# Explainable AI for Android Malware Detection: Towards Understanding Why the Models Perform So Well?

Yue Liu[†], Chakkrit Tantithamthavorn[†*], Li Li[†*], and Yepang Liu[‡]
[†]Faculty of Information Technology, Monash University, Melbourne, Australia.
[‡]Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China
Email: {yue.liu1, chakkrit, li.li}@monash.edu, liuyp1@sustech.edu.cn

*Abstract*—**Machine learning (ML)-based Android malware detection has been one of the most popular research topics in the mobile security community. An increasing number of research studies have demonstrated that machine learning is an effective and promising approach for malware detection, and some works have even claimed that their proposed models could achieve 99% detection accuracy, leaving little room for further improvement. However, numerous prior studies have suggested that unrealistic experimental designs bring substantial biases, resulting in over-optimistic performance in malware detection. Unlike previous research that examined the detection performance of ML classifiers to locate the causes, this study employs Explainable AI (XAI) approaches to explore what ML-based models learned during the training process, inspecting and interpreting why ML-based malware classifiers perform so well under unrealistic experimental settings. We discover that temporal sample inconsistency in the training dataset brings over-optimistic classification performance (up to 99% F1 score and accuracy). Importantly, our results indicate that ML models classify malware based on temporal differences between malware and benign, rather than the actual malicious behaviors. Our evaluation also confirms the fact that unrealistic experimental designs lead to not only unrealistic detection performance but also poor reliability, posing a significant obstacle to real-world applications. These findings suggest that XAI approaches should be used to help practitioners/researchers better understand how do AI/ML models (i.e., malware detection) work—not just focusing on accuracy improvement.**

## I. Introduction

DESPITE significant and continuous improvements of cybersecurity mechanisms, malware remains one of the most serious threats in cyberspace. According to McAfee's report [1], the total number of malware samples reached about 1.5 billion in 2020, with a gradual increase. Everywhere malware is eroding cyberspace, spawning a slew of subtypes such as mobile malware, MacOS malware, IoT malware, and Coin Miner malware, all of which are causing massive financial losses to both individuals and industries.

To this end, machine learning (ML) and deep learning (DL)-based malware detection has received significant research attention in recent years. Specifically, when trained on a large set of data, the built model can distinguish malware from benign samples automatically. Based on the surveyed results on malware detection by [2], [3], [4], we discovered that ML/DL techniques can generally achieve quite high performance, with detection accuracy reaching up to 99%, leaving little room for future research. However, these high-performance approaches appear less viable in practice, as malware defences remain a challenging problem to tackle and malicious applications continue to pose a growing threat to people [5], [6], [7], [8], [9].

Previous studies have reported that existing ML-based malware detection models are susceptible to experimental biases, which could produce unrealistic model performance [10]. Particularly, Pendlebury *et al.* [11] found that the performance of ML-based malware detection models [12] is substantially decreased from 90% to 58% (F-score) after removing experimental biases (e.g., spatial bias, concept drift). According to our analysis of the recent research (see Table I) [2], we discovered that 29 out of the 30 reviewed relevant studies published between 2014 and 2020 do not consider the realistic experimental design, achieving high accuracy of the ML-based Android malware detection (i.e., over 99% of F1).

While accuracy improvement is the primary focus of prior work in ML-based Android malware detection, little research is known about *why the models perform so well*. Recent research has discovered that the high accuracy of Android malware detection could be due to various sources of experimental biases [11]. However, prior studies still focus only on the predictions (i.e., accuracy) without investigating whether the models can detect malware based on the actual malware-related characteristics or not. For example, a highly-accurate classifier could correctly classify an image as a dog using background features, which are irrelevant to the dog at all. Similarly, ML-based malware detection models could correctly classify an app as malware based on deprecated features (e.g., GET_TASKS Permission), which may be irrelevant to the actual malicious/benign behaviours. Therefore, there is a critical need to examine the explanations of the classifications of ML-based Android malware detection as well (i.e., why does the model classify an app as malware?).

*In this paper*, we investigate the impact of the temporal inconsistency on the accuracy and the explanations of the ML-based malware detection. The **temporal inconsistency** refers to an unrealistic experimental setup where malware and benign samples are randomly chosen without considering

*The Corresponding author.

**Table I:** A list of ML-based Android malware detection that achieves high performance (over 96% of F1) under an unrealistic experimental setup due to the temporal inconsistency. That means malware samples were chosen from different periods of the benign samples.

| Paper | Malware sources | Malware periods | Benign sources | Benign periods | Performance |
|---|---|---|---|---|---|
| MalDozer [13] | Drebin, Genome, Virushare, Contagio | 2011 - 2017 | Google Play | 2017 | 0.96 F1 score |
| DeepRefiner [14] | VirusShare | <2015 | Google Play | 2016 | 0.977 accuracy |
| Su *et al.* [15] | Drebin, Genome, Contagio | 2011 - 2016 | Google Play | 2016 | 0.995 accuracy, 0.975 F1 score |
| Khoda *et al.* [16] | Drebin, Genome | 2010 - 2016 | Google Play | 2019 | 0.987 accuracy, 0.985 F1 score |
| Fan *et al.* [17] | Drebin, Genome, AMD, etc. | 2010 - 2017 | Google Play | 2019 - 2020 | 0.996 precision, 0.977 F1 score |
| DroidDeep [18] | Drebin, Genome, etc. | 2011 - 2017 | Google Play | 2020 | 0.995 accuracy |

the time period (meaning that malware samples were chosen from 2010, while benign samples were chosen from 2020, see Table I). In our experiment, we collected a total of 165,000 Android applications (i.e., 33,000 malware and 132,000 benign applications) that span ten years (2010-2020). Then, we focus on the three state-of-the-art Explainable ML-based Android malware detection models (i.e., Drebin [12], XMal [19] and Fan *et al.* [17]).

Our experimental results reveal several important findings:

- Temporal inconsistencies between malware and benign in the data significantly increase the detection performance.
- When a temporal inconsistency is introduced in the datasets, the explanations of the ML-based Android malware detection indicate that the models can correctly predict malware based on the temporal-related features, instead of the actual characteristics of malicious and benign behaviors.
- Although adjusting the experimental setups like feature sets and malware/benign rates, temporal inconsistencies still unrealistically increase the performance of the ML-based malware detection approaches.

These findings suggest that security analysts should use explainable AI approaches to better understand the models (why the models predict an app as malware or benign?) to better select the most appropriate malware detection models when deciding to deploy them in production.

**Novelty.** To the best of our knowledge, this paper is the first to:

- Investigate the impact of temporal inconsistency in Android Malware Detection
- Employ Explainable AI approaches to understand why ML-based malware detection approaches perform so well under temporal inconsistency.

**Open Science.** To support the open science initiative, we publish the studied dataset and a replication package, which is publicly available in GitHub.[1]

**Paper Organization.** The remainder of this paper is structured as follows: Section 2 presents the background and related work. Section 3 details our study design. Section 4 provides our experimental results and analysis. Section 5 discusses the study's limitations and potential threats to its validity. Finally, Section 6 concludes the paper.

## II. Background and Related Work

Researchers raised concerns that many ML-based malware detection techniques are over-optimistic [10], [11], [20], [21]. In addition, these malware detection approaches were usually black-box models [4]. Thus, security analysts often asked questions, e.g., How can we trust the predictions of the so-accurate ML-based malware detection models? How can we understand whether we are selecting a proper model before deployment? To address this challenge, several studies proposed various approaches to explain the predictions of ML-based malware detection models [12], [19], [17], [22], [23] (i.e., local explainability). Below, we introduce our motivational example and summarize the three state-of-the-art explainable ML-based malware detection techniques.

