Understanding and Mitigating Label Bias in Malware Classification: An Empirical Study

Jia Yan[1,2], Xiangkun Jia[1], *, Lingyun Ying[3], Jia Yan[1], and Purui Su[1,4,*]
[1]TCA/SKLCS Institute of Software Chinese Academy of Sciences, Beijing, China
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China
[3]QI-ANXIN Technology Group Inc., Beijing, China
[4]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{yanjia2016, xiangkun, yanjia, purui}@iscas.ac.cn, {yinglingyun}@qianxin.com
*corresponding author

***Abstract*—Machine learning techniques are promising for malware classification, but there is a neglected problem of label bias in the annotation process which decreases the performance in practice. To understand the label bias problems and existing solutions, we conduct an empirical study based on two Portable Executable (PE) malware sample datasets (i.e., open-sourced BODMAS with 52,793 samples and a new collected MAIN dataset of 153,811 samples), and 67 anti-virus engines in VirusTotal. We first show the two ways of label bias problems, including chaotic naming rules and annotation inconsistency. Then we present the effects of two solutions (i.e., electing one reputable AV engine and aggregating multiple labels based on majority voting) and find they face the problems of feature preference and engine independence. Finally, we propose some recommendations for improvements and get a 7.79% increase in the F1 score (i.e., from 84.83% to 92.62%). The dataset will be open-source for further study.**

***Keywords*—Malware Classification; machine learning; annotation bias**

I. Introduction

Malware remains a severe threat to cyber security. To detect malware better, machine learning-based (ML-based) malware detection techniques are proposed and have achieved good results [1, 2]. Malware classification is an important task of malware detection, and ML-based solutions are also proposed and developing quickly [3]. Compared to the binary-detection task which only reports if the sample is malicious or benign, the classification task provides the malware family information which helps researchers understand attack logic, track malware authors and deploy targeted defenses [4–7].

In ML-based malware classification solutions, researchers usually extract static and dynamic features to construct feature vectors, annotate feature vectors with family names to build training data, and train a supervised model. Then, the trained model is used to classify which family a malware sample belongs to. Compared to traditional solutions with static program analysis or dynamic monitor [8, 9], ML-based solutions can process massive malicious samples in a short time and handle unknown malware samples.

Unfortunately, when applying ML-based malware classification in practice, the performance of the trained models decreases significantly. Ge et al. [10] conduct an empirical study of the impact of datasets on ML-based Android malware detection and summarize three dataset factors (i.e., class imbalance, quality and timelines). But we find it more complex to deal with malware classification and a neglected reason is the *label bias* problems in the annotation process.

In fact, label bias is a general problem for ML techniques. In common machine learning tasks (e.g., images or text), training data is annotated by employing workers [11], but researchers find it hard to guarantee the label quality without experts [12–14]. Similarly, in the malware analysis domain, anti-virus (AV) engines play the role of workers to annotate malware samples. Samples are usually submitted to VirusTotal (VT) [15], an integrated analysis platform connected to more than 70 AV engines, to obtain scan results including malware family information. However, although the AV engines could be treated as the expert annotators, the label bias problems also exist and further harm the model performance.

Specifically, we find label bias in ML-based malware classification mainly represent in two ways, i.e., chaotic naming rules and annotation inconsistency. First, although some organizations (e.g., the Computer Antivirus Research Organization) have proposed some naming conventions [16, 17], AV engines adopt chaotic naming rules. For example, a sample[1] is named as *Gen:Variant:Razy.564123* by BitDefender, where *Razy* is a unique detection technique in BitDefender and *564123* is part of the sample hash. The chaotic names make class information hard to understand by humans and may bring too fine-grained specific classes to train classification models [6]. Annotation inconsistency is that different AV engines may give different opinions on the same sample based on their own detection techniques. For example, Microsoft considers two samples as variants and annotates them both as *Worm:Win32/Mira!rfn*, while Kaspersky names these two samples as different families (i.e., *Trojan.Win32.Agent.nezvfi*[2] and *Trojan.Win32.Agent.icgh*[3]).

