller standard deviation. In contrast, datasets sampled from a snapshot of AndroZoo when D_T was originally sampled (D_T^{2019}) present a lower average family overlap in the last two years, with a higher standard deviation, indicating greater variability.

These findings confirm our first hypothesis: sampling a dataset from a fixed time frame through Creation Timestamps will yield different malware distributions depending on *when* the sampling is performed (in this case, 2019 vs. 2025). Moreover, this further highlights that old malware may reappear in a population at a later time than its original creation, making the task of modeling the evolution of malware variants through simple Creation Timestamps inaccurate.

Hypothesis 2. To validate our second hypothesis, we compare the temporal distribution of GooglePlay Publication Timestamps (*upload_dates*) with Third-party Timestamps from AndroZoo (*crawl_dates*) and VirusTotal (*VT_first_dates*). We model this distribution as a binary matrix of shape $(N_T, |D_T|)$, where N_T is the number of time units within the time frame $T = [T_1, T_2]$ (e.g., 12 months from 1-1-2021 to 31-12-2021), and $|D_T|$ is the number of samples published on GooglePlay in $[T_1, T_2]$. Each entry (i, j) equals 1 if sample j was published on the market or uploaded to a third-party service during the i^{th} time slot, 0 otherwise. Each column in the matrix represents an individual APK and can sum to 0 or 1. For a third-party service, a column can sum to 0 if the corresponding APK was uploaded outside of T.

We reuse the metadata of 430k APKs that we collected for the experiments in §G. These APK were published by GooglePlay between 1-1-2021 and 31-12-2023 and are available on AndroZoo. Using *upload_dates*, *VT_first_dates*, and *crawl_dates*, we construct yearly temporal distributions for GooglePlay, VT, and AndroZoo, respectively. To assess the similarity of the three temporal distributions, we compute the *cosine similarity* between the GooglePlay distribution and those of VT and AndroZoo. Table II reports the monthly cosine similarities with their standard deviations. The results show that the monthly mean values of AndroZoo *crawl_dates* are closer to 1 in 2021 and 2023, while those of VirusTotal *VT_first_dates* are closer to 0 for all three years, indicating dissimilarity from GooglePlay *upload_dates*.

However, the temporal distributions of VT and AndroZoo may be misaligned with respect to GooglePlay, as APKs are expected to enter these services only after publication on GooglePlay. To verify this, we align each pair of temporal distributions by solving the Dynamic Time Warping (DTW) problem [20] and report the final cumulative distance (normalized as in [41]) in Table III. For reference, we also compute the DTW distance between the GooglePlay distribution and a random one, constructed by assigning each APK a random timestamp succeeding its original *upload_date*. Both VT and AndroZoo distances are close to those of a random distribu-

Table II: **Monthly mean cosine similarity between GooglePlay *upload_dates* and two timestamp distributions: VirusTotal *VT_first_dates* and AndroZoo *crawl_dates*.** 1 indicates identical and -1 indicates inversely-correlated vectors. *crawl_dates* are closer to 1 compared to *VT_first_dates*, suggesting *VT_first_dates* are more dissimilar to *upload_dates*.

	Goodware		**Malware**	
	VT_first_dates	*crawl_dates*	*VT_first_dates*	*crawl_dates*
2021	0.20±0.09	0.76±0.04	0.23±0.10	0.71±0.06
2022	0.13±0.07	0.24±0.05	0.14±0.07	0.67±0.08
2023	0.18±0.06	0.63±0.13	0.21±0.08	0.58±0.13

Table III: **DTW distance between GooglePlay *upload_dates* and three timestamp distribution: VirusTotal *VT_first_dates*, AndroZoo *crawl_dates*, and random temporal distributions.** Values in brackets are normalized against the random distribution, with lower values indicating the time-series aligns better against *upload_dates*.

	Goodware			**Malware**		
	Random	*VT_first_dates*	*crawl_dates*	Random	*VT_first_dates*	*crawl_dates*
2021	116 (1.00)	94 (0.81)	71 (0.61)	52 (1.00)	44 (0.85)	34 (0.65)
2022	111 (1.00)	93 (0.84)	68 (0.61)	46 (1.00)	45 (0.98)	35 (0.76)
2023	100 (1.00)	90 (0.90)	78 (0.78)	38 (1.00)	38 (1.00)	34 (0.90)

tion. Notably, the VT distance for malware is identical to that of a random distribution in 2023 and nearly identical in 2022. These results indicate that Third-party Timestamps cannot reliably represent the temporal distribution of GooglePlay.

C. Recommendations

Recommendation: Timestamp Type

Use Publication Timestamps to model the population of a market (e.g., GooglePlay *upload_dates*).

V. Factor 2: Temporal Luck

A. Hypothesis

Two datasets, D_1 and D_2, sampled from different time frames (e.g., 2012–2014 and 2016–2018), can be expected to represent different distributions (e.g., they may vary in terms of family composition). Therefore, no meaningful conclusions can be drawn from comparing the performance of a malware classifier on D_1 with the performance of another malware detection method evaluated on D_2. This issue of comparing malware detection methods on different datasets presenting a distribution mismatch may arise when researchers directly compare their results with those reported in other related papers ("comparison on paper").

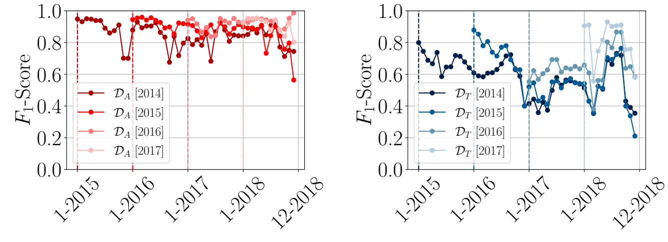
Given two malware detection approaches A_1 and A_2, a straightforward solution would be to evaluate both on the *same* dataset, as done in recent work (e.g., [19], [94], [97]). Nevertheless, this may still not suffice to conclude which method performs best, as the reported results may still be influenced by what we name *Temporal Luck*. This phenomenon arises when an arbitrary time-aware split into training and testing data inflates the performance of a given method, due to the training set being "exceptionally" good or the subsequent testing period not drifting significantly. In practical terms, given a dataset $\mathcal{D}_{2012-2014}$, A_1 may perform better than A_2 when training on 2012 and testing on 2013, whereas A_2 may exhibit superior performance when training on 2013 and testing on 2014. Hence, we postulate the following hypothesis.

Hypothesis: Temporal Luck

Given a dataset, the performance of a malware detection method may vary based on the temporal train/test splits.

B. Impact

Figure 4 provides an overview of the impact of Temporal Luck on both D_A and D_T. We employ the following settings: we train classifiers on data from each year in 2014–2018 and test them in the following years. In practice, we divide the



(a) D_A (b) D_T

Figure 4: **Impact of Temporal Luck on the performance of Hcc.** Each line indicates the F_1-Score of a model trained on each year between 2014 and 2017. It can be seen that training on different years within the same dataset can yield different performance profiles. The issue is evident on D_T, but also present in D_A. For other classifiers, refer to Appendix §E.

Table IV: **Temporal Luck.** 12-month window Area Under Time (AUT): 2014-2015 indicates that the model is trained on 2014 and its AUT value computed on 2015. The performance of a model on a given dataset changes depending on how we temporally split the dataset.

Classifier	2014-2015	2015-2016	2016-2017	2017-2018	μ_{AUT}	σ_{AUT}
Drebin	0.85	0.91	0.90	0.87	0.88	0.02
DeepDrebin	0.84	0.93	0.91	0.91	0.90	0.03
MalScan	0.67	0.65	0.66	0.87	0.72	0.09
Ramda	0.75	0.81	0.77	0.59	0.73	0.09
Hcc	**0.89**	**0.93**	**0.91**	**0.92**	**0.91**	**0.01**

(a) D_A

Classifier	2014-2015	2015-2016	2016-2017	2017-2018	μ_{AUT}	σ_{AUT}
Drebin	**0.71**	**0.75**	**0.63**	0.80	**0.72**	**0.06**
DeepDrebin	0.69	0.74	0.55	0.82	0.70	0.10
MalScan	0.45	0.42	0.53	**0.83**	0.56	0.16
Ramda	0.54	0.56	0.44	0.74	0.57	0.11
Hcc	0.70	**0.75**	**0.63**	0.80	**0.72**	**0.06**

(b) D_T

datasets into four subsets: the first having 2014 as training window and testing on 2015–2018, the second having 2015 for training and 2016–2018 for testing, and so on. It is visually evident in D_T that significantly different performance profiles may be obtained depending on the temporal split. This can also be observed for D_A, albeit to a smaller extent.

To quantify Temporal Luck in D_A and D_T, we compute the AUT when training on one year and testing only on the following one (i.e., (i) training on 2014 and testing on 2015, (ii) training on 2015 and testing on 2016, etc.). The AUT summarizes the performance of a classifier over a certain time frame, and in the presence of Temporal Luck, it can emphasize when a proposed approach works consistently across different temporal splits. The results summarized in Table IV show again significantly different results on D_T for all classifiers and temporal configurations, and smaller differences for D_A.

The AUT can reveal substantial performance variation across training years. For example, all classifiers show a significantly lower performance in 2016–2017 than in 2017–2018 of D_T. However, the AUT may complicate the comparison between classifiers, as no individual one may perform better than the others in all scenarios (with the exception of HCC for D_A). For example, MALSCAN performs better than both DREBIN and HCC in 2017–2018 of D_T. However, the reverse is true for the other three temporal configurations. Similarly, DEEPDREBIN performs better than DREBIN in 2015–2016, 2016–2017, and 2017–2018, but worse in 2014–2015 of D_A.

To facilitate direct comparison between different malware detection methods, we propose the following extension of the AUT metric, providing a compact summary of the time-aware performance of a given classifier (given a metric f, e.g., F_1-Score), while accounting for Temporal Luck:

$$\text{A-AUT}_f(\mathcal{C}, T, E) = \left(\mu_{AUT_f}, \sigma_{AUT_f}\right) \tag{1}$$

where μ_{AUT_f} and σ_{AUT_f} are the mean and standard deviation of $\text{AUT}_f(C_{t_i}, e_i)$, computed by training classifier $\mathcal{C}$ on each $t_i \in T$ and evaluating on the corresponding $e_i \in E$, with:

$$T := \{D_{in_T:(i+1)n_T-1}\}_{i=0}^{k-1} \tag{2}$$
$$E := \{D_{(i+1)n_T:(i+1)n_T+n_E-1}\}_{i=0}^{k-1} \tag{3}$$
$$k = \lfloor (|D| - n_E)/n_T \rfloor \tag{4}$$

More specifically, a dataset $\mathcal{D}$ is divided into k temporally successive training sets (T) as well as into k evaluation sets (E), having sizes n_T and n_E, respectively, with $n_E \geq n_T$. The formula for the case $n_T > n_E$ is described in Appendix §E. For each dataset pair $(t_i, e_i) \in \{(t_0, e_0), \cdots, (t_{|D|/n_T}, e_{|D|/n_T})\}$, with $t_i \in T$ and $e_i \in E$, we train the classifier $\mathcal{C}$ on t_i and compute its AUT on e_i. Notice that each e_i starts at time unit in_T and includes the following n_E time units. A practical example of this *rolling window* approach is provided in §E.