### *A. Motivation*

Recent research raised concerns that the accuracy of the ML-based malware detection approaches is nearly perfect [10], [4]. Liu *et al.* [4] systematically reviewed 132 existing research studies on ML/DL-based Android malware defence approaches. Their review results show that most ML-based malware detection approaches achieve an accuracy/F1 measure of 0.98, or even higher. Moreover, 33 out of 132 surveyed papers present up to 0.99 accuracy/F1 measure, indicating that most ML-based malware detection approaches achieve nearly perfect predictions. While existing ML-based malware detection approaches are extremely accurate, it remains unclear why such approaches are so accurate, which still casts some doubt on the research community.

As suggested by prior studies [11], [21], [24], [25], [26], the evolution of both the Android platform and Android applications leads to a severe model aging problem (or called time decay, model degradation, and concept drift). Specifically, malware detection approaches perform poorly on new malware samples. For example, TESSERACT [11] reproduced three state-of-the-art ML-based malware detectors which achieved a high F1 score (up to 0.98), but they found that the performance dropped significantly to 30% in a time-aware setting (i.e., older apps were used for training and newer ones for testing). **It is still unknown why these Android malware detection approaches perform so well on the original data (i.e., 0.98 F1 score). In other words, there is still uncertainty about whether these approaches correctly identify samples based on malware-related characteristics.**

Although the majority of research studies surveyed by [4] did not include information about the time period of collected

[1] https://github.com/yueyueL/XAIforAndroidMalware

experimental samples, six relevant primary studies were found, as shown in Table I. It is interesting to observe that these six primary studies collected malware and benign samples from different time periods (i.e., malware samples were older while benign samples were newer). This result may be explained by the fact that most malware datasets are usually not maintained or updated after being released, whereas recent benign samples are available via Google Play or third-party markets [4], [27], [28]. However, prior work [11], [21], [24], [25], [29], [26], [30] has proven that Android malware samples evolve and exhibit distinct characteristics over time. As a result, it may lead to unfair predictions, as malware samples and benign samples are collected from distinct time periods. To the best of our knowledge, no prior literature has studied whether this unfair setting provides a reliable evaluation result for ML/DL-based Android malware detection. In this study, we use **temporal inconsistency** to define this problem, which is caused by temporally inconsistent distributions of malware samples and benign samples.

To evaluate the impacts of temporal inconsistency on Android malware detection models, we consider three well-known malware detection approaches using explainable machine learning techniques (i.e., Drebin [12], XMal [19] and Fan *et al.* [17]). First, we would like to stress that we make no specific criticisms of these three approaches. Because they are available and provide consistent baselines, we choose these three explainable methodologies for our evaluation.

### B. Drebin with Linear Support Vector Machine

Arp *et al.* [12] leveraged an interpretable ML technique, i.e., a linear Support Vector Machine (SVM) to classify if an unknown application is malware or benign. To train an SVM-based malware detection model, a linear SVM technique determines a hyperplane that separates both malware and benign classes with maximal margin based on the feature vectors of malware and benign applications in the training data. To detect the malicious activities of an unknown application, Drebin requires a comprehensive yet lightweight representation of mobile apps. In particular, Drebin extracts eight feature sets from two main sources. First, the *manifest* file (i.e., AndroidManifest.xml) is used to store information of the requested hardware components (e.g., camera), the requested permissions (e.g., SEND_SMS), the list of used Android components (e.g., activities, services, content providers, and broadcast receivers), and filtered intents (e.g., BOOT_COMPLETED). Second, the disassembled *dex* code is used to store information of the restricted API calls, used permissions, suspicious API calls, and network addresses. Then, this information is used to generate a vector representation using a one-hot encoding technique where 1 indicates that an application $x$ contains a feature $x_i$, otherwise 0. Once the SVM models are trained, the SVM-based malware detection model is applied to classify if an unknown application in testing data is considered as malware or benign. Finally, Drebin generates an explanation of each prediction using the multiplication ($w_i = w * v_i$) of the feature weights ($w$) of the linear classifier and the actual feature value ($v$) of that test instance ($i$).

### C. XMal with Attention Mechanism

Wu *et al.* [19] leveraged a multi-layer perceptron (MLP) with the attention mechanism for malware classification, while being able to locally explain the prediction. Similar to the Debrin [12]'s approach, the XMal approach leverages feature sets related to API calls and permissions. Since there exists a large number of possible features (i.e., 20,000+), the XMal approach selects only the top-154 effective features (including 94 API calls and 60 permissions) for model training. The XMal approach consists of two layers: the attention layer and the multi-layer perceptron (MLP). First, a feature vector is generated using a one-hot encoding technique with a dimension of 158. Then, the feature vector is fed into the attention layer. The attention layer leverages the attention mechanism proposed by Bahdanau *et al.* [31], which is used to capture the relationship between the features in the input sequence and the next output features, allowing models to retain all the information of the input sequence. Formally, the attention vector $\alpha_i = (\alpha_i^{(1)}, ..., \alpha_i^{(j)})$ which represents the attention weight of the $j^{\text{th}}$ feature of the $i^{\text{th}}$ test instance is computed through a softmax function: $\alpha_i^{(j)} = \frac{exp(e_i^{(j)})}{\sum_{k=1}^{n} exp(e_i^{(k)})}$, where $\alpha_i^{(j)}$ denotes the attention weight of the $j^{\text{th}}$ feature for the $i^{\text{th}}$ test instance, where $e_i^{(k)})$ is the feature vector of the $i^{\text{th}}$ test instance. Then, the attention vector is fed into the Multi-layer Perceptron (MLP) layer to map the feature weights into the binary classification. Finally, the explanation of each prediction is generated based on the attention weights to indicate which features contribute the most to the prediction.

### D. *Fan* et al. *with Model-Agnostic Explainable Approaches*

Fan *et al.* [17] assessed five different local and model-agnostic explanation approaches (i.g.,LIME [32], Anchor [33], LORE [34], SHAP [35] and LEMNA [36]) for Android malware analysis. Unlike model-specific explanation approaches, model-agnostic explanation approaches can explain any machine learning model. For example, LIME explains a prediction by approximating the decision boundary of any black-box classifier by a simple weighted linear regression model. Thus, Fan *et al.* [17] evaluated the stability, robustness and effectiveness of model-agnostic explanation approaches on several different malware classifiers (i.e., multilayer perceptron (MLP), random forest (RF), and support vector machines (SVM)).

### E. AI/ML-based Experimental Bias

Researchers have already realized the problem of unrealistic performance in Android malware detection and identified many pitfalls related to high performance, including temporal biases (i.e., time decay or evolved malware) between training and testing data [37], [38], [24], [25], [11], [39], [21], [40], [26], [41], inappropriate malware rate [42], [11], [24], [43], sampling duplication [44], etc. These studies have highlighted a correlation between over-optimistic performance and specific unrealistic experimental settings. For example, when studying temporal biases, researchers have experimentally found that applications that alter or update over time will cause the trained models to perform poorly on future testing samples [21]. As a result, the actual performance of the proposed ML models might not be as high as the one reported. Our work takes the initial attempt towards understanding the inner logic of ML-based malware models under unrealistic settings to empirically confirm that learning models could be misled by pitfalls instead of solving the actual task based on benign and malicious behaviors.