To deal with the label bias problems, researchers have proposed several methods, including selecting one reputable AV engines [7, 18, 19], or aggregating multiple labels based on majority voting [20–22]. However, these methods mainly deal with the textual tokens of the family names, and don't

[1]SHA-1: 3ed31058d03631bebe90b15f8dd4d1add154bbbf
[2]SHA-1: f5e585d6fabed01100d8270f9cd8a3809abaebbb
[3]SHA-1: 1c478f71c5e02d7d4b71b107d32a81ae199a2b21

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00057

take a deep study of the annotation process based on the knowledge of malware researchers. According to our studies, the method of selecting one reputable AV engine faces the feature preference problem that makes it difficult to get an effective classification model, and the aggregation method ignores the independence of AV engines which decreases the model performance.

So, we are motivated to conduct an empirical study for label bias in ML-based malware classification. It is worth noting that the label bias problems we propose in this paper do not exactly equal the noise problems, because all the AV engines are experts and there are no "wrong" labels. Thus, our goal is not to "correct" the labels, but find a better way to leverage the labels to improve the performance of the models. Compared to Ge's work [10], we focus on label bias which is different from the dataset bias. Pendlebury et al. [23] point out the experimental biases in spatial bias and temporal bias when evaluating ML-based malware detection. In addition to a fairer evaluation, we expect to leverage the existing labels to improve the models' performance in this paper. Zhu et al. [24] measure the label dynamics of online AV engines and study the scan result changes along with the timeline, and we care more about the results of different AV engines.

Specifically, in the empirical study of this paper, we set the following research questions (RQs).

- RQ1: What are the label bias problems in ML-based malware classification?
- RQ2: Do existing methods solve the label bias problems?
- RQ3: Can we improve the label bias problems better?

To answer the RQs, we perform a series of experiments on real-world malware samples to study the label bias problems systematically. Specifically, we mainly utilize the effectiveness of the trained model (i.e., F1 score) as a comprehensive evaluation metric. In addition, since the effectiveness of the models depends on the machine learning algorithm and the training data constructed from labels and features, we choose a representative algorithm used in recent related works (i.e., DNN [2]) as the fixed training algorithm and focus on the study of labels. Overall, we utilize the public dataset BODMAS [6], and construct a newer and larger dataset with a security company (i.e., QI-ANXIN Technology Group Inc. [25]). All labels for the dataset are obtained from the VT-provided API and cleaned.

For RQ1, We first show the two presentation ways of label bias problems (i.e., chaotic naming rules and annotation inconsistency). We find that AV engines may name malware samples mainly based on four categories (i.e., traditional organization, attack techniques, detection methods and develop kits), and different AV engines may give the names based on different metrics that brings the inconsistency for the same samples. For RQ2, we present the effects of two solutions (i.e., electing one reputable AV engines and aggregating multiple labels based on majority voting). We find that the models' performance vary a lot with different label source of reputable AV engines because of the feature preference problem, and the label aggregation method is not as good as expectation because of ignoring the independence of engines. For RQ3, we propose a feature preference table of different AV engines and recommend user to select engines based on the extracted features. And we recommend users to eliminate consumer engines while aggregating labels. The results show that we can get a 7.79% increase in terms of F1 score (i.e., from 84.83% to 92.62%).

In summary, this paper makes three key contributions:

- We conduct an empirical study to understand the label bias problems of machine learning based malware classification (i.e., chaotic naming rules and annotation inconsistency) and two existing methods to deal with label bias problems (i.e., electing reputable engines and aggregating labels based on majority voting) based on two datasets (i.e., BODMAS and MAIN dataset).
- We find that existing methods ignore the feature preference problem and the engine independence problem. We propose our mitigation methods for the label bias problems and get a better performance results.
- We will open-source a new dataset of 153,811 PE malware samples with static features and dynamic API calls sequence to help future related research.

II. Background & Related Work

In this section, we start by introducing machine learning based malware classification. Then, we summarize the label bias problems and clarify our focus in this paper.

A. ML-based malware classification

Usually, the malware classification tasks are modeled as a supervised learning process [26–28]. Compared to the malware binary-detection (i.e., benign or malicious) [1, 2, 29, 30], the malware samples are not only annotated with malicious tags but also with family names in malware classification tasks.