In Table IV, we report the A-AUT of the F_1-Score computed for our considered classifiers and datasets, setting both n_T and n_E to one year. The proposed metric facilitates the direct comparison of different malware detection methods, while accounting for Temporal Luck. For instance, it can help determine which classifier between DREBIN and DEEPDREBIN on average performs the best on D_T: DREBIN shows a slightly better performance than DEEPDREBIN ($\mu_{AUT} = 0.72$ versus $\mu_{AUT} = 0.70$) while also exhibiting more stable performance across different temporal splits ($\sigma_{AUT} = 0.06$ versus $\sigma_{AUT} = 0.10$). MALSCAN shows weaker performance than both, with higher variability. HCC achieves the highest μ_{AUT} and lowest σ_{AUT} for D_A. For D_T, HCC and DREBIN share the best performance with $\mu_{AUT} = 0.72$ and $\sigma_{AUT} = 0.06$.

C. Recommendation

As a single train-test split may unfairly (dis)advantage one classifier over another, we recommend using the proposed Average Area Under Time (A-AUT) metric to provide a more robust mean estimate of the classifier performance, along with a measure of performance stability over time splits.

Recommendation: Temporal Luck

You **must** use A-AUT as a metric to compare malware detection methods. If multiple methods have comparable A-AUTs, you **must** provide an AUT breakdown. You **may** use A-AUT with $N = 12$ months, to have a yearly summary of the results.

VI. FACTOR 3: APP MARKETS

A. Hypothesis

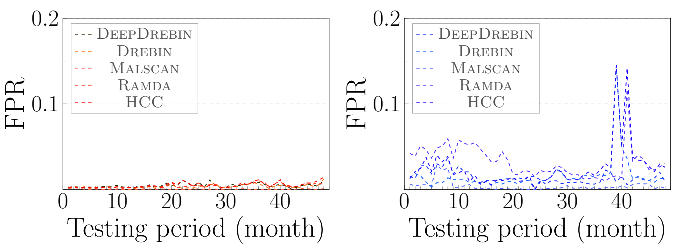
Android apps are distributed across multiple markets. For example, GooglePlay is the dominant platform in Western regions, while other markets are more prevalent in Eastern contexts. When constructing Android malware datasets, recent studies [19], [25], [78], [97] adopt multi-market sampling to increase data diversity and scale. However, this approach may introduce artifacts during training, and classifiers may learn market-related features instead of security-relevant ones [17].

Table V shows the market composition of AndroZoo, D_A, and D_T, based on metadata from AndroZoo [1]. The data reveal a substantial imbalance: certain markets contribute disproportionately to the AndroZoo population, and D_A and D_T differ distinctly in their sampling. In particular, D_A exhibits a pronounced sampling bias, sourcing malware primarily from VirusShare and goodware from GooglePlay.

We therefore hypothesize that an imbalanced market composition in the training data, where malware and goodware are predominantly sampled from different markets, results in statistically-significant performance degradation when evaluated on market-balanced or differently-composed test sets.

Table V: **Comparison of market sources for the APKs within AndroZoo, D_A and D_T.** The market information of each APK is taken from the AndroZoo metadata (note that each APK can be present in multiple markets).

Market	AndroZoo		D_A		D_T	
	Goodware	Malware	Goodware	Malware	Goodware	Malware
angeeks	0.18%	0.64%	0.21%	0.34%	-	-
anzhi	1.64%	**29.25%**	0.37%	1.89%	**3.24%**	**37.57%**
apk_bang	-	-	-	0.01%	-	-
appchina	**2.57%**	**22.74%**	**1.75%**	3.75%	2.28%	**11.28%**
fdroid	0.27%	0.01%	-	-	0.08%	-
freewarelovers	0.02%	-	0.03%	-	-	-
genome	-	0.06%	-	0.57%	-	-
hiapk	0.01%	0.04%	0.01%	0.01%	0.02%	0.06%
mi.com	0.13%	2.82%	0.06%	0.23%	0.4%	4.12%
PlayDrone	**5.75%**	6.41%	**33.52%**	9.75%	**13.26%**	6.35%
GooglePlay	**94.91%**	**29.50%**	**99.99%**	**10.13%**	**95.33%**	**49.29%**
praguard	-	0.47%	-	0.01%	-	-
proandroid	0.02%	0.01%	0.06%	0.02%	-	-
slideme	0.21%	0.27%	0.3%	0.14%	0.02%	-
unknown	0.01%	1.27%	0.08%	**23.74%**	0.02%	0.49%
VirusShare	0.12%	15.95%	0.07%	**81.93%**	0.14%	2.05%
1mobile	0.2%	0.48%	1.36%	0.35%	-	-



(a) D_A (b) D_T

Figure 5: FPR of the five classifiers trained on 2014 and evaluated across 2015–2018.

Hypothesis: App Markets

Altering the composition of App Markets used in dataset sampling significantly affects goodware and malware class-distribution, and leads to reduced detection performance (e.g., F_1-Score or AUT) when models are evaluated on out-of-sample market distributions.

B. Impact

To investigate whether the performance on D_A is influenced by goodware being sampled entirely from GooglePlay and malware primarily from VirusShare, we first analyze the False Positive Rate (FPR) of both datasets. As expected, classifiers trained on D_A tend to be steadily confident in predicting benign APKs. In contrast, classifiers trained on D_T exhibit a much more variable FPR. Figure 5 shows the FPR trend of all five classifiers when trained and evaluated on both datasets.

For a more systematic analysis, we now empirically evaluate the impact of market composition by constructing six dataset configurations, designed to simulate the influence of market sources on dataset composition, and evaluating their effect on malware detection performance. We sample all these datasets from AndroZoo; in particular, from two groups of markets. We collect samples only from GooglePlay (*GP*) as it is the majority market in AndroZoo, and we group all remaining 3rd-Party Markets together (*3PM*), as 3PM are otherwise individually too small (compared to GP). Overall, we collected 15,000 goodware and 15,000 malware samples for each GooglePlay and 3PM, summing up to 60,000 samples. A detailed breakdown of the configurations is given in Appendix §A. Each dataset configuration consists of 20,000 samples for training and 5,000 samples for testing, randomly selected from the collected pool of samples. Due to the limited number of samples per time split, we do not perform a temporal evaluation and instead sample three 3 datasets per configuration and report the average F_1-Score.

Table VI: **Evaluation results of sampling from diverse app market sources.** Training and testing on the same markets consistently yield higher F_1-Score compared to training and testing on different markets, showing APKs from multiple markets are inherently different. Mixing markets can lead to unclear evaluation results skewed towards the majority market. Combining malware and goodware from separate markets inflate performance, corroborating findings in [17]. Standard deviation for all results is lower than 0.03.

No.	**Train Set**	**Test Set**	**Drebin** F_1-Score	**DeepDrebin** F_1-Score	**MalScan** F_1-Score	**Ramda** F_1-Score	**Hcc** F_1-Score
1	$\mathcal{D}_{GP}$	$\mathcal{D}_{GP}$	0.776	0.788	0.841	0.524	0.813
2	$\mathcal{D}_{3PM}$	$\mathcal{D}_{3PM}$	0.658	0.699	0.614	0.514	0.782
3	$\mathcal{D}_{GP}$	$\mathcal{D}_{3PM}$	0.265	0.247	0.298	0.227	0.261
4	$\mathcal{D}_{3PM}$	$\mathcal{D}_{GP}$	0.415	0.290	0.416	0.262	0.348
5	$\mathcal{D}_{GP}$	$\mathcal{D}_{EVEN}$	0.422	0.395	0.511	0.344	0.417
6	$\mathcal{D}_{3PM}$	$\mathcal{D}_{EVEN}$	0.622	0.558	0.581	0.454	0.642
7	$\mathcal{D}_{EVEN}$	$\mathcal{D}_{GP}$	0.740	0.747	0.789	0.576	0.799
8	$\mathcal{D}_{EVEN}$	$\mathcal{D}_{3PM}$	0.659	0.566	0.666	0.467	0.781
9	$\mathcal{D}_{EVEN}$	$\mathcal{D}_{EVEN}$	0.659	0.706	0.694	0.504	0.775
10	$\mathcal{D}_{PROP}$	$\mathcal{D}_{GP}$	0.774	0.798	0.836	0.595	0.829
11	$\mathcal{D}_{PROP}$	$\mathcal{D}_{3PM}$	0.573	0.586	0.625	0.373	0.706
12	$\mathcal{D}_{PROP}$	$\mathcal{D}_{PROP}$	0.712	0.763	0.761	0.517	0.796
13	$\mathcal{D}_{GP3PM}$	$\mathcal{D}_{GP3PM}$	0.899	0.904	0.814	0.844	0.936
14	$\mathcal{D}_{GP3PM}$	$\mathcal{D}_{3PMGP}$	0.082	0.067	0.080	0.061	0.076
15	$\mathcal{D}_{3PMGP}$	$\mathcal{D}_{GP3PM}$	0.142	0.116	0.140	0.074	0.107
16	$\mathcal{D}_{3PMGP}$	$\mathcal{D}_{3PMGP}$	0.829	0.856	0.851	0.626	0.910

Similarity of APKs. We begin by analyzing the impact on performance when training and testing on either the same or different markets. As shown in Table VI, training and testing on the same market (rows 1–2) consistently yields higher F_1-Score scores compared to cross-market scenarios (rows 3–4). These results indicate notable differences in the characteristics of goodware and malware across markets.

This point is further underscored when examining the F_1-Score of classifiers trained on samples from one market, but tested on an even distribution from mixed markets($\mathcal{D}_{EVEN}$ rows 5–6). For example, training on $\mathcal{D}_{GP}$ and testing on $\mathcal{D}_{EVEN}$ yields F_1-Score significantly lower than the scores observed when testing on $\mathcal{D}_{GP}$ alone (row 1). Similar trends are observed for $\mathcal{D}_{3PM}$ and are consistent for all classifiers.

Combining markets for training/testing. Prior work [19], [78], [97] has adopted multi-market sampling. Here, we analyze training on mixed markets and testing on mixed or single markets. We find that, regardless of the proportion ($\mathcal{D}_{EVEN}$ or $\mathcal{D}_{PROP}$) of GooglePlay to 3PM apps, the F_1-Score when testing on multiple markets lies within the range of the F_1-Scores when testing on the individual markets (rows 7–9), with

$\mathcal{D}_{PROP}$ skewed towards the majority market (GooglePlay in rows 10–12).

Goodware and malware from different markets. Including goodware and malware from distinct markets ($\mathcal{D}_{GP3PM}$) lead to unrealistically-high F_1-Score (row 13,16). This aligns with findings from [17], where classifiers inadvertently learn spurious correlations, distinguishing app origins rather than actual differences between goodware and malware. These results clearly demonstrate the influence of app markets on classifier performance and the risk of introducing biases.