## III. Study Design

### A. Goal, Motivation, and Research Questions

The goal of this paper is to perform a detailed model inspection analysis on the explanations generated by three explainable ML-based malware detection techniques (i.e., Drebin [12], XMal [19], and Fan *et al.* [17]). Such a detailed model inspection analysis could help security analysts better select the most appropriate malware detection models when deciding to deploy in production and help researchers better understand the potential risks associated with unrealistic experimental setups. To achieve this goal, we aim to address the following three research questions.

*(RQ1) What is the impact of temporal inconsistency on the performance of ML-based malware detection approaches?*

**Motivation**. Numerous research studies evaluate malware classification performance using temporally inconsistent datasets, as we discussed before. We formulate this research question to ascertain the effect of temporal inconsistency on the performance of machine learning-based malware detection techniques. Through the replication of three high-profile ML-based malware detection approaches (i.e., Drebin [12], XMal [19] and Fan *et al.* [17]), we can confirm whether temporal inconsistency results in over-optimistic detection performance.

*(RQ2) Why does temporal inconsistency make ML-based malware detection approaches perform so well?*

**Motivation**. Only examining classification performance metrics (e.g., accuracy and F1 score) still does not determine what the model is based on to make accurate predictions. Drebin [12], XMal [19] and Fan *et al.* [17] approaches are designed to achieve high accuracy, while being explainable to security analysts. Therefore, such explainable malware detection techniques allow security analysts to better understand what features contribute to the predictions. Unfortunately, these three studies have not performed a model inspection analysis on the generated explanations to better understand if the models behave correctly or not. Such a lack of detailed model inspection analysis for the ML-based malware detection models could lead to inappropriate model selection when deploying them in production (i.e., practitioners still do not know which models to be deployed given the same highly accurate models). Thus, we formulate this research question to analyze the explanations generated by these three approaches to better understand why such ML-based malware detection approaches under temporal inconsistency are highly accurate.

*(RQ3) How sensitive is the impact of the temporal inconsistency on the accuracy and explanation of the ML-based malware detection approaches?*

**Motivation**. Prior studies [45], [46], [47], [10] raise concerns that the experimental components often have a large impact on the accuracy and explanations of defect prediction models (e.g., data quality [48], class imbalance [49], parameter settings [50], model validation techniques [51]). Similar to ML-based malware detection studies, different studies also use different experimental components [12], [19], [2]. Yet, little is known about the impact of the experimental components on the accuracy and explanation of explainable malware detection approaches. As a result, we formulate this research question to gain a better understanding of the impacts of temporal inconsistency under different experimental settings.

### B. Experimental setup

To address our research questions, our experiment consists of the following steps: (1) data collection; (2) feature extraction; (3) model training & model evaluation; and (4) model explanations. Figure 1 illustrates the overview of our experiment. We describe each step below.

*1) Data Collection:* Generally, ML-based malware detection is formulated as a binary classification task (i.e., classifying whether an application is considered as Malware or Benign). Thus, it requires samples from two distinct classes (i.e., Malware and Benign apps). To do so, we download Android applications from the AndroZoo corpus [52]. The AndroZoo corpus consists of a collection of more than 15 million Android applications published between 2010 and 2021, together with the ground-truth labels provided by the VirusTotal software [11], [53], [54]. Since the number of applications in the AndroZoo corpus is too large to be studied (15 million), we randomly select a subset of applications in the AndroZoo corpus.

To ensure that our random sample is representative of the population of the AndroZoo corpus, we decide to maintain the same malware ratio as the AndroZoo corpus (i.e., a malware ratio of 17.3%). Thus, our studied dataset contains a total of 165,000 Android applications (i.e., 33,000 malware and 132,000 benign applications) that span across a 10-year period (2010-2020).

*2) Feature Extraction:* Similar to prior studies [12], [19], we use the same feature extraction approach to generate feature vectors in order to capture the characteristics of Android applications. Thus, we use a reverse engineering approach to extract the feature set of Android applications using Androguard,[2]. Androguard is a common open-source tool for static analysis to exploit features like permissions, API calls and activities.

[2] https://code.google.com/archive/p/androguard/

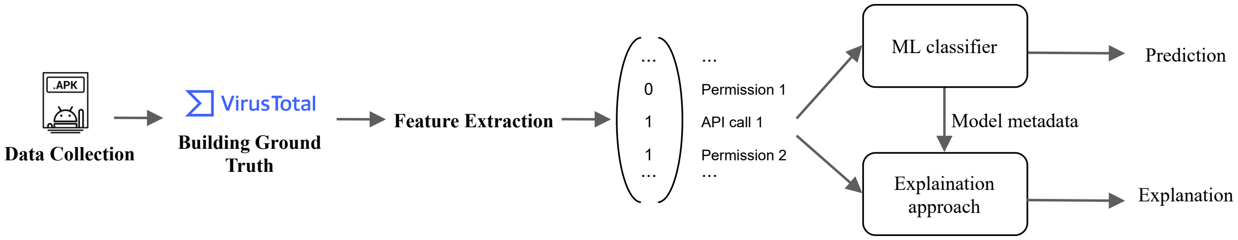

**Fig. 1:** An overview of our experiment: the apps are collected from AndroZoo and labelled by VirusTotal; reverse engineering tools are used to extract features; feature vectors are fed into an ML classifier to generate prediction and explanation.

For Drebin, we extract a total of eight feature sets from two main sources, i.e., (1) the *manifest* file (i.e., AndroidManifest.xml), which stores information of the requested hardware components, the requested permissions, the list of used Android components, and filtered intents; and (2) the disassembled *dex* code which stores information of the restricted API calls, used permissions, suspicious API calls, and network addresses. For Xmal, we extract the same set of 154 features (including 94 API calls and 60 permissions) for model training.[3] For Fan *et al.*, we use the feature set with XMal since the detailed feature lists are not publicly available.[4] Then, feature information is used to generate a vector representation using a one-hot encoding technique where one indicates that an application $x$ contains a feature $x_i$, otherwise zero.

*3) Model Training & Evaluation:* According to Wu *et al.* [19], 10-fold cross-validation (CV) is one of the most commonly-used model validation techniques for ML-based malware detection. Thus, we use a 10-fold CV for model training and model evaluation. 10-fold CV splits a dataset into K partitions, with one partition used for model evaluation and the remaining partitions used for model training. Then, the process is repeated ten times, and each testing performance is recorded to ensure the stability of the models [51]. For Drebin, we train the model using a linear support vector machine (SVM). For XMal, we train the model using an attention-based multi-layer perceptron (MLP). For Fan *et al.*, we train four different ML models (i.e., MLP, KNN, RF and SVM) with the same settings as those used in the original paper.

*4) Model Explainability (i.e., Most Important Features):* Finally, we generate explanations from the Drebin, XMal and Fan *et al.* approaches. For Drebin with SVM models, we generate an explanation of each prediction using the multiplication ($w_i = w * v_i$) of the feature weights ($w$) of the linear classifier and the actual feature value ($v$) of that test instance ($i$). For the XMal approach, we generate an explanation of each prediction based on the attention weights to indicate which features contribute the most to the prediction. For the Fan *et al.* approach, we generate an explanation of each prediction based on the LIME approach to indicate which features contribute the most to the prediction. Note that we don't focus on other model-agnostic explanation approaches discussed in Fan *et al.* [17] since their experimental results demonstrate that LIME provides a better explanation for ML-based malware detection approaches. Because we employ 10-fold cross-validation, we record explanations for testing data at each run.