Figure 1 shows the training and evaluating workflow of ML-based malware classification. In the training phase, researchers first extract and construct feature vectors from malware samples through malware analysis methods, format the training data by annotating the feature vectors, and use the training data (i.e., annotated feature vectors) to obtain a trained malware classifier. In the evaluating stage, the unknown samples are also processed to feature vectors and fed to the trained model. Then, the model classifies whether the samples are malicious and which families they belong to. As introduced above with Figure 1, the effectiveness of ML-based malware classification depends on the machine learning algorithm and the training data with labels.

1) Algorithm: Most researchers have been working on improving the algorithms, from the traditional machine learning to deep learning. Specifically, traditional machine learning algorithms (e.g., SVM [31]) can be used for both malware detection [32, 33] and classification [34, 35], but they encounter the bottleneck of domain expertise and the raised demand for handling massive malware. Deep learning is gaining popularity due to its supremacy in terms of accuracy when trained with huge amount of data, less requirement of domain expertise,

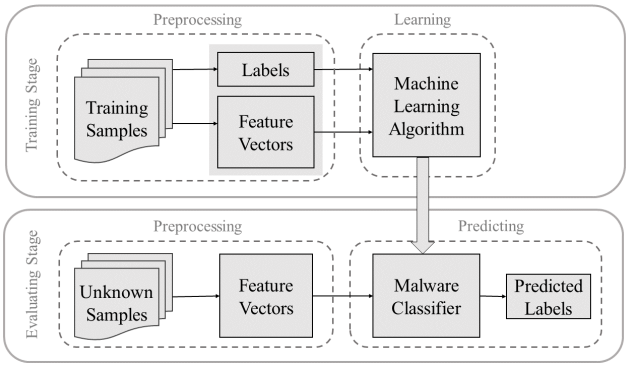

Figure 1: Workflow of ML-based malware classification.

and the detection ability to handle the newly unknown malware. The representative deep learning algorithms, e.g., Deep Neural Network (DNN) and Convolutional Neural Network (CNN), are utilized in recent ML-based malware classification works [2, 4, 26, 36–38]. For example, MalConv [26] leverages the CNN-based algorithm to train a malware classifier by eating the raw bytes of program binaries, which avoids brittle features and over-focusing on PE structure information.

2) Feature Vector: In order to adapt to machine learning algorithms and decrease the overhead of training, original data (i.e., program binaries) are usually transformed to feature vectors [3]. The features contain static features extracted with static analysis (e.g. IDA Pro[39]) and dynamic features catched from the program execution in a sandbox (e.g. Cuckoo sandbox [40]). The difference between the two types of features is whether the analyzed malware is executed or not. Specifically, we detail the two types of features as following.

Static features. (1) Bytes and characters are the obvious features of programs [2, 26, 41]. Specifically, the entropy of bytes can model the content of files generally, and the printable characters show more unique information of specific malware families, such as the strings of file path, network domain and IP address. (2) Structure information and the semantics information of PE file headers and metadata can be used as features [2], such as section numbers, timestamps, and import functions. (3) Disassembled op-codes or instructions are also used in some works [42–44]. Static features are rich and powerful, but own inherent flaws of precision. Also, when the code is packed or obfuscated, the static analysis fails to obtain any available information.

Dynamic features. (1) API call sequence [45, 46] is a common dynamic feature to capture malicious behaviors no matter the malware is obfuscated. (2) Memory usages or changes of computer status are used in some works [47, 48] to determine whether the program is malicious or benign. Although dynamic features can catch more details including the true intentions of malware, it is a costly method for both extracting and using. At the same time, there is a strong premise that the malware can run in the sandbox and the behaviors are adequately triggered [8, 49].

3) Label: Manually craft analysis with expertise knowledge is the best way to collect convincing labels [31, 50], but the cost of analyzing massive samples is unaffordable. To trade off the overhead, researchers usually utilize AV engines and platforms to get the labels. The most popular platform is VirusTotal (VT) [15] which integrates over 70 AV engines and covers most of the mainstream detection engines. After submitting a sample to VT, all engines in VT analyze the sample and output their reports with rich information including malicious tags and family names. VT also provides APIs to download and process the reports in batches, that makes it practical for annotating malware datasets automatically.