C. Recommendations

Arp et al. [17] showed that sampling malware and goodware from different markets can lead to spurious correlations. We extend this finding by showing apps from different markets are inherently different (even if the class ratio stays consistent) and training on one market does not transfer well to another. To ensure goodware and malware is not biased by market source, one should sample both from the same market. To include multiple markets, one must sample from each separately and test on single markets to obtain best and worst case estimates.

Recommendation: App Markets

The App Markets distribution **must** be consistent between goodware and malware apps, to avoid spatial bias. When performing multi-market evaluation, you **must** report the single-market test performances to obtain the best- and worst-case outcomes.

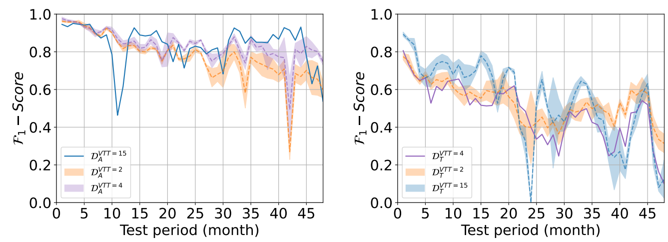
VII. Factor 4: VirusTotal Threshold

A. Hypothesis

Although the VirusTotal Threshold is typically used to label a given dataset, it also acts as a sampling parameter: samples with a number of detections between one and the chosen VTT value are excluded from the base population (these samples are usually called *greyware*). This is particularly evident in AndroZoo, where setting a VTT of 4 excludes over 56% of the non-benign population. For this reason, we posit that selecting different VTT values will significantly affect the composition of malware in a dataset, consequently impacting the reported performance of a classifier. We focus on three representative VTTs, VTT=15 based on D_A, VTT=4 based on D_T, and VTT=2, based on the fact that previous literature demonstrated it is the lowest stable VTT one can choose [99].

Hypothesis: VirusTotal Threshold

The choice of VTT affects the distribution from which Android malware is sampled, as it acts as a sampling parameter that filters the base population. In turn, this affects the reported performance of a classifier.



(a) D_A (b) D_T

Figure 6: **$\mathbf{F_1}$-Score of Hcc classifier for D_T and D_A sampled with a VTT=2, VTT=4, and VTT=15**. All other classifiers are available in Appendix §F.

B. Impact

In this section, we investigate the impact of VTT on detection performance for both D_A and D_T. To this end, we imitate the sampling performed by D_A and D_T, varying only the VTT values. More specifically, we sample new malware using VTTs 2, 4, 15 based on the amount of malware per month in D_A and D_T. We imitate D_T by filtering out samples that were not present on AndroZoo after 2019. For D_A, we discard samples crawled by AndroZoo after 2020 as that was when [97] was published and only include APKs found in VirusShare, the VT, and the AMD [91] datasets. We perform each sampling three times for statistical significance. Further details regarding the imitation of the sampling of D_A and D_T can be found in Appendix §A.

Figure 6 shows the F_1-Score of an Hcc classifier on the datasets sampled using different VTT values. We include the performance plots for the remaining classifiers in Appendix §F. We observe that resampling D_A with a VTT of 4 does not statistically affect the AUT of an Hcc classifier when evaluating on the complete evaluation window (i.e., AUT=0.84 for the original D_A against an average AUT of 0.83±0.01). In contrast, using a VTT of 2 does impact the reported performance, with the average AUT dropping to 0.77±0.03. However, we note that drift trends are accentuated in the last 18 months of our resampled datasets (particularly VTT=2), with the AUT dropping from 0.84 for the original D_A to 0.79±0.03 for VTT=4 and to 0.68±0.05 for VTT=2.

In the case of D_T, employing a VTT of 2 or 15 does not statistically affect the average yearly AUT (from 0.46 in the original dataset to 0.45±0.02 and 0.50±0.03). However, we notice that D_T has only 75 samples in December, 2016, which affects evaluation metrics. When excluding this date, we find that datasets with VTT=15 have an average AUT of 0.56±0.13 against an AUT of 0.49 for the original D_T.

C. Recommendations

The choice of VTT impacts the sampling process and may affect classification performance. Considering that a higher VTT will filter out more samples, a lower VTT will result in a more inclusive and therefore representative dataset. Provided that a minimum of 2 detections is required to reduce noisy labels [99], we recommend using VTT=2.

Recommendation: VirusTotal Threshold

You **must** use VTT=2 to sample from a distribution of app markets closer to the original population.

VIII. Factor 5: Dataset Size

A. Hypothesis

Evaluating classifier performance over the entire Android APK population (or a surrogate such as AndroZoo) is infeasible: AndroZoo alone comprises more than 25.8 million samples spanning 12+ years. Conversely, arbitrarily-small samplings may not reflect the underlying population.

D_A and D_T contain 241,611 and 259,230 samples, respectively, between 2014–2018, yet neither work justifies its chosen sample size. For instance, D_A samples 500 malware and nine times as much goodware per month, while D_T uses varying monthly sizes in its early years before fixing the count at 500 malware and 5,000 goodware in 2017–2018. Although their statistical representativeness remains unclear, both datasets follow Tesseract domain constraints and best practices from prior work: $C1$ (temporal training consistency), $C2$ (consistent time windows for goodware and malware), and $C3$ (realistic 10% malware-to-goodware testing ratios) [78].

In contrast, Miranda et al. [76] propose a statistical framework that determines minimum sample sizes for representativeness using the classical *margin of error* with finite population correction and a Bonferroni adjustment over multiple and security-unrelated characteristics (e.g., binned APK size and release year) [22], [76]. The key motivation for this design was that security-related characteristics (e.g., cryptographic API calls) were considered hard to compute and, if incorporated, might introduce bias into the dataset.

However, certain domain-specific constraints should still be taken into account determining the sample size. We posit that combining the uniform statistical sample of DaDa with domain-specific constraints by Tesseract leads to more consistent curating datasets in the same timeframe.

Hypothesis: Dataset Size

Relying solely on non-security related characteristics is insufficient for constructing datasets appropriate for Android malware classification; security-specific constraints proposed in Tesseract [78] are necessary to ensure representative datasets.

B. Impact

Adapting DaDa.

DaDa [76] originally estimates the minimal sample size using uniform sampling, i.e., without distinguishing between malware and goodware, and only accounting for general application characteristics shared across both classes. We introduce two incremental modifications to DaDa. First, we move from uniform to stratified sampling by applying DaDa independently to malware and goodware (Stratified), ensuring statistically sufficient representation of both classes. Second, to align with the $C3$ constraint suggesting for 10% malware test ratio [78], we compute the minimum sample size per class and then increase goodware to nine times the malware amount, yielding realistic malware-to-goodware ratios while preserving same malware volume [78]. Moreover, notice that the malware-to-goodware ratio is a parameter of the algorithm and can therefore be changed to other values. This flexibility is required to address potential future changes in the percentage of malware in the wild. We refer to the resulting procedure as *Statistically representative,* Tesseract*-guided Android Sampling strategy* (Stas). This method combines the benefits of stratified sampling with realistic class ratios.

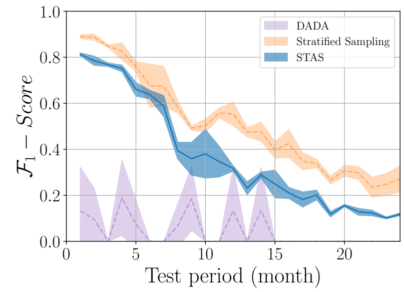


Figure 7: **$\mathbf{F_1}$-Score score for Hcc on datasets sampled using DaDa, Stratified and Stas**. DaDa alone is insufficient for creating Android malware datasets. Stratified inflates detection performance since it produces unrealistic class ratios (almost 50% malware). Stas creates stable and realistic datasets. All other classifiers are available in Appendix §G.

Sampling new datasets. To assess the impact of uniform and stratified sampling, we construct nine datasets using DaDa, stratified sampling, and Stas. Guided by our findings and recommendations in §IV, §VI, and §VII, we apply a VTT=2 to capture more of the population, eliminate market-specific biases by restricting sampling to GooglePlay, and use GooglePlay *upload_dates* as the most reliable timestamps. Following prior work showing that malware labels stabilize within one year [75], [99], we sample three years of data spanning 2021–2023. We apply the generic Android characteristics defined in DaDa (i.e., 4 values of APKs, 10 boolean permissions) and only change APK release year to 2021–2023 (3 values).[1]

Figure 7 shows the temporal performance of the three sampling approaches. A key distinction between DaDa, a uniform sampling approach, versus the two stratified sampling approaches is that the latter guarantees a statistically sufficient number of both goodware and malware samples Table VII.

[1]To aid future works, we release Hypercube, the first version of the dataset sampled using Stas following the aforementioned parameters. Although Hypercube is in a fixed time frame, the Stas methodology can be easily used to sample newer datasets in the future.

Table VII: **Average A-AUT across three datasets sampled using DaDa, Stratified and Stas for 2021–2023**. Depending on the sampling strategy employed, the resulting dataset size changes, affecting the overall performance.

Classifier	Average A-AUT across three datasets DaDa	Stratified	Stas
Drebin	0.09±0.03	0.59±0.01	0.43±0.00
DeepDrebin	0.01±0.01	0.61±0.01	0.44±0.01
MalScan	0.11±0.03	0.58±0.01	0.35±0.00
Ramda	0.01±0.01	0.55±0.02	0.27±0.03
Hcc	0.03±0.03	0.60±0.02	0.46±0.00
Malware Size	206 (0.78%)	19,645 (42%)	19,645 (10%)
Goodware Size	26,741 (99.22%)	26,946 (58%)	176,805 (90%)
Dataset size	26, 497	46,591	196,450

DaDa yields a small dataset of 26,947 samples compared to 46,591 samples for stratified sampling without enforcing Tesseract's $C3$ constraint. However, the implications is that stratified sampling produces a relatively stable F_1-Score, suggesting suitability for Android malware evaluation.

Both stratified sampling approaches show low variance when sampled multiple times, Stas shows a lower F_1-Score overall. Although a lower F_1-Score does not necessarily indicate better representativeness, it correlates with the findings of Liu et al. [69] and Tesseract [78], where an imbalance of malware-to-goodware can inflate performance results. Moreover, following Tesseract's recommendation of enforcing $C3$ would result in a significantly larger dataset size of 196,450 samples. While this entails higher computational costs, Stas nevertheless represents the minimum necessary to construct a dataset that is both statistically sufficient and realistic.

C. Recommendations

We show that combining temporal and spatial constraints with statistical sampling algorithms is necessary to estimate an adequate minimum number of samples required for a dataset to be representative of the real-world population.

Recommendation: Dataset Size

Dataset size **must** be statistically representative of the population and follow domain-specific guidelines. You **may** use Stas, to determine an appropriate sampling size for an Android malware dataset.

IX. Factor Prevalence Analysis

To confirm whether the identified factors have been overlooked by the research community, we conduct a thorough analysis on existing Android malware datasets.

We first compile a representative list of Android malware datasets used in previous work using a forward-citation methodology, i.e.,, identifying papers that cite a given dataset. We begin with two of the earliest and most widely used datasets: Malgenome [98] and Drebin [18], each with over 3,000 citations and were published in a top security venue (IEEE S&P 2012 and NDSS 2014, respectively).