## IV. Experimental Results

In this section, we present the approach and the results of our three research questions.

### *RQ1: What is the impact of temporal inconsistency on the performance of ML-based malware detection approaches?*

In the first research question, we are interested in checking the impacts of temporal inconsistency on detection performance.

**Experimental Setup**. In this work, we decide to replicate three prior studies, namely the Drebin, XMal and Fan *et al.* approaches, which have been considered the most representative ones available in the community. For each of these three approaches, we further resort to five training datasets (i.e., settings) to highlight their performances and examine the potential impacts brought by the different settings. Given a dataset, we first extract the features following the strategies provided by these three approaches, respectively. After that, 10-fold cross-validation will be leveraged to assess their performances. In this work, we evaluate the classification performance through four metrics: Accuracy, F1 measure, Precision, and Recall.

The five settings are detailed as follows.

- **Baseline.** For the default setting (hereinafter referred to as the baseline), we select all the malicious apps collected in this work to form the training dataset. As discussed in Section III-B1, we have prepared 33,000 malicious apps that are released at times ranging from 2010 to 2020, with each year containing 3,000 samples. To form a balanced training dataset, as highlighted in Table II, we supplement the training dataset by further adding 33,000 benign apps (i.e., with also 3,000 samples per year), which are also randomly selected from the apps collected in this work.
- **Variant 1.** The first variant keeps most of the configurations in the Baseline setting except that, in this case, only the samples in the latest three years (i.e., years 2018-2020, as highlighted in Table II) are considered. In

[3] The original paper claimed 158 features, but they only provided a feature set with 154 features on their personal page.

[4] The original paper claimed 296 features, including 259 API calls, 22 permissions and 3 intents.

|  | Time periods of experimental samples |  |  |  |  |  |  |  |  |  |  |  | Drebin |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Setup | Year | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | Accuracy | F1 | Precision | Recall |
| Baseline (Temporal consistent) | Malware: | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | 0.9239 | 0.9243 | 0.9203 | 0.9283 |
|  | Benign: | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |  |  |  |  |
| Variant1 (Temporal consistent) | Malware: | ▨ | ▨ | ▨ |  |  |  |  |  |  |  |  | 0.9263 | 0.9259 | 0.9311 | 0.9208 |
|  | Benign: | ▨ | ▨ | ▨ |  |  |  |  |  |  |  |  |  |  |  |  |
| Variant2 (Temporal consistent) | Malware: |  |  |  |  |  |  |  |  | ▨ | ▨ | ▨ | 0.9535 | 0.9538 | 0.9470 | 0.9608 |
|  | Benign: |  |  |  |  |  |  |  |  | ▨ | ▨ | ▨ |  |  |  |  |
| Variant3 (Temporal inconsistent) | Malware: |  |  |  |  |  |  |  |  | ▨ | ▨ | ▨ | 0.9911 | 0.9911 | 0.9941 | 0.9881 |
|  | Benign: | ▨ | ▨ | ▨ |  |  |  |  |  |  |  |  |  |  |  |  |
| Variant4 (Temporal inconsistent) | Malware: | ▨ | ▨ | ▨ |  |  |  |  |  |  |  |  | 0.9927 | 0.9927 | 0.9907 | 0.9948 |
|  | Benign: |  |  |  |  |  |  |  |  | ▨ | ▨ | ▨ |  |  |  |  |

**Table II:** Experimental settings and part of evaluation results of the replication study. The full evaluation results can be found on our online supplementary. The second column (i.e., sample dates) includes 11 years of app samples ranging from 2010 to 2020. The black cell represents that the apps (i.e., all the 3,000 apps for the malware set while randomly selected 3,000 apps from the goodware set) in the corresponding time frame are selected for training.

other words, variant 1 forms a balanced training dataset including 9,000 malware and 9,000 goodware.

- **Variant 2.** Similar to the setting of Variant 1, in the second variant, the app samples in the first three years (i.e., years 2010-2012, as highlighted in Table II) are considered for training.
- **Variant 3.** The readers may have observed that the first two variants (Variants 1-2) have kept the training apps collected from the same period of time. In the third variant, we form the training dataset by collecting the goodware and malware samples from two different periods, i.e., benign samples from the first three years (i.e., years 2010-2012) while malicious samples from the latest three years (i.e., years 2018-2020).
- **Variant 4.** The last variant is essentially equivalent to Variant 3 except that, in this case, the benign samples are collected from the latest three years (i.e., years 2018-2020) while the malicious samples are from the first three years (i.e., years 2010-2012).

**Finding 1: The performance of ML-based Android malware detection models could be significantly improved if the temporal inconsistency is introduced in the evaluation dataset.**

Table II summarizes the experimental results for Drebin. For all the five experimental settings, we are able to achieve high performance with respect to all the four considered evaluation metrics. The highest case can achieve over 99% for all four metrics. These results experimentally confirm that we are indeed able to replicate the high performance of prior studies targeting machine learning-based malware detection.

When comparing the Baseline setting with the four variants, we can observe that the Baseline setting is not able to yield better performance. Variants 3-4 (temporally inconsistent between malware samples and benign samples) achieve significantly higher performance than the other settings, including that achieved by Baseline and Variants 1-2. The only difference between Variants 3-4 and others is the involvement of temporal biases, whether the malware and benign datasets are collected from the same time period or not. This evidence suggests that temporal inconsistency could significantly impact the classifiers' performance if introduced in the experimental datasets of machine learning approaches. Drebin can achieve the highest detection performance (i.e., 99.27% accuracy) under Variant 4 where malware samples are older while benign ones are newer. As for XMal and Fan *et al.* , we also observe the similar effects of temporal inconsistencies, which can be found on our online supplementary. In addition, Table I and the prior work [4] confirm that Variant 4 is the most common cause of temporal biases between malware and benign samples. Thus, we mainly focus on discussing the temporal biases that the benign samples are collected from the latest time while the malware samples are collected from the older time in the following experiments.

> ***Answer to RQ1:*** *Introducing a temporal difference between malware and benign samples in the experimental datasets could significantly increase the detection performance of ML-based Android malware detection approaches.*

### *RQ2: Why does temporal inconsistency make ML-based malware detection approaches perform so well?*

In the second research question, we are interested in exploring why the aforementioned machine learning approaches can achieve high performances, especially why temporal-related settings (i.e., Variants 3-4) can achieve better performance than non-temporal-related settings (i.e., Baseline and Variants 1-2). To the best of our knowledge, temporal inconsistency in the training dataset has not been well explored by our fellow researchers yet, and it is still unknown to the community why there is such an impact when temporal inconsistency is introduced in the training dataset. This question motivates us to go one step further. To this end, we resort to explainable machine learning techniques to highlight the features that significantly contribute to the classifications, hoping to understand the impact brought by these features with respect to the corresponding training datasets.

**Explainable Machine Learning Approach**. After the three ML-based malware detectors output prediction results, an explanation vector $a_i^k$ with feature importance values is calculated for each test sample $x_i^k$, where k represents the size of feature list $S$. For each feature $S_j$, the average feature importance $Avg\_fi(S_j)$ can be calculated as: $Avg\_fi(S_j) = \frac{1}{N} \cdot \sum_{i=1}^{N} a_i^j$, where N is the size of test samples. Thus, we can obtain the average feature importance for each characteristic when the model generates the predictions on a test set.