B. label bias problems

The ground-truth label is essential to supervised learning but hard to get. In common machine learning tasks (e.g., image or text processing), researchers propose several methods to create trustworthy labels, e.g., to aggregate labels from crowdsourcing platforms [11, 51], or build systematic projects (e.g., Amazon SageMaker [52]) to assure the quality of annotation [53, 54]. However, the annotation process still faces label bias problems. For example, human observers are more likely to perceive women's faces as happier than men's faces even when their smiles have the same intensity [55].

Similarly, ML-based malware classification also faces label bias problems, although the AV engines could be seen as experts. It is not exactly the same as the label noise problem as there are no "wrong" malware family names. The label bias problems in this paper are mainly caused by the subjectivity of annotation, such as AV engines' own naming rules or analysis techniques. It presents as the disagreement of family names for the samples from different AV engines, and harms the models' performance as the noisy labels.

Currently, researchers propose two main methods to handle label bias problems including selecting reputable annotators [7, 18, 19, 56] and label aggregation based on majority voting [2, 6, 20, 21, 57]. For example, Kurt et al. [19] take Kaspersky and AVG as reputable AV engines to detect and annotate malware samples. AVCLASS [20] simplifies the VT results of each engine, truncates useless information, filters generic tokens, counts the simplified labels, and finally gives the family name of a sample with the maximum count simplified label. Unfortunately, according to our empirical study, the two methods still have some limitations.

Focus of this paper. In this paper, we are trying to empirically study the label bias problems and existing methods for handling them. There are some related works for data bias problems. Compared to the metrics of statistical analysis proposed by Mohaisen et al. [58], we perform the empirical study by training models based on the bias labels and comparing the models' performance. Pendlebury et al. [23] point out the experimental biases in spatial bias and temporal bias when evaluating ML-based malware detection. Besides performing a fairer evaluation, we expect to leverage the existing labels to improve the models' performance in this paper. Zhu et al. [24] measure the label dynamics of online AV engines and study

TABLE I: Malware datasets. Binaries: whether have original executable files. Family: whether have family information derive from VirusTotal, meanwhile BODMAS has its own family attribute.

Dataset	# of samples	Collected period	Family	Binaries
BODMAS	30,852	8/29/2019 - 9/30/2020	✓	✓
MAIN	153,811	1/1/2021 - 1/31/2021	✓	✓

* BODMAS has its own family results.
MAIN's family information comes from VT.

TABLE II: Label source. The reputable engines work on both MAIN and BODMAS dataset, but the label aggregation based majority voting only work on BODMAS dataset. BODMAS has its own implementation.

Method	Label Source
Reputable engines	AVG, Avria, Kaspersky, Symantec BitDefender, F-Secure, Microsoft
Majority Voting	*AVCLASS*, *BODMAS*

the scan result changes along with the timeline, and we care more about the results of different AV engines. A recent work proposed by Ge et al. studies the selection bias problem in datasets [10], but we focus on the label bias problems in the annotation process. In fact, according to our experiments, the performance of the trained models with the same samples but with different labels varies significantly. In addition, we mainly study the windows Portable Executable (PE) format malware which plays a critical role in APT attacks, while others mainly study the Android malware as the dataset is more convenient.

III. Methodology

A. Dataset

1) Data Collection: We collect two datasets of various malware samples for the empirical study as shown in Table I.

BODMAS dataset. BODMAS [6] is a dataset containing 52,793 PE samples which is published recently. The samples are all malicious with the family information (total of 581 family labels). The labels come from VT but processed with the custom scripts of BODMAS which is similar to AVCLASS [20] but pluses expertise knowledge. In order to meet the request of our experiments, we reduce the BODMAS dataset to 30,852 samples with 79 families after cleaning. Cleaning details are introduced in Subsection III-A3.