To narrow our scope, we initially include works published in one of the top-four security venues: USENIX Security, ACM CCS, IEEE S&P, and NDSS. This helps us filter out papers that do not address security topics and focus on those that apply machine learning to security-related research problems. We also include relevant works outside these venues by querying Google Scholar for “Android Malware Detection” and selecting the top 100 results (as of July 10th, 2025).

This results in an initial list of 233 papers, which we review to identify datasets used for experimental evaluation. To ensure comprehensiveness of our survey, we performed forward-citations for every dataset we found, and filtered for the top-four security venues. This not only informed us of the prevalence of each dataset, but also reduced the chance that we missed well-known datasets.

In total, we analyze 527 Android malware detection papers and identify 42 datasets, which are explicitly used in 154 papers, including 35 from top-four security venues. Appendix §H details our methodology, dataset characteristics, and factor assessment criteria.

Figure 8 summarizes the prevalence of the five factors across the 42 datasets. We observe that crucial curation practices are often neglected: 33 datasets ignored timestamp types for sampling (§IV), which was shown to affect the distribution the sampled dataset represents; only 7 evaluated their dataset on different temporal splits; and over half mixed APK sources. Labeling practices were similarly inconsistent, with 7 datasets using VTT=2, 12 relying on other criteria, and 20 employed VTT>2 or mixed methods. Although 17 datasets were statistically significant in size, none explicitly justified their chosen dataset size. Ultimately, only 2 datasets [19], [78] satisfied at least three of our recommendations, while the remaining 40 violated three or more.

Overall, our findings suggest that best practices defined in prior work have not clearly outlined and addressed these issues we have highlighted.

X. Discussion

While we extend the set of best practices for evaluating Android malware classifiers under spatio-temporal drift ([17], [36], [78], [80], [83]), several open problems still exist. Here, we contextualize our findings and discuss related challenges.

Other malware domains. While our study focuses on Android malware (as it is the only domain for which large-scale timestamped repositories of both benign and malicious apps are available), many of the challenges may be shared with other malware domains, such as Windows and PDFs. For example, getting a reliable timestamp source can be difficult, especially when there are no centralized app markets such as in Android. Jiang et al. [50] recommend using *VT_first_dates* as the best estimate for Windows PE; however, we have shown how Third-Party Timestamps may be misaligned with the real publication date of software, at least for GooglePlay. Temporal Luck and our recommendation remains relevant in other malware domains as well. Lastly, our algorithm for

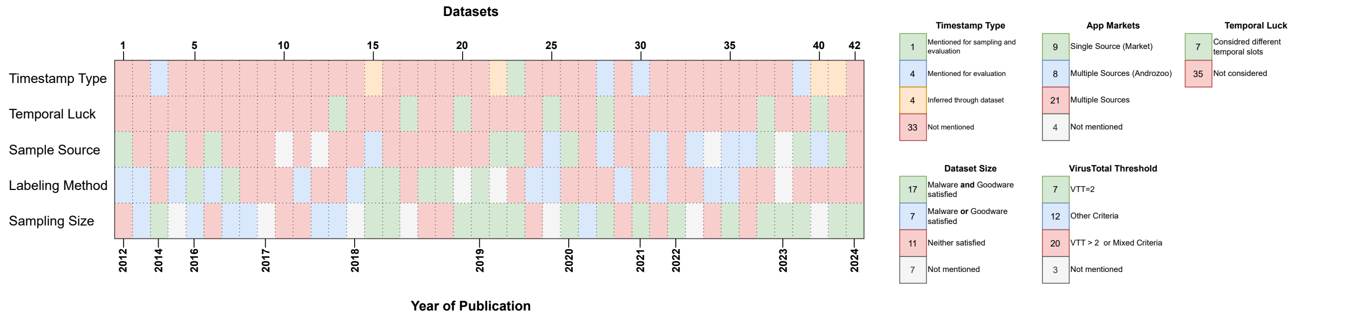


Figure 8: **Summary of the prevalence analysis of bias factors** in datasets used in the top-4 security conferences and in the top-100 Google Scholar results. Of the 42 datasets we surveyed, only two datasets satisfy three of our recommendations [19], [78], all other datasets violate three or more. This suggests our factors are largely overlooked in previous work.

estimating the dataset size can be applied to other malware domains if the general population distribution is available.

Evolution of TESSERACT $C3$ [78]. We observed that malware is not consistently 10% across all months in AndroZoo. However, obtaining a "true" malware ratio would require help from industrial partners. TESSERACT [78] estimated the percentage of malware samples at 10% in 2019, which may no longer be valid in the current landscape. For lack of a better source, assuming 10% as a constraint allows for consistency in the sampled datasets. Future work should investigate the temporal evolution of realistic malware ratios.

K-fold evaluation. TESSERACT explained that K-fold cross-validation is an upper bound estimation for performance in the absence of drift [78], and proposed AUT for time-aware evaluations [78]. In §V, we argue that temporal evaluation should be performed with rolling-window splits to mitigate non-stationary "lucky splits", similar to how K-fold avoids stationary lucky splits. Although this approach is standard for financial time-series forecasting [27], its implications remain largely unexplored in the security domain.

Family trends in a dataset. This work highlights the prevalence of re-emerged malware (§IV) in the wild, which may impact the reported performance of a classifier. This is indirectly reflected in our discussion on "lucky temporal splits," to which the A-AUT offers a mitigation (§V). Showing the family overlap in a dataset may help understand better the impact this may have on the underlying classifier. We acknowledge this requires a well-defined protocol to encourage sound comparisons; we highlight this as a current limitation of our approach, leaving this as future work.

Grayware. Our work provides evidence that lower VTT values better represent the malware population. While we generally consider the problem of benign vs. non-benign, accounting for *grayware*, i.e., APKs detected by at least one Anti-Virus but less than VTT, which are predominant in AndroZoo, remains an open problem. Previous work [16] has attempted to categorize different grayware types in GooglePlay. However, systematizing grayware remains an open problem.

Adversarial settings. Adversarial attacks in the Android malware domain typically involve modifications to malicious code that induce misclassifications during classification [43], [59]. While such settings are important for evaluating the robustness of classifiers, the focus of this work is on curating realistic datasets that capture natural distribution shift of Android malware, not adversarial benchmark datasets. Future research should further investigate how dataset biases may influence evaluations under adversarial conditions. Similarly, real-world attackers have been using code obfuscation techniques and packing to fool malware detectors [31]. Future work should investigate the impact of code packers on dataset construction.

Alternative sampling strategies. We acknowledge that our statistical-based down-sampling may unintentionally under-represent rare malware families, as these are statistically less prevalent in the population. Consequently, datasets sampled using our approach may be dominated by widespread families while excluding those with fewer samples. Addressing this limitation is non-trivial: obtaining reliable family labels for entire malware populations is impractical due to VirusTotal API rate limits and inconsistent labeling across antivirus vendors. We leave the exploration of family-aware sampling strategies to future work.

XI. RELATED WORK

In this section, we cover the research most closely related to our work. We present both seminal research papers tackling the wider problem of achieving fair evaluations in ML-based computer security research and more directly comparable publications on dataset sampling and debiasing.

Guidelines for realistic ML evaluations in computer security. Our research aligns with efforts to establish guidelines for realistic evaluations in security-related applications. Sommer and Paxson [83] highlighted key challenges in applying ML to network intrusion detection systems (NIDS), pointing out fundamental issues that persist in this domain. Rossow et al. [80] and Van der Kouwe et al. [88] further analyzed common shortcomings and best practices in security system evaluations. However, these works primarily identify challenges and methodological pitfalls without quantifying their impact on classifier performance. Arp et al. [17] extended this line of work by demonstrating the effects of these pitfalls in realistic scenarios, although with focus on general cybersecurity. In contrast, our work investigates the specific properties and risk factors during dataset creation for the Android malware domain that lead to pitfalls identified in prior studies.

Temporal evaluations. Other relevant work includes the domain of temporal evaluation primarily treated by Allix et al. [13] and Miller et al. [75]. Recently, Pendlebury et al. [78] proposed TESSERACT as a framework to measure impact of performance decay over time. We expand on TESSERACT by proposing a new evaluation guideline in Temporal Luck. Additionally, we address the creation timestamps validity by experimentally showing the impact of inconsistencies and providing practical recommendations for researchers (§IV).

Realistic datasets for ML-based cybersecurity. Prior work also focused on the problem of realistic and benchmark datasets for ML-based cybersecurity. Sommer and Paxon [83] highlighted the challenges of creating realistic network intrusion detection datasets, mostly due to privacy concerns; attempts to create datasets artificially, such as DARPA98-99 [67] and KDD99 [5], failed because they introduced artifacts [32], [71], [74]. In the mobile malware domain, Haque et al. proposed a new dataset LAMBDA [45], which contains over one million APKs collected across 12 years, using crawl dates and VTT =4. However, LAMBDA does not resolve the biases we identify: STAS is a sampling strategy designed to help researchers build statistically-sound and bias-aware datasets for any past or future time window, rather than simply aggregating a large corpus of samples.

Similarly, Jian et al. [50] created a benchmark dataset for Windows malware by sampling from VirusShare, following a similar dataset curation pipeline to the one we presented. However, we focus on factors related to representativeness, such as sample size and markets, rather than the quality of family labels. [23] also presented a preliminary discussion on dataset size and markets but lacks empirical evaluation or concrete recommendations, while we provide both. Notably, [23] calls for better dataset practice—our work directly answers its call. Recently, Flood et al. [36] identified six poor practices in NIDS-specific datasets, including poor data diversity, highly dependent features, unclear ground truths, traffic collapse, artificial diversity, and wrong labels. Instead of focusing on the limitations of network datasets, we focus our attention on the Android malware domain, where the presence of AndroZoo [1], [11], [14] has offered opportunities to build more realistic datasets. Furthermore, all our factors are directly applicable during the sampling process, helping to prevent the identified "bad" choices in [36]. [40] analyzed three dataset factors (i.e., class imbalance, quality, and timelines) by examining three state-of-the-art datasets. We differ from this work as we not only show how our sampling factors can affect the sampled distribution but also give actionable recommendations that researchers can follow to reduce the biases highlighted in [40]. Additionally, we do not limit our work to only two state-of-the-art datasets, but instead include the wider Android population in our experimental analysis. In conclusion, we identify five factors that can affect realistic evaluations and that can be controlled with actionable recommendations (§II).

Research on Android malware sampling. Prior work has focused on the creation of representative Android benchmark datasets. Miranda et al. proposed DADA [76], a two-step sampling approach for debiasing Android malware datasets, which combines statistical methods with general APK characteristics. However, in §VIII, we have shown that DADA lacks critical domain-specific knowledge. In particular, it violates $C3$ of [78] and P1 of [17]. Sun et al. [86] proposed a dataset restructuring algorithm that unrealistically requires family information and applies it to [38]. However, it does not correct MalNet's severe spatial bias (only about 6% goodware). A recent work [87] proposed an approach to detect and mitigate sampling bias between a labeled training and an unlabeled testing dataset, based on domain discrimination. In contrast, our work questions how well datasets represent the real-world population, rather than fitting a classifier on two divergent datasets.