Except calculating average feature important, we sort the feature importance vector $a_i^k$ and count the frequency $Count\_top(S_j)$ whether characteristic exist in top features: $Count\_top(S_j, T) = \frac{1}{N} \cdot \sum_{i=1}^{N} [S_j \in top(a_i^k, T)]$, where function $top(a_i^k, T)$ means getting the top $T$ important features from the feature importance vector $a_i^k$. Then, this equation will judge whether feature $S_j$ exists in top T important features of $x_i^k$. A proportion of feature $S_j$ in top features would be calculated.

**Experimental Setup**. To help readers better understand this work, we resort to the same experimental settings proposed for answering RQ1 to fulfill the experiments of RQ2, i.e., three approaches with five settings constructed with balanced training datasets. The only difference is that, when rerunning the machine learning classifications, we apply the explanation module mentioned above to the original classification so as to further collect the feature importance information for each testing example. Prior studies [22], [12] have proven that the selection of feature sets plays an important role in explaining machine learning-based malware classifications, for which their performances are often decided by a small number of top-ranked features. Thus, our follow-up detailed analyses hence mainly focus on the top-ranked features.

**Finding 2: When applied to ML-based malware detection, the top-ranked features highlighted via explainable machine learning approaches may not always capture the difference between malicious and benign behaviors. They could simply be time-specific features that only exist in either historical or latest apps.**

As discussed previously, there is a strong correlation between the evaluation performance of ML classifiers and the temporal distribution of the training samples. In this RQ, we hence resort to explanations of ML classifiers for each testing sample to determine if such temporal distributions will impact the classification results. Specifically, in this work, we have identified two types of time-specific features: (1) Added ones and (2) Removed ones, which are respectively defined as follows.

- **Added Features.** Features that are added to the Android framework after the apps are released to the ecosystem. Therefore, these apps, either malicious or benign, will have no chance to access those features. However, the remaining apps, which are released after the time when the features are added, may have the opportunity to include those features. It hence introduces biases specific to time rather than maliciousness. Table III includes several added features on Drebin's predictions. For example, the "gms.ads.adactivity" app component was only added at Android 4.4 (or API level 19, the first revision released in 2013) to allow apps to display advertisements and earn revenue. Google Mobile Ads APIs became a part of Google Play services (gms) in Oct. 2013, so the apps released before 2012 will have unlikely included this characteristic, while apps released after 2018 could have. Under Variant 4 where malware samples are before 2012 while benign samples are after 2018, Drebin considers this ads-related feature as one of the top features in terms of benign identification, with 0.92 feature importance, while a zero value for identifying malware. Yet, when Drebin is trained under Variant 3 with malware data after 2018 but benign data before 2012, this feature is recognized as a malware-related feature, with 0.59 feature importance but performs a low impact on benign identification. From Table III, added features usually have higher feature importance in identifying the category containing the latest samples, whether the category is benign or malware.
- **Removed Features.** Features that have been deleted (or deprecated) from the Android framework at some stage, and hence the apps released after that will unlikely access them. In this work, we also consider deprecated ones as removed. Although deprecated features are still available, they are explicitly discouraged from being used anymore. Likely, developers who follow the official recommendations will no longer use them. Table III also includes several removed features on Drebin's predictions. For example, the "TelephonyManager.getDeviceId" API call were deprecated from Android 8.0 (or API level 26, first revision released in 2017), as this new release updated new API calls to return the unique device ID. With the same conclusion with [12], [19], the key features relevant to malware identification outputted by Drebin under Variant 4 include "getDeviceID", with 0.43 and 0.13 feature importance respectively. However, when Drebin are trained under Variant 3 with malware data after 2018 but benign before 2012, this risky feature is recognized as a benign-related feature. From Table III, removed features usually have higher feature importance to identify the category containing historical samples, whether the category is benign or malware.

Table IV summarizes the ratio of time-specific features involved in each classification of the five experimental settings by Drebin. Among top $k$ important features observed, the ratio of added and removed features (i.e., say $x$ and $y$) are calculated via $x/k$ and $y/k$, respectively. Since only a small number of features will be regarded as important ones, as we experimentally discovered previously, in this table, we only summarize the ratio based on top-10 and top-20 features ranked based on their importance.

As indicated in the Baseline setting, the ratios of added and removed features are quite high. This is expected as this setting has included a wide range of samples (i.e., from 2010 to 2020). The ratios of added features, w.r.t. predicting malware and benign apps, are more or less the same, as the ratios of removed features have a slight discrepancy. Similar results could also be observed in Variants 1&2 settings as both of them have collected app samples (i.e., both malware and benign apps) from the same period. However, when comparing the results

[5] https://developer.android.com/reference

**Table III:** Examples of time-specific features

| Feature Name | Feature Type | Updated at version | Updated at Year | Feature importance (Drebin) | | | |
|---|---|---|---|---|---|---|---|
| | | | | Variant 3 | | Variant 4 | |
| | | | | Malware | Benign | Malware | Benign |
| com.google.android.gms.ads.adactivity | App components | Added at API level 19 | 2013 | 0.59 | 0.00 | 0.00 | 0.92 |
| android.permission.read_external_storage | Requested permissions | Added at API level 16 | 2012 | 1.13 | -0.03 | -0.18 | 0.54 |
| android.permission.foreground_service | Requested permissions | Added at API level 28 | 2017 | 0.09 | 0.00 | 0.00 | 0.05 |
| landroid/telephony/telephonymanager:->getdeviceid | Suspicious API calls | Removed at API level 26 | 2017 | -0.03 | 0.05 | 0.44 | -0.16 |
| lorg/apache/http/client/methods/httppost | Suspicious API calls | Removed at API level 22 | 2015 | -0.03 | 0.10 | 0.38 | -0.36 |
| android.permission.get_tasks | Used permissions | Removed at API level 21 | 2014 | 0.00 | 0.00 | 0.08 | -0.02 |

**Table IV:** Comparison of feature importance of time-specific features for malware/benign prediction by Drebin. The chart records the ratios of test samples containing relevant time-specific features in top features when ML classifiers make predictions. The ground truth of temporal information of each feature is generated based on the official Android Developer Documentation[5].

| | | Top 10 | | Top 20 | |
|---|---|---|---|---|---|
| | | Added | Removed | Added | Removed |
| **Baseline** | Malware | 0.3342 | 0.8303 | 0.5230 | 0.8640 |
| | Benign | 0.4935 | 0.3705 | 0.5783 | 0.5765 |
| **Variant 1** | Malware | 0.1402 | 0.9010 | 0.2163 | 0.9018 |
| | Benign | 0.1097 | 0.3727 | 0.1229 | 0.4093 |
| **Variant 2** | Malware | 0.4818 | 0.3535 | 0.8194 | 0.5752 |
| | Benign | 0.6697 | 0.1575 | 0.8216 | 0.3941 |
| **Variant 3** | Malware | 0.9259 | 0.3015 | 0.9342 | 0.6941 |
| | Benign | 0.0968 | 0.4312 | 0.1142 | 0.4871 |
| **Variant 4** | Malware | 0.1834 | 0.8445 | 0.1970 | 0.8834 |
| | Benign | 0.9047 | 0.2503 | 0.9187 | 0.6471 |

obtained in Variants 3&4, for which the malware and benign samples are collected from different time periods, we could observe clear differences. Under Variant 3, we observe that when Drebin makes decisions to identify malware, 92% of malware in testing samples contain newly added features in the top 10 features, but only 9% benign in testing samples contain added features in the top 10 features. The observation is not surprising since the malware samples in this setting are collected from apps released from 2018 to 2020 but benign from 2010 to 2012. On the contrary, when looking at Variant 4, where malware samples are from 2010 to 2012, but benign samples are created after 2018, Drebin can build distinguish rules indicating that the benign identification greatly depends on newly added features while malware identification highly depends on removed (or deprecated) characteristics. This evidence indicates that the important features contributing to the high performance of Drebin may not necessarily be related to apps' maliciousness (or benignness) but could simply be discrepancies introduced by temporal inconsistencies in the training dataset. When experimental data is temporally inconsistent, newly added features have a higher positive impact on identifying the category collected on a later date, while deprecated/removed features have a higher positive impact on the category collected on an earlier date.