MAIN dataset. We collected a large amount of PE malware samples in the wild by working with a cyber security Inc. (i.e., QI-ANXIN Technology Group Inc.). Specifically, we collect the samples submitted and analyzed in VirusTotal in 31 days during January 2021. After filtering the unsuitable samples, we establish a PE malware dataset containing 153,811 malicious samples. For each sample, we acquire the analysis reports of 67 AV engines[4] from VT.

Ethical considerations. BODMAS is an open-source malware dataset, and we collected the MAIN dataset from a cybersecurity company based on the ethical policy. All experiments were done in-house. In addition, all samples have been submitted to VT by others already, and details about each malware are public (e.g. HashID, Creation Time, Last Analysis Time, etc.). If this paper is accepted, we will share the MAIN dataset after desensitization, including feature vectors, dynamic behavior trace and original binary programs.

2) Data Annotation: We annotate samples based on the scan results from 67 AV engines in VT. Additionally, we take two existing solutions for label bias problems for the empirical study, i.e., selecting reputable AV engines and label aggregation based on majority voting.

Selection of reputable AV engine. According to the related report [61] and papers [6, 20, 24], we select 7 of the 67 AV engines as the reputable label sources. We take labels from 7 reputable AV engines as single ground-truth label sets respectively. Some AV engines have much more citations in academic research [24], such as Kaspersky, Symantec, etc. The other AV engines are often intergrated by certain consumer AV engines as a third-party engine. For example, F-Secure have the same family information with Avria, where Avria plays a role of supplier [61].

Aggregation based on majority voting. We process and obtain the labels from two implementations of label aggregation based on majority voting, i.e., AVCLASS [20] and BODMAS [6]. AVCLASS is the representative work of aggregation based majority voting which is used in several works [62, 63]. It takes VT reports as original label source, truncates redundant information from each engine's full labels, replaces aliases, and votes to select the majority of labels as ground-truth labels. Although BODMAS takes the similar way to obtain *BODMAS*[5] labels as *AVCLASS*, it owns in-house scripts assisted by expertise analysis. The similarity of algorithms makes them agree on some results, but *BODMAS* provides more accurate labels as it claimed in the paper [6].

3) Data Cleaning: Both of the MAIN and BODMAS datasets are unbalanced that some families have very few samples, perhaps as few as one or two. It is difficult to train a multi-classification model on such a highly imbalance dataset. Thus, we clean all the minority families under the following rules to reduce the stress of the training model.

First, each sample must have been annotated by all the label sources respectively. For the MAIN dataset, each sample should be labeled by reputable AV engines. For the BODMAS dataset, each sample should not only be analyzed by the 7 AV engines, but also processed by the methods of AVCLASS and BODMAS. Second, each family of any independent label set must contain more than 5 samples. Then, the two dataset are respectively divided into training set, test set and validating set in a ratio of 3:1:1, that requests the families have enough samples for the 4-fold cross-validation experiments [64]. Fi-

[4]We excludes the AV engines which are only based on ML methods according to VT's introduction , such as Cylance [59] and MAX [60], as they only show the probabilities of maliciousness instead of family information.

[5]To distinguish between datasets, methods, and label sets, we use AVCLASS and BODMAS for dataset and method, italics *AVCLASS* and *BODMAS* for label sets.

nally, we settle down the scales of the two dataset that there are 153,811 samples in the MAIN dataset, and 30,852 samples in the BODMAS dataset.

There is an exception situation to be noticed that AVCLASS generates 1,406 singleton labels on the BODMAS dataset which should have been filtered by the above cleaning process, as the other 8 label sources share a common list of samples. But, to keep enough samples for learning classifiers, we do not filter these discrete samples which only belong to *AVCLASS*. In fact, this exception helps us get more observations and we will discuss them in Subsection IV-C and V-B.

B. Evaluation Method Selection

We choose the DNN algorithm to train the malware multi-classification models, which is verified by several works [1, 2, 6]. The model is made up of 5 layers, and the first 4 layers are divided into 2 groups. Each group contains a 1024-node layer, followed by a dropout layer as 0.5. Then, the model takes a sigmoid function as the last hidden