XII. CONCLUSION

Motivated by the stark performance difference of Android malware classifiers on the *APIGraph* and *Transcendent* datasets, we reflected on dataset curation parameters, and identified five factors overlooked by previous research which can lead to unrealistic datasets: Timestamp Type, Temporal Luck, App Markets, VirusTotal Threshold, and Dataset Size. We have shown their impact on dataset composition (via family overlap) and on detection performance on five SotA classifiers. Since these factors are deeply interconnected (e.g., modifying one will affect the others), they need a cohesive approach: for each bias factor, we proposed actionable recommendations to cancel their impact; we propose A-AUT as an evaluation metric to address the impact of lucky temporal splits; we then consolidate our findings in proposing STAS, a statistically guided TESSERACT-constrained sampling strategy.

To encourage future research to exercise greater caution when selecting parameters during dataset curation, we release code for the STAS strategy, and hashes of HYPERCUBE, a dataset sampled for 2021–2023 following our guidelines.

Although curating a static benchmark dataset may not be practical for Android malware classification due to natural distribution shift, following principled dataset curation practices is essential. Doing so will enable fairer, more reliable, and ultimately more trustworthy evaluations in this domain.

ACKNOWLEDGMENT

Zeliang Kan contributed to this work during his Ph.D. studies at King's College London. This research was partially supported by: the UK EPSRC Grant EP/X015971/2; Google ASPIRE and GARA Awards; and the Vienna Science and Technology Fund (WWTF) through the BREADS project (10.47379/VRG23011).

REFERENCES

[1] Androzoo. https://androzoo.uni.lu. Last Accessed: 2025-01-18.

[2] Contagio. https://contagiominidump.blogspot.com/.

[3] Google issue tracker - last modified timestamp. https://issuetracker.google.com/issues/37116029. Accessed: 2025-01-09.

[4] Kaggle datasets. https://www.kaggle.com/datasets?fileType=csv. Accessed: 2025-01-12.

[5] Kdd cup data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[6] Uci machine learning datasets. https://archive.ics.uci.edu/datasets. Accessed: 2025-01-12.

[7] Virusshare. https://virusshare.com/. Last Accessed: 2025-01-18.

[8] Virustotal. https://www.virustotal.com. Last Accessed: 2025-01-18.

[9] Andi Fitriah Abdul Kadir, Natalia Stakhanova, and Ali Akbar Ghorbani. Android botnets: What urls are telling us. In *Network and System Security: 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings 9*, pages 78–91. Springer, 2015.

[10] Shahid Alam, Zhengyang Qu, Ryan Riley, Yan Chen, and Vaibhav Rastogi. Droidnative: Automating and optimizing detection of android native code malware variants. *computers & security*, 65:230–246, 2017.

[11] Marco Alecci, Pedro Jesús Ruiz Jiménez, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. Androzoo: A retrospective with a glimpse into the future. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pages 389–393. IEEE, 2024.

[12] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 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*, 21:183–211, 2016.

[13] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Are your training datasets yet relevant? an investigation into the importance of timeline in machine learning-based malware detection. In *International Symposium on Engineering Secure Software and Systems*, pages 51–67. Springer, 2015.

[14] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[15] Iman Almomani, Tala Almashat, and Walid El-Shafai. Maloid-ds: Labeled dataset for android malware forensics. *IEEE Access*, 2024.

[16] Benjamin Andow, Adwait Nadkarni, Blake Bassett, William Enck, and Tao Xie. A study of grayware on google play. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 224–233. IEEE, 2016.

[17] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[18] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[19] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[20] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pages 359–370, 1994.

[21] Haipeng Cai, Na Meng, Barbara Ryder, and Daphne Yao. Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, 14(6):1455–1470, 2018.

[22] George Casella and Roger Berger. *Statistical inference*. CRC press, 2024.

[23] Fabrício Ceschin, Marcus Botacin, Albert Bifet, Bernhard Pfahringer, Luiz S Oliveira, Heitor Murilo Gomes, and André Grégio. Machine learning (in) security: A stream of problems. *Digital Threats: Research and Practice*, 5(1):1–32, 2024.

[24] Sen Chen, Minhui Xue, and Lihua Xu. Towards adversarial detection of mobile malware: poster. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 415–416, 2016.

[25] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *USENIX Security Symposium*, 2023.

[26] Theo Chow, Zeliang Kan, Lorenz Linhardt, Lorenzo Cavallaro, Daniel Arp, and Fabio Pierazzi. Drift forensics of malware classifiers. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 197–207, 2023.

[27] Marcos Lopez De Prado. *Advances in financial machine learning*. John Wiley & Sons, 2018.

[28] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE transactions on dependable and secure computing*, 16(4):711–724, 2017.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[30] Meghna Dhalaria and Ekta Gandotra. A hybrid approach for android malware detection and family classification. *IJIMAI*, 6(6):174–188, 2021.

[31] Wael F Elsersy, Ali Feizollah, and Nor Badrul Anuar. The rise of obfuscated android malware and impacts on detection methods. *PeerJ Computer Science*, 8:e907, 2022.

[32] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12. IEEE, 2021.

[33] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security*, 13(8):1890–1905, 2018.

[34] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[35] Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. Anastasia: Android malware detection using static analysis of applications. In *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE, 2016.

[36] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. Bad design smells in benchmark nids datasets. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, pages 658–675. IEEE, 2024.

[37] Yanick Fratantonio, Antonio Bianchi, William Robertson, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Triggerscope: Towards detecting logic bombs in android applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 377–396. IEEE, 2016.

[38] Scott Freitas, Rahul Duggal, and Duen Horng Chau. Malnet: A large-scale image database of malicious software. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3948–3952, 2022.

[39] Tatiana Frenklach, Dvir Cohen, Asaf Shabtai, and Rami Puzis. Android malware detection via an app similarity graph. *Computers & Security*, 109:102386, 2021.

[40] Xiuting Ge, Yifan Huang, Zhanwei Hui, Xiaojuan Wang, and Xu Cao. Impact of datasets on machine learning based methods in android malware detection: an empirical study. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 81–92. IEEE, 2021.

[41] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

[42] Hugo Gonzalez, Natalia Stakhanova, and Ali A Ghorbani. Droidkin: Lightweight detection of android apps similarity. In *International Conference on Security and Privacy in Communication Networks: 10th International ICST Conference, SecureComm 2014, Beijing, China, September 24-26, 2014, Revised Selected Papers, Part I 10*, pages 436–453. Springer, 2015.

[43] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *ESORICS*. Springer, 2017.

[44] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers & Security*, 110:102399, 2021.

[45] Md Ahsanul Haque, Ismail Hossain, Md Mahmuduzzaman Kamol, Md Jahangir Alam, Suresh Kumar Amalapuram, Sajedul Talukder, and Mohammad Saidur Rahman. Lamda: A longitudinal android malware benchmark for concept drift analysis. *arXiv preprint arXiv:2505.18551*, 2025.

[46] Chihiro Hasegawa and Hitoshi Iyatomi. One-dimensional convolutional neural networks for android malware detection. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pages 99–102. IEEE, 2018.

[47] Yiling He, Jian Lou, Zhan Qin, and Kui Ren. Finer: Enhancing state-of-the-art classifiers with feature attribution to facilitate security analysis. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 416–430, 2023.

[48] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[49] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. Euphony: Harmonious unification of cacophonous antivirus vendor labels for android malware. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 425–435. IEEE, 2017.

[50] Yongkang Jiang, Gaolei Li, Shenghong Li, and Ying Guo. Benchmfc: A benchmark dataset for trustworthy malware family classification under concept drift. *Computers & Security*, 139:103706, 2024.

[51] Abdullah Talha Kabakus. Droidmalwaredetector: A novel android malware detection framework based on convolutional neural network. *Expert Systems with Applications*, 206:117833, 2022.

[52] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital investigation*, 24:S48–S59, 2018.

[53] David Sean Keyes, Beiqi Li, Gurdip Kaur, Arash Habibi Lashkari, Francois Gagnon, and Frédéric Massicotte. Entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pages 1–12. IEEE, 2021.

[54] Saurabh Kumar, Debadatta Mishra, Biswabandan Panda, and Sandeep Kumar Shukla. Androobfs: time-tagged obfuscated android malware dataset with family information. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 454–458, 2022.

[55] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual conference on privacy, security and trust (PST)*, pages 233–23309. IEEE, 2017.

[56] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan conference on security technology (ICCST)*, pages 1–7. ieee, 2018.

[57] Michael Ley et al. Dblp computer science bibliography. https://dblp.org, 2024. Accessed: 2025-06-26.

[58] Heng Li, Zhang Cheng, Bang Wu, Liheng Yuan, Cuiying Gao, Wei Yuan, and Xiapu Luo. Black-box adversarial example attack towards {FCG} based android malware detection under incomplete feature information. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1181–1198, 2023.

[59] Heng Li, ShiYao Zhou, Wei Yuan, Jiahuan Li, and Henry Leung. Adversarial-example attacks toward android malware detection system. *IEEE Systems Journal*, 14(1):653–656, 2019.

[60] Heng Li, Shiyao Zhou, Wei Yuan, Xiapu Luo, Cuiying Gao, and Shuiyan Chen. Robust android malware detection against adversarial example attacks. In *Proceedings of the Web Conference 2021*, pages 3603–3612, 2021.

[61] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[62] Li Li, Tegawendé Bissyandé, and Jacques Klein. Moonlightbox: Mining android api histories for uncovering release-time inconsistencies. In *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*, pages 212–223. IEEE, 2018.

[63] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security*, 12(6):1269–1284, 2017.

[64] Wen Li, Xiaoqin Fu, and Haipeng Cai. Androct: ten years of app call traces in android. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 570–574. IEEE, 2021.

[65] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *2015 IEEE 39th annual computer software and applications conference*, volume 2, pages 422–433. IEEE, 2015.

[66] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. Andrubis–1,000,000 apps later: A view on current android malware behaviors. In *2014 third international workshop on building analysis datasets and gathering experience returns for security (BADGERS)*, pages 3–17. IEEE, 2014.

[67] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[68] Xing Liu and Jiqiang Liu. A two-layered permission-based android malware detection scheme. In *2014 2nd IEEE international conference on mobile cloud computing, services, and engineering*, pages 142–148. IEEE, 2014.

[69] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. Explainable ai for android malware detection: Towards understanding why the models perform so well? In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 169–180. IEEE, 2022.

[70] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A Ghorbani. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 515–522. IEEE, 2020.

[71] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[72] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. MaMaDroid: Detecting android malware by building markov chains of behavioral models. *NDSS*, 2017.

[73] Alejandro Martín, Raúl Lara-Cabrera, and David Camacho. Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142, 2019.

[74] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[75] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *DIMVA*. Springer, 2016.