**Finding 3: When using testing samples from distinct periods, ML models still distinguish malware/benign based on temporal differences learned from training data, resulting in extremely poor performance.**

To further understand the impacts of temporal inconsistency, we obtain the models from RQ1 with the best performance under Variant 3 and Variant 4, respectively, and test the performance on another temporally inconsistent dataset. Table V presents the prediction performance and relevant explanation results. When Drebin is trained on Variant 4 (malware is from 2010-2012 while benign is from 2018-2020), but tested on Variant 3 (malware is from 2018-2020 while benign is from 2010-2012), it only obtains 14% accuracy, which is much less than 99% obtained in RQ1. The explanation results show that the trained model under Variant 4 still thinks that the samples with more added features are more likely to be benign, while the samples with more removed features are more likely to be malware. Similarly, when Drebin is trained on Variant 3 but tested on Variant 4, the results show that Drebin considers the samples with more added features as malware, causing only 9% accuracy. Therefore, this experiment demonstrates that if the time difference between malware and benign changes, the ML-based model trained under temporal biases doesn't work. This observation further demonstrates that ML-based malware detectors distinguish malware from benign based on time-specific features under the temporal inconsistency.

**Finding 4: If temporal inconsistency exists, all three ML-based malware detection approaches provide highly accurate predictions based on temporal differences.**

Table VI presents the prediction results and explanation results of three ML/DL-based malware detection approaches under Variant 4. We can observe that six ML-based Android malware detection approaches achieve fairly high performance, where all the accuracy and F1 score are higher than 0.98 under temporal inconsistency. This finding indicates that temporal biases between the experimental malware and benign samples influence predictions regardless of the type of machine learning algorithms. From the explanation results, we can observe that ML-based Android malware detection approaches capture time differences between malware and benign (i.e., under Variant 4, malware is likely to include removed features while benign is likely to include added features).

***Answer to RQ2:*** *Many time-specific features are only available in either historical or latest applications. Explanations for testing samples reveal that ML models correctly identify the temporal differences between malware and benign samples, resulting in high performance.*

**Table V:** Results of prediction performance and feature importance of time specific features when ML-based malware classifier is trained on one temporally inconsistent dataset and is tested on another one (i.e., Variant 3 and Variant 4).

| | Prediction Performance | | | | | Explanation results | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Top 10 | | Top 20 | |
| | Accuracy | F1 | Precision | Recall | | Added | Removed | Added | Removed |
| **Trained on Variant 3, tested on Variant 4** | 0.0968 | 0.0368 | 0.0395 | 0.0346 | Malware | 0.8754 | 0.7282 | 0.9074 | 0.8660 |
| | | | | | Benign | 0.1151 | 0.4864 | 0.1366 | 0.8033 |
| **Trained on Variant 4, tested on Variant 3** | 0.1415 | 0.0536 | 0.0598 | 0.0487 | Malware | 0.0831 | 0.5626 | 0.0965 | 0.5695 |
| | | | | | Benign | 0.7970 | 0.0778 | 0.8010 | 0.2759 |

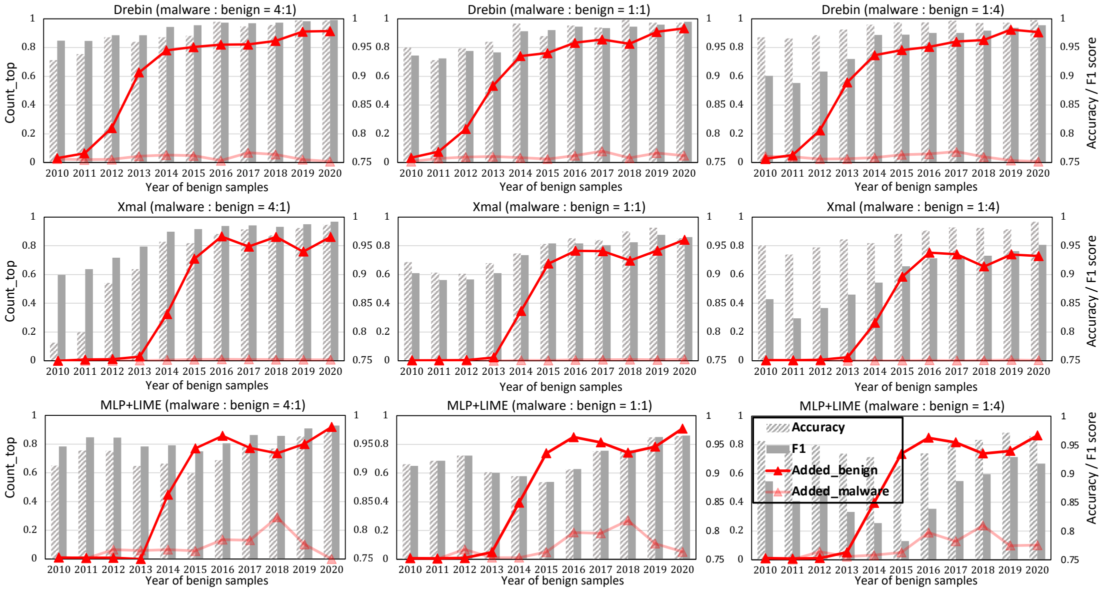

**Fig. 2:** Experimental results obtained via Setting 1 that keeps the malware data untouched (in the year 2010) and train the models on different benign samples collected at different years (from 2010 to 2020)

**Table VI:** Evaluation results of the replication study under temporal biases (Malware is from 2010-2012 while benign is from 2018-2020)

| | Prediction Performance | | Explanation Results | | |
|---|---|---|---|---|---|
| | Accuracy | F1 | | Added | Removed |
| **Drebin** | 0.9927 | 0.9927 | Malware | 0.1834 | 0.8445 |
| | | | Benign | 0.9047 | 0.2503 |
| **Xmal** | 0.9822 | 0.9824 | Malware | 0.0104 | 0.8180 |
| | | | Benign | 0.7702 | 0.7723 |
| **MLP + LIME** | 0.9861 | 0.9861 | Malware | 0.00821 | 0.7790 |
| | | | Benign | 0.81651 | 0.7704 |
| **RF + LIME** | 0.9869 | 0.9869 | Malware | 0.00687 | 0.7819 |
| | | | Benign | 0.81603 | 0.7670 |
| **SVM + LIME** | 0.9806 | 0.9807 | Malware | 0.0238 | 0.7709 |
| | | | Benign | 0.8274 | 0.7806 |

### *RQ3: How sensitive is the impact of the temporal inconsistency on the accuracy and explanation of the ML-based malware detection approaches?*

Our previous experiments have empirically demonstrated that the performance of ML-based malware classification approaches could be significantly increased when temporal inconsistencies are introduced in the training dataset. Especially on Variant 4, malware samples are from an earlier time than benign samples, achieving up to 98% accuracy for all ML-based malware classifiers. Actually, Variant 4 is commonly occurring in the research domain, as shown in Table I. The finding is however only confirmed through a limited number of experimental settings, letting it unknown if it holds true for other experimental settings (i.e., the finding per se is generic). To this end, in the last research question, we would like to explore this by conducting large-scale experiments with different settings. Specifically, we confirm the genericity through varying settings with customized malware/goodware ratios and temporal sample inconsistencies.

**Experimental Setup**. The purpose of this section is used to further confirm the correlation between inconsistent temporal distributions and classification performances when the rate of malware to benign is changed. To this end, we split our obtained dataset obtained in Section III-B1 into 22 subsets based on the APK type (Malicious/benign) and appearance time (2010 to 2020). Specifically, to investigate the influence of time interval size on final prediction performance, we consider the following time-related settings: keep the malware data untouched (in the year 2010) and train the models on different benign samples collected in different years Except that, we further consider the impacts over three malware/benign ratios: a balanced dataset with malware/benign ratio (i.e., 1:1), a large malware set with malware/benign ratio (i.e., 4:1), and a smaller malware set with malware/benign ratio (i.e., 1:4).

**Finding 5: Varying the Malware/Benign Rates in the**

**training dataset will have a great impact on the performance of machine learning-based malware detection approaches.**

Figure 2 further present the performance results and the proportion of time-related features in top 10 features ($Count\_top(S_j, 10)$) obtained via the explanable AI approach. What stands out in the figures is that malware rates can influence detection performance. When malware/benign rate is set to 4:1, three ML-based malware classifiers always present a much higher F1 score than the other two at all time points. When the malware/benign rate is set to 1:4, the malware detectors obtain the lowest F1 score. These results mirror those of the previous studies [11], [42] that have examined the impacts of unrealistic malware rates on ML-based malware classifiers. As Pendlebury et al. [11] described, most mobile applications in the real world are benign samples, but most research studies build an unrealistic malware rate, causing over-optimistic detection performance.

**Finding 6: When temporal inconsistencies between malware and benign get bigger (with a larger time interval size), ML classifiers tend to achieve "better" performance.**

From Figure 2, it can be seen that Drebin, XMal and Fan *et al.* can often achieve higher F1 and accuracy values regardless of malware rate, when temporal biases between malware and benign become larger. The first two subfigures of Figure 2 show that when the time interval between malware in 2010 and benign gets bigger, the detection performance of ML-based malware classifiers gradually improves. Indeed, there is a steady increase in the proportion of benign applications with newly added features in their top 10 key features as time increases when 2010 malware data is combined with variable benign data at different time spots. This observation indicates that as benign samples evolve, added features become increasingly important in distinguishing benign from historical malware samples. The experimental results further support our previous finding that ML classifiers learn the temporal differences between malware and benign samples, resulting in unrealistic performance to rise.

Overall, our observations suggest that the rules for distinguishing malware built by the ML models strongly depend on the temporal distribution of the training malware and benign samples. When training data is inconsistent in time, malware identification of the ML-based approaches is highly reliant on learning temporal differences, and the temporal differences are reflected in a wide range of characteristics. Explanations for testing samples reveal that the feature importance of time-specific features gradually increases as the unrealistic performance improves. Further analysis of key feature explanations reveals that temporal differences are related to a wide range of features in the feature sets.

> ***Answer to RQ3:*** *The positive correlation between temporal sample inconsistency in the training dataset (regardless of balanced or imbalanced malware/benign sample sets) and the ML-based classification results is generic. When the temporal inconsistencies between malware and benign samples are greater, ML classifiers learn a greater number of time-related differences, which subsequently contribute to higher prediction performances.*

## V. Discussion

**Explainability of Malware Detection.** In this study, we explore three explainable machine learning-based malware classifiers. Currently, more complex deep learning algorithms, such as recurrent neural networks and conventional neural networks, are becoming more popular for building malware classifiers because they could provide better detection performance without a feature selection process. However, these algorithms are usually black-box models with limited explainability. Although prior works suggest that complex deep neural networks boost performance, our experiments have demonstrated that even simple ML models can achieve a high performance up to 99% accuracy when experimental samples are temporally inconsistent. We explore the over-optimistic and unreliable experimental results caused by unrealistic evaluation designs, which have no direct connection with the types of classifiers used. Four types of ML models (i.e., SVM, attention-based neural network, MLP and RF) generate consistent evaluation results, confirming the generality of our findings for other machine learning-based malware detection approaches. In addition, we investigate two types of explainable AI approaches including model-specific explainable approaches (i.e., linear SVM and attention-based neural networks) and model-agnostic explainable approaches (i.e., LIME), confirming the validity of explainable AI approaches for analyzing or improving ML-based Android malware detection approaches.

**Time-specific Features.** We define the time-specific features based on the official descriptions of the Google developer documentation. By inspecting the feature importance of time-specific features, we found that ML-based malware detection approaches learn temporal differences to identify malware from benign when training data is temporally inconsistent. When using a smaller feature set with fewer time-specific features, the explanation results of XMal and Fan *et al.* are not always consistent with that of Drebin in RQ2, but from RQ3, we found the performance of XMal and Fan *et al.* is highly correlated with time biases in the training data. The explanation is that except for the time-specified features we defined based on Google Developer Documentation, temporal differences depend on much more complex factors. For example, we output the top 10 features for benign identification when Drebin is trained on 2010 malware and 2020 benign of RQ3. We observe that three added features are regarded as key benign-related features but these features have no impact when benign is from the historical time. What is surprising is that other top 10 features also only show a high feature importance only when benign is from the latest period. This result can be explained by the fact that the temporal differences are represented not just in the added features we defined based on the Google Developer Documentation. Our motivation is not to locate all time-specific features, but we utilize the feature importance of these features to determine whether malware detectors are reliable. More importantly, the evaluation results help us confirm that temporal biases can't be eliminated by feature selection or reduction. Although XMal and Fan *et al.* only use 154 features, our evaluation results of RQ3 show that their detection performance is highly related to temporal inconsistency.

**Threats to Validity** The primary threat to internal validity lies in the implementation of the study. To reduce this threat, we utilized three ML-based malware detection approaches. The external threat to validity mainly lies in the used datasets. We collect malware and benign samples from the Androzoo repository, which comprises a collection of more than 15 million Android samples from various application markets. We follow the same process with the reproduction study to process the application and construct the feature vectors. To further investigate the generality of our findings, we evaluate the three approaches on the different period data.

## VI. Conclusion

The paper utilized explainable malware detection models to investigate why the existing research works present highly accurate performance. By evaluating the explanation results of ML models, we found that most of the results are not realistic since ML models haven't figured out the real difference between malware and benign. Specifically, accurate predictions are strongly related to the temporal inconsistency in training data. Our work demonstrates that a robust experimental setup for malware classification models is required, otherwise, ML/DL-based models present over-optimistic results. We encourage the community to jump outside of the ideal world of high-performance of machine learning and should focus more on reliability and applicability, not only on classification evaluation metrics. Furthermore, explainable AI techniques helped us understand the inner logic and infer the decision reasons for ML-based Android malware detection models. We expect that our work will inspire future researchers to utilize explainable AI techniques to explore the underlying issues in ML/DL-based systems.