[76] Tomás Concepción Miranda, Pierre-Francois Gimenez, Jean-François Lalande, Valérie Viet Triem Tong, and Pierre Wilke. Debiasing android malware datasets: How can i trust your results if your dataset is biased? *IEEE Transactions on Information Forensics and Security*, 17:2182–2197, 2022.

[77] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. {FlowCog}: Context-aware semantics extraction and analysis of information flow leaks in android apps. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1669–1685, 2018.

[78] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX security symposium (USENIX Security 19)*, pages 729–746, 2019.

[79] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.

[80] Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE symposium on security and privacy*, pages 65–79. IEEE, 2012.

[81] SerpApi, LLC. Serpapi: Real-time google search api. https://serpapi.com, 2025. Accessed: 2025-06-26.

[82] Feng Shen, Justin Del Vecchio, Aziz Mohaisen, Steven Y Ko, and Lukasz Ziarek. Android malware detection using complex-flows. *IEEE Transactions on Mobile Computing*, 18(6):1231–1245, 2018.

[83] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[84] N Stakhanova, A Ghorbani, et al. An empirical analysis of android banking malware. 2016.

[85] Bo Sun, Takeshi Takahashi, Tao Ban, and Daisuke Inoue. Detecting android malware and classifying its families in large-scale datasets. *ACM Transactions on Management Information Systems (TMIS)*, 13(2):1–21, 2021.

[86] Tiezhu Sun, Nadia Daoudi, Weiguo Pian, Kisub Kim, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. Temporal-incremental learning for android malware detection. *ACM Transactions on Software Engineering and Methodology*, 2024.

[87] Saravanan Thirumuruganathan, Fatih Deniz, Issa Khalil, Ting Yu, Mohamed Nabeel, and Mourad Ouzzani. Detecting and mitigating sampling bias in cybersecurity with unlabeled data. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1741–1758, 2024.

[88] Erik van der Kouwe, Gernot Heiser, Dennis Andriesse, Herbert Bos, and Cristiano Giuffrida. Sok: Benchmarking flaws in systems security. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 310–325. IEEE, 2019.

[89] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. Malradar: Demystifying android malware in the new era. *Proc. ACM on Measurement and Analysis of Computing Systems*, 2022.

[90] Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. Malwhiteout: Reducing label errors in android malware detection. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[91] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In Michalis Polychronakis and Michael Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 252–276, Cham, 2017. Springer International Publishing.

[92] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 252–276. Springer, 2017.

[93] Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In *Proceedings of the 2014 conference on research in adaptive and convergent systems*, pages 247–252, 2014.

[94] Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 139–150. IEEE, 2019.

[95] Ke Xu, Yingjiu Li, and Robert H Deng. Iccdetector: Icc-based malware detection on android. *IEEE Transactions on Information Forensics and Security*, 11(6):1252–1264, 2016.

[96] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 719–736. IEEE, 2023.

[97] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 757–770, 2020.

[98] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

[99] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online {Anti-Malware} engines. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2361–2378, 2020.

APPENDIX

A. Datasets used in Evaluations

***APIGraph* and *Transcendent* Dataset.** Our work begins with a motivational example of two state-of-the-art datasets, *APIGraph* [97] and *Transcendent* [19]. While they are not the only Android Malware datasets available, we picked them according to their popularity and quality. Of the 527 papers we reviewed in §IX, we found 2 top conference papers that use the *APIGraph* and *Transcendent* datasets. We also noticed many of the papers sampled their own dataset, suggesting there is a lack of consistency in the datasets used in our community. To enable us to compare datasets within the same time frame, we picked *APIGraph*, a well established dataset that was used in the state-of-the-art android malware classifier [25] and overlaps with D_T. Both these datasets help us highlight issues with how datasets are being curated for Android Malware detection.

App Markets dataset. In §VI we collected 15,000 goodware and 15,000 malware samples for each GooglePlay and Third party app markets through AndroZoo, resulting in a total of 60,000 samples. We then combined them to create 6 different dataset configurations detailed in Table VIII.

HYPERCUBE. The HYPERCUBE dataset was sampled using STAS, a statistically guided TESSERACT constrained sampling strategy described in §VIII. We used STAS to determine the amount of malware and goodware per month required for a representative dataset. We chose a recent time frame of 2021–2023 as it was shown in prior work that labels tend to stabilize after one year [75], [99]. The dataset follows a VTT=2 (c.f. §VII), samples only from GooglePlay (c.f. §VI) and uses GooglePlay *upload_dates* for both sampling and evaluation (c.f. §IV). In §VIII, we sampled using STAS three times. HYPERCUBE refers to the first version, which we release alongside the code for STAS.

Imitating APIGraph. In §IV and §VII, we attempt to imitate the sampling in D_A by sampling APKs from AndroZoo that was collected from VirusShare, VirusTotal and AMD dataset. We filtered out samples with a crawl date of 2020 (same publication year as D_A) as they would not have been in the AndroZoo when D_A was sampled. We kept the amount of malware and goodware per month exactly the same, hence the dataset size and class ratio would be exactly the same as the original D_A. We used *dex_dates* for sampling since we were unable to obtain *VT_first_dates* for all samples on AndroZoo. We used VTT=15 (except in the §VII where we changed VTT values to 2 and 4).

Imitating Transcendent. In §IV-B and §VII, we attempt to imitate the sampling in D_T by sampling APKs from AndroZoo. After contacting the authors regarding the exact details for sampling, we discovered the D_T dataset was constructed in two parts by sampling in mid 2017 for the period 2014–2016

Table VIII: **Experimental configurations for the analysis of market sources.** The configurations include sampling solely from (GP), and Third Party Markets (3PM), sampled from both evenly, sampled from both but with more from GooglePlay, goodware from GooglePlay and malware from 3PM, and vice versa.

Configuration	***Train set***				***Test set***			
	Goodware		Malware		Goodware		Malware	
	GooglePlay	3PM	GooglePlay	3PM	GooglePlay	3PM	GooglePlay	3PM
$\mathcal{D}_{GP}$	10,000	0	10,000	0	4,500	0	500	0
$\mathcal{D}_{3PM}$	0	10,000	0	10,000	0	4,500	0	500
$\mathcal{D}_{EVEN}$	5,000	5,000	5,000	5,000	2,250	2,250	250	250
$\mathcal{D}_{PROP}$	8,000	2,000	8,000	2,000	3,600	900	4,000	100
$\mathcal{D}_{GP3PM}$	10,000	0	0	10,000	4,500	0	0	500
$\mathcal{D}_{3PMGP}$	0	10,000	10,000	0	0	4,500	500	0

and early 2019 for period 2017–2018. We imitated this by filtering out samples using crawl dates after mid 2017 for the first part and early 2019 for the second part. We sampled from all markets in AndroZoo for both malware and goodware, used *dex_dates* for sampling and used a VTT=4 (except in the §VII where we changed VTT values to 2 and 15).

B. Classifier Details

In addition to releasing the code of our approach, we also detail here which parameters we used in all the classifiers mentioned in §III and the time required by each for training and inference. All training and evaluations are performed on an Intel(R) Core(TM) Ultra 9 185H CPU with 16 cores and a GeForce RTX 4070 GPU, using D_T 2014 for training and D_T 2015-2018 for inference. Notice that we performed feature extraction for all three feature spaces before carrying out any experiment and, therefore, the reported times do not include it.

For DREBIN, we use a LinearSVM with the hyper-parameter $C = 1$, which has been found to be representative for a variety of scenarios [18], [19], [25], [28], [78]. A LinearSVM is a classical supervised ML algorithm that aims to find n hyperplane that separates data points belonging to the different classes (i.e., goodware and malware). Both training and inference are relatively fast, with data preprocessing, feature reduction, and model fitting took ≈122 seconds for one year of data, while inference took ≈346 seconds.

DEEPDREBIN is implemented as a Multi Layer Perceptron (MLP) with two hidden and densely connected layers composed of 200 neurons each and an output layer with two neurons; hidden layers use the Rectified Linear Unit activation function, whereas the output neurons employ the softmax function. Training (including data preprocessing, feature reduction, and model fitting) took ≈267 seconds for 20 epochs, while inference took ≈263 seconds.

MALSCAN [94] uses social-network-based centrality analysis to extract relevant features from function call graphs in APKs. In this paper, we have used the configuration with degree centrality and a Random Forest Classifier, with the hyper-parameter $n_{estimators} = 100$. Training took ≈232 seconds but required up to 70GB of memory, whilst inference took ≈434 seconds.

RAMDA [60] employs a subset of the Drebin feature space (which includes permissions, intent actions, and sensitive API calls) to represent APKs and combines a variational autoencoder (VAE) with an MLP; in particular, the compressed representation learned by the VAE is fed into the MLP. A sample is classified as malware if either the reconstruction error of the VAE is over a certain threshold or the MLP outputs the correspondent label. In this paper, we used the same configuration as in the original work, setting $\lambda_1 = 10$, $\lambda_2 = 1$ and $\lambda_3 = 10$. Training took ≈1,034 seconds for 50 epochs of VAE training and 50 epochs of MLP training, while inference took ≈404 seconds.

HCC [25] employs the Drebin feature space with a low-variance (0.1%) feature reduction. It combines an encoder, trained through contrastive hierarchical learning) and an MLP. In this work, we train the model for up to 40 epochs as we did not observe any performance benefits if it was increased. Training (including data preprocessing, feature reduction, and model fitting) took ≈1,459 seconds, while inference took ≈433.

C. Family Overlap Metric

Family Overlap Metric. In our analysis, we are interested in describing the *temporal distribution shift* of the malware population within one dataset without having to rely on any classifier's performance. Therefore, we introduce *family overlap* (Φ), a metric that models the shift in malware distribution within a dataset. Shown previously in Figure 2, it allows us to obtain more general insights into the dataset, without relying on the representation or the classifier employed. From this, we can observe how certain sampling parameters or algorithms influence the resulting dataset composition. While methods quantifying the difference of distributions exist (e.g., the Kullback-Leibler divergence), they typically require a representation space to operate in. In contrast, we designed family overlap to be representation-independent and interpretable, and specifically designed based on domain knowledge from the malware domain.

The *family overlap* (Φ) measures the the percentage of samples in a dataset $\mathcal{D}$ belonging to a family that was already known in different dataset $\mathcal{D}_{\text{ref}}$.

$$\Phi(\mathcal{D}, \mathcal{D}_{\text{ref}}) = \frac{|\{(x, y) \in \mathcal{D} \mid \exists\, (x', y') \in \mathcal{D}_{\text{ref}} \text{ s.t. } y = y'\}|}{|\mathcal{D}|} \tag{5}$$

Here, (x, y) are tuples of APKs and their malware family label. Unless specified otherwise, we use the training split of any given dataset as the reference dataset $\mathcal{D}_{\text{ref}}$ and some testing split for which to calculate the family overlap as $\mathcal{D}$.

Φ is helpful for measuring and visualizing malware trends over time, as it describes the percentage of malware belonging to families that already existed in the past. As it provides a compact representation of the distribution shift, it is particularly helpful for comparing the effects of different sampling strategies and is therefore largely employed in the remainder of this section.

D. Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm that measures the similarity between two time series. DTW tries to "warp" one time series to align it with the other; hence, it assumes that two time sequences are similar but "out of phase." For this reason, we apply it in IV-B to compare the time distributions of samples when using different timestamp types, as these may just be misaligned; for example, the *VT first_submission_date* time distribution may be in principle identical in shape to that of GooglePlay *upload_dates*, just shifted forward.

More specifically, DTW is a dynamic programming technique that attempts to minimize the total accumulated distance between points belonging to the two sequences. It achieves this by re-aligning the two sequences. Once the optimal solution is found, the total accumulated cost describes the dissimilarity between the two sequences.

E. Temporal Luck

A-AUT for smaller evaluation windows. When $n_E < n_T$, the evaluation window is smaller than the training window. To avoid gaps in temporal coverage, we advance by n_E at each step, resulting in contiguous evaluation sets while training sets overlap:

$$T := \{D_{in_E:in_E+n_T-1}\}_{i=0}^{k-1} \tag{6}$$

$$E := \{D_{n_T+in_E:n_T+(i+1)n_E-1}\}_{i=0}^{k-1} \tag{7}$$

$$k = \lfloor(|D| - n_T)/n_E\rfloor \tag{8}$$

Temporal Luck Example. We here provide an example of a dataset split according to Temporal Luck, as a useful overview to the A-AUT introduced in §V.

A dataset $\mathcal{D}$ containing data from January 2014 to December 2019 would be divided into the following dataset splits when using $n_T = n_E = 12$ (months):

$$T := \{D_{2014}, D_{2015}, D_{2016}, D_{2017}, D_{2018}\}$$
$$E := \{D_{2015}, D_{2016}, D_{2017}, D_{2018}, D_{2019}\}. \tag{9}$$

When $n_T \neq n_E$, for example $n_T = 12$ and $n_E = 24$ (months), $\mathcal{D}$ would be split into:

$$T := \{D_{2014}, D_{2015}, D_{2016}, D_{2017}\}$$
$$E := \{D_{2015:2016}, D_{2016:2017}, D_{2017:2018}, D_{2018:2019}\}. \tag{10}$$

Additional Temporal Luck results. In §V we showed the effect of having different training and testing slots. We provide additional results for DREBIN, DEEPDREBIN, MALSCAN and RAMDA in Figure 9.

F. VirusTotal Threshold

In §VII, we discussed hypothesized the impact VTT has on classification performance. To show this, we sampled both D_A and D_T for VTT 2, 4 and 15 and comapared it agains tthe original. D_A uses a VTT=15 and D_T uses a VTT=4. §A contains details regarding how we imitated the sampling for both datasets by discarding samples that did not exist in AndroZoo when the original dataset was sampled. Figure 10 shows the performance across all 5 different classifiers.

G. Dataset Size

In §VIII, we discussed using different sampling strategies for determining the minimum sample size. Figure 11 shows the performance of each strategy for all 5 classifiers.

H. Prevalence Study

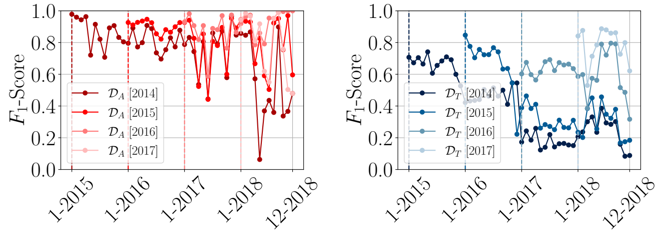
We begin our analysis by conducting a prevalence study of benchmark datasets used for Android malware detection. Given the vastness of the domain, with over 81,000 results returned when searching "Android Malware" on Google Scholar, we narrow our focus. We begin with two of the earliest Android malware datasets, Malgenome [98] and Drebin [18]. Both these datasets has over 3,000 citations and was published in a top security venue (IEEE S&P 2014, 2016 respectively). In total, we collected 4,995 unique papers that cited either Drebin and Malgenome.

To ensure we only survey relevant high quality papers, we select works that was published in one of the top four security venues: USENIX Security Symposium, ACM CCS, IEEE S&P, and NDSS. This helps us filter out papers that do not address security topics or apply machine learning to security related research problems. After applying the top-4 venue filtering, we ended up with 133 papers.

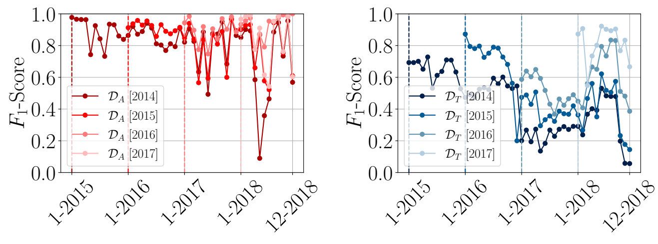
However, not all notable works was published in the top security venues. To include relevant works that may have been missed from the above criteria, we include the top 100 results of Google Scholars using the search query "Android Malware Detection" on July 10th 2025.

This forms an initial list of 233 papers which we review to identify datasets that were used for experimental evaluation. For each paper, we read through the relevant sections related to dataset curation and evaluation. If a paper did not use a publicly available dataset but collected its own samples, we documented the method of collection. We excluded papers not directly relevant to Android malware detection, including those focused on Windows PE malware, PDF malware, Android UI/UX elements, or privacy policies.

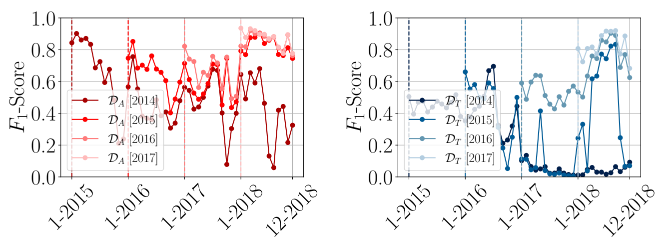
From this initial list, we found 35 Android malware datasets used in prior works. To capture the prevalence of these datasets, we performed forward citations (i.e. works that cited the dataset) and applied the top four security venue filter to



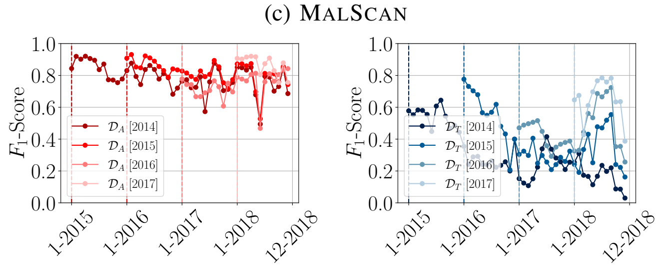
(a) DREBIN



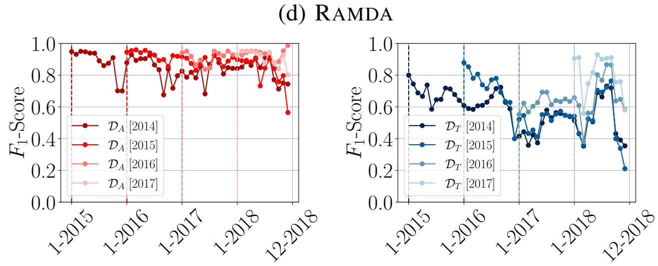
(b) DEEPDREBIN



(c) MALSCAN

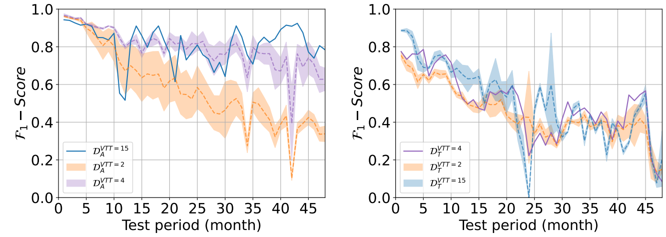


(d) RAMDA

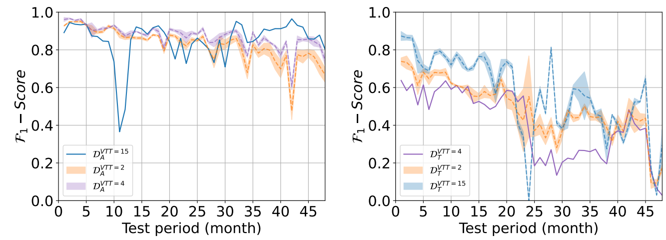


(e) HCC

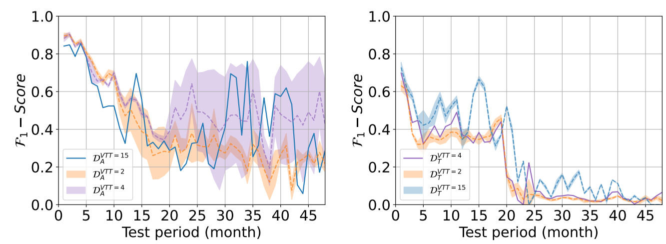
Figure 9: **Impact of Temporal Luck on performance.** Each plot refers to a specific model and dataset. Each line indicates the F_1-Score of a model trained on a different year (between 2014 and 2017). It can be seen that training on different years within the same dataset can yield different performance profiles. The issue is evident on D_T, but also present in D_A.



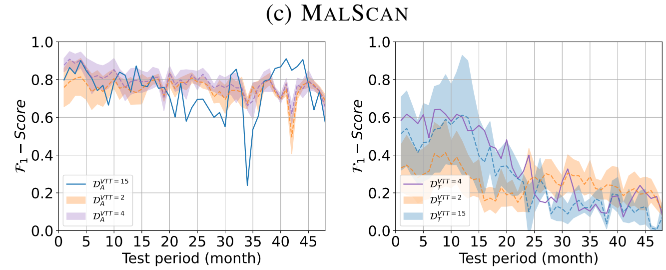
(a) DREBIN



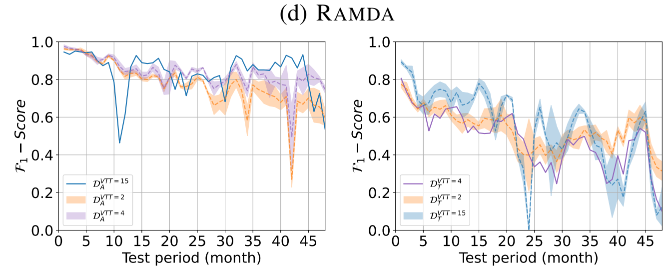
(b) DEEPDREBIN



(c) MALSCAN



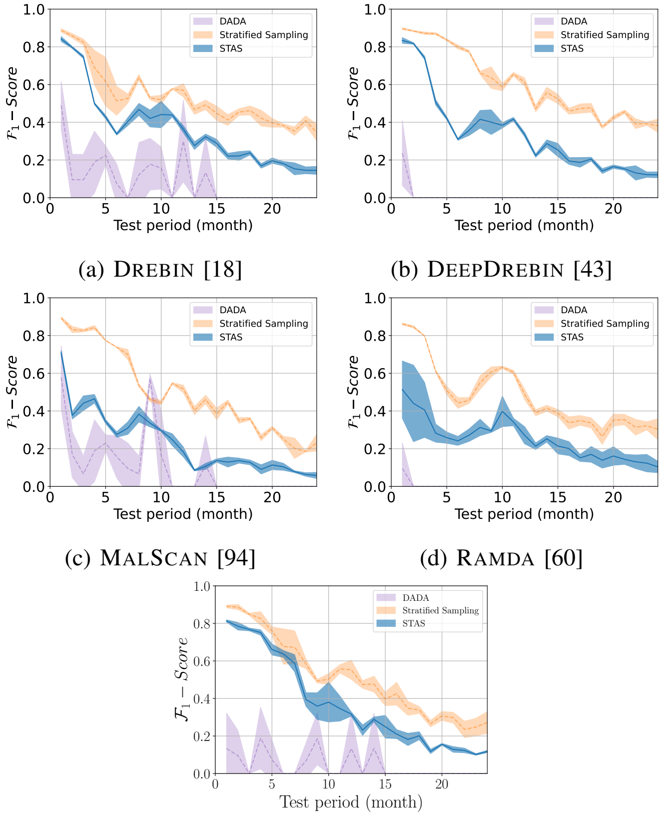
(d) RAMDA



(e) HCC

Figure 10: **F_1-Score of classifier for D_T and D_A sampled with a VTT=2, VTT=4 and VTT=15**

obtain a new list of papers. We review each paper in a similar fashion as above, adding details about newly curated datasets or incrementing prevalence count of existing ones. We repeat this process until there are no new datasets added. To retrieve forward citations, we used SerpApi [81], a Google Scholar API, and verified the publication venues using DBLP [57], an online bibliographic database for computer science research. Overall, we found 42 unique Android malware datasets. An



(a) DREBIN [18] (b) DEEPDREBIN [43]

(c) MALSCAN [94] (d) RAMDA [60]

(e) HCC [25]

Figure 11: **Comparison of F_1-Score between datasets sampled using DADA versus STAS sampling**. The F_1-Score of DADA is very poor and highly inconsistent, therefore not appropriate for temporal evaluations.

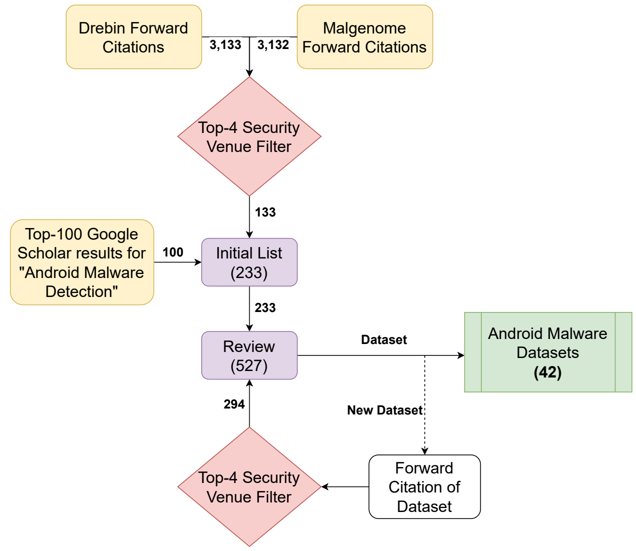


Figure 12: Survey pipeline with queries performed in July 2025. Starting from Drebin and Malgenome, we reviewed papers that are published from one of the top-4 security venues. We also reviewed top-100 Google Scholar results for ”Android malware detection”. We performed forward citation of each new dataset to ensure we did not ‘miss’ any datasets. After reviewing 527 papers, we found 42 Android malware datasets.

overview of the survey methodology is presented in Figure 12.

For all 42 datasets, we assessed them based on how well they considered each of our factors even if they do not fully satisfy our recommendations. We provide details regarding each factor criteria below.

Timestamp Type. In §IV we show the importance of timestamps when curating a dataset. Therefore, if a dataset explicitly mentions the timestamp used for sampling and evaluation, we consider that they recognized timestamps when sampling . If it only mentioned timestamp for temporal order, we consider it partly satisfied. We noticed some datasets use a combination of other datasets that contains this information, hence mark them as orange. Finally, if a dataset never mentions timestamps and can not be easily inferred by the dataset they chose, then it is marked as red.

App Markets. Ideally datasets should sample from a single market/source §VI. Given that AndroZoo crawls from multiple markets, we make the distinction that if datasets was explicitly crawled from a single source or single market from AndroZoo, it satisfies our market recommendation and is marked green. If a dataset only samples from AndroZoo but does not filter out specific markets, we mark it as blue. For all other datasets that sample from multiple sources (including combination of datasets), we mark it as red. Finally, gray indicates market/-source information was not provided in the paper.

Temporal Luck. We check if the paper that each dataset was proposed considers temporal splits during evaluation. If they do, we mark the dataset as green, otherwise it is red. Note we do not consider temporal evaluation or K-fold Cross-validation alone as sufficient.

Dataset Size. For each dataset, we compute the statistical significant dataset size for each time frame and VTT chosen. Here, we only check for statistical significance and do not consider class ratio where goodware should be nine times malware amount. If both classess are statistically significant, we mark the dataset as green. If only one class is, we mark it as blue. If neither satisfies the minimum statistical significance, we mark it as red. If a dataset does not mention the dataset size or the minimum is undeterminable, we mark it as gray.

VirusTotal Threshold. In §VII, we recommend for a VTT=2. If a dataset uses VTT=2, we mark this as green. If a dataset does not use VirusTotal Threshold and instead opts for pseudo-labels or manual labeling, we mark it as blue as it does not satisfy nor violate our recommendation. If a dataset uses a VTT above 2 or mixes multiple different labeling methods (by combining different datasets), we mark this as red. Lastly, if labeling method was not mentioned, we mark it as gray.

Dataset	**# Malware**	**# Goodware**	**Time Frame**	**Hashes**	**C3** [78]	**Source**	**Top-4 Prevalence**	**Prevalence**	**Venue**
Malgenome [98]	1,260	0	2010-2011	○	○		3	20	S&P 2012
Liu et al. [68]	1,536	28,548	2012	○	●	✚, [98]	0	1	MobileCloud 2014
Andrubis [66]	41.14%	27.90%	2010-2014	○	◐	GP,✚, [18], [98]	0	3	ESORICS 2014
DroidDolphin [93]	32,000	32,000	-	○	◐	GP	0	1	RACS 2014
Drebin [18]	5,560	123,453	2010-2012	●	●	[98], AZ, GP	9	46	NDSS 2014
Triggerscope [37]	-	9,582	-	○	○	GP	1	1	S&P 2016
ICCDetector [95]	5,264	12,026	2012	○	◐	GP, [18]	0	1	TIFS 2016
Anastasia [35]	18,677	11,187	2009-2015	○	◐	[18], [98], VT	0	1	NTMS 2016
Droidnative [10]	5,490	3,732	-	○	◐	[18], [98]	0	1	Computers & Security 2017
Piggybacking [63]	1,497	0	2009-2014	●	○	GP, ✚, [98]	0	1	TIFS 2017
Shen et al. [82]	4,699	3,899	2014-2016	○	◐	GP, [98]	0	1	ICDCS 2017
AMD [92]	24,553	0	2010-2016	○	○	GP, VS, ✚	7	29	DIMVA 2017
MamaDroid [72]	35,500	8,500	2010-2016	◐	●	[18], VS, GP	0	2	NDSS 2017
Faldroid [33]	8,407	6,593	-	●	◐	VS	1	1	TIFS 2018
EnDroid [34]	55,213	58,806	2015-2018	○	◐	GP, AZ, [18]	0	1	IEEE Access 2018
Flowcog [77]	1,500	4,500	-	●	◐	GP, [18]	1	1	USENIX 2018
Maldozer [52]	20,089	37,627	-	○	◐	VS, [2], [18], [98]	0	1	Digital Investigation 2018
CICAndMal2017 [56]	429	5,065	2015-2017	○	◐	VT, [2], [9], [42], [55]	0	4	ICCST 2018
Hasegawa et al. [46]	5,000	2,000	-	○	◐	✚, [18], [98]	0	1	CSPA 2018
Li et al. [61]	54,694	310,926	2012-2014	◐	●	GP, ✚	0	1	IEEE Industrial Inf. 2018
Marvin [65]	≈15,000	≈120,000	2012-2014	◐	●	GP, ✚	1	2	ASE 2019
Malscan [94]	15,430	15,285	2011-2018	●	◐	AZ	1	1	ASE 2019
Tesseract [78]	12,753	116,993	2014-2016	●	●	AZ	3	3	USENIX 2019
DroidCat [21]	135	136	2015	◐	◐	GP, [98]	0	1	TIFS 2019
Omnidroid [73]	11,000	11,000	-	●	◐	AZ	0	1	Information Fusion 2019
Pierazzi et al. [79]	17,635	152,632	2017-2018	●	●	AZ	2	2	S&P 2020
CICMalDroid2020 [70]	17,341	0	2018	○	○	VT, [2], [9], [52], [84], [92]	0	3	DASC 2020
APIGraph [97]	32,089	290,505	2012-2018	●	●	VS, VT, [92], AZ	2	2	CCS 2020
Dhalaria et al. [30]	1,747	1,800	-	●	◐	VT, ✚	0	1	IMAI 2020
Kronodroid [44]	28,745	35,256	2008-2020	●	◐	[43], [92] VT, VS	0	2	Elsevier CoSe 2021
AndroCT [64]	17,679	18,277	2019-2019	●	◐	AZ, VS, GP	0	1	MSR 2021
VTAz [39]	5,016	4,987	2017-2020	◐	◐	VT, AZ	0	1	Computer Security 2021
Sun et al. [85]	12,685	49,045	2014	○	●	GP, [92]	0	1	TMIS 2022
DroidMalware Detector [51]	7,752	6,634	-	○	◐	AZ, VS, [18], [24]	0	1	Expert Systems with Applications 2022
Malradar [89]	4,534	0	2014-2021	●	○		0	1	ACM MACS 2022
Madroid [90]	35,121	39,571	2010-2023	●	◐	AZ, [12], [94], [98]	0	2	ASE 2022
AndroOBFS [54]	16,279	0	2018-2020	●	○	AZ, VS	0	1	MSR 2022
Transcendent [19]	26,387	232,843	2014-2018	●	●	AZ	2	2	S&P 2022
Finer [47]	4,742	12,807	2017-2019	●	◐	-	1	1	CCS 2023
Androzoo [25]	10,200	89,853	2019-2021	◐	●	AZ	1	1	USENIX 2023
Heng et al. [58]	22,975	21,399	-	◐	◐	AZ, [18], [33]	1	1	USENIX 2023
Jigsaw [96]	14,775	134,759	2015-2016	◐	●	AZ	0	1	S&P 2023
Maldroid-DS [15]	47,971	0	2010-2024	●	○	[18], [53], [56], [92], VS	0	1	IEEE Access 2024

Table IX: Breakdown of existing Android malware datasets discovered in our survey arranged by publication date. AZ samples crawled from Androzoo repository [14], GP samples crawled directly from Google Play, VS samples from VirusShare, ✚ samples crawled from other sources.