## Acknowledgment

The authors would like to thank the anonymous reviewers who have provided insightful and constructive comments on this paper. Chakkrit Tantithamthavorn was partly supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100941). Li Li was partly supported by the Australian Research Council (ARC) under a Discovery Early Career Researcher Award (DECRA) project DE200100016, and a Discovery project DP20010002.

## References

[1] Mcafee labs threats report, april 2021. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-apr-2021.pdf

[2] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, 2020.

[3] A. Razgallah, R. Khoury, S. Hallé, and K. Khanmohammadi, "A survey of malware detection in android apps: Recommendations and perspectives for future research," *Computer Science Review*, vol. 39, p. 100358, 2021.

[4] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Deep learning for android malware defenses: a systematic literature review," *ACM Computing Surveys*, jun 2022. [Online]. Available: https://doi.org/10.1145/3544968

[5] J. Samhi, L. Li, T. F. Bissyandé, and J. Klein, "Difuzer: Uncovering suspicious hidden sensitive operations in android apps," in *The 44th International Conference on Software Engineering (ICSE 2022)*, 2022.

[6] X. Sun, X. Chen, K. Liu, S. Wen, L. Li, and J. Grundy, "Characterizing sensor leaks in android apps," in *The 32nd International Symposium on Software Reliability Engineering (ISSRE 2021)*, 2021.

[7] O. Zungur, A. Bianchi, G. Stringhini, and M. Egele, "Appjitsu: Investigating the resiliency of android applications," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 457–471.

[8] A. Possemato, D. Nisi, and Y. Fratantonio, "Preventing and detecting state inference attacks on android," in *Proceedings of the 2021 Network and Distributed System Security Symposium (NDSS), Virtual, 21st-25th February*, 2021.

[9] T. Liu, H. Wang, L. Li, X. Luo, F. Dong, Y. Guo, L. Wang, T. F. Bissyandé, and J. Klein, "Maddroid: Characterising and detecting devious ad content for android apps," in *The Web Conference 2020 (WWW 2020)*, 2020.

[10] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *Proc. of the USENIX Security Symposium*, 2022.

[11] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT}: Eliminating experimental bias in malware classification across space and time," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 729–746.

[12] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[13] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[14] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 473–487.

[15] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to android malware feature learning and detection," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 244–251.

[16] M. E. Khoda, J. Kamruzzaman, I. Gondal, T. Imam, and A. Rahman, "Mobile malware detection: An analysis of deep learning model," in *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2019, pp. 1161–1166.

[17] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu, "Can we trust your explanations? sanity checks for interpreters in android malware analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 838–853, 2020.

[18] X. Su, W. Shi, X. Qu, Y. Zheng, and X. Liu, "Droiddeep: using deep belief network to characterize and detect android malware," *Soft Computing*, vol. 24, no. 8, pp. 6017–6030, 2020.

[19] B. Wu, S. Chen, C. Gao, L. Fan, Y. Liu, W. Wen, and M. R. Lyu, "Why an android app is classified as malware: Toward malware classification interpretation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–29, 2021.

[20] L. Li, K. Allix, D. Li, A. Bartel, T. F. Bissyandé, and J. Klein, "Potential Component Leaks in Android Apps: An Investigation into a new Feature Set for Malware Detection," in *The 2015 IEEE International Conference on Software Quality, Reliability & Security (QRS)*, 2015.

[21] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. 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*, 2020, pp. 757–770.

[22] M. Melis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Explaining black-box android malware detection," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 524–528.

[23] M. Melis, M. Scalas, A. Demontis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Do gradient-based explanations tell anything about adversarial robustness to android malware?" *arXiv preprint arXiv:2005.01452*, 2020.

[24] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara, "Experimental study with real-world data for android app security analysis using machine learning," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 81–90.

[25] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu *et al.*, "Reviewer integration and performance measurement for malware detection," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 122–141.

[26] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 2, pp. 1–28, 2020.

[27] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, "Why are android apps removed from google play? a large-scale empirical study," in *The 15th International Conference on Mining Software Repositories (MSR 2018)*, 2018.

[28] H. Wang, Z. Liu, J. Liang, N. Vallina-Rodriguez, Y. Guo, L. Li, J. Tapiador, J. Cao, and G. Xu, "Beyond google play: A large-scale comparative study of chinese android app markets," in *The 2018 Internet Measurement Conference (IMC 2018)*, 2018.

[29] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Understanding the evolution of android app vulnerabilities," *IEEE Transactions on Reliability (TRel)*, 2019.

[30] Y. Liu, L. Li, P. Kong, X. Sun, and T. F. Bissyandé, "A first look at security risks of android tv apps," in *The 4th International Workshop on Advances in Mobile App Analysis (A-Mobile 2021), co-located with ASE 2021*, 2021.

[31] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[32] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[33] ——, "Anchors: High-precision model-agnostic explanations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[34] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," *arXiv preprint arXiv:1805.10820*, 2018.

[35] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st international conference on neural information processing systems*, 2017, pp. 4768–4777.

[36] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 364–379.

[37] L. Li, T. F. Bissyandé, and J. Klein, "Moonlightbox: Mining android api histories for uncovering release-time inconsistencies," in *The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018)*, 2018.

[38] Y. Lin, T. Liu, W. Liu, Z. Wang, L. Li, G. Xu, and H. Wang, "Dataset bias in android malware detection," *arXiv preprint arXiv:2205.15532*, 2022.

[39] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 625–642.

[40] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 47–62.

[41] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "{CADE}: Detecting and explaining concept drift samples for security applications," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[42] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, Y. Le Traon *et al.*, "Empirical assessment of machine learning-based malware detectors for android," *Empirical Software Engineering*, vol. 21, no. 1, pp. 183–211, 2016.

[43] Y. Bai, Z. Xing, X. Li, Z. Feng, and D. Ma, "Unsuccessful story about few shot malware family classification and siamese network to the rescue," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1560–1571.

[44] Y. Zhao, L. Li, H. Wang, H. Cai, T. F. Bissyandé, J. Klein, and J. Grundy, "On the impact of sample duplication in machine-learning-based android malware detection," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–38, 2021.

[45] C. Tantithamthavorn, "Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Modelling," in *Companion Proceeding of the International Conference on Software Engineering (ICSE)*, 2016, pp. 867—-870.

[46] C. Tantithamthavorn and A. E. Hassan, "An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges," in *In Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2018, pp. 286–295.

[47] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Comments on "Researcher Bias: The Use of Machine Learning in Software Defect Prediction"," *Transactions on Software Engineering (TSE)*, vol. 42, no. 11, pp. 1092–1094, 2016.

[48] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models," in *Proceeding of the International Conference on Software Engineering (ICSE)*, 2015, pp. 812–823.

[49] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–1219, 2018.

[50] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The Impact of Automated Parameter Optimization on Defect Prediction Models," *Transactions on Software Engineering (TSE)*, pp. 683–711, 2018.

[51] ——, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *Transactions on Software Engineering (TSE)*, vol. 43, no. 1, pp. 1–18, 2017.

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

[53] M. Cao, S. Badihi, K. Ahmed, P. Xiong, and J. Rubin, "On benign features in malware detection," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 1234–1238.

[54] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1332–1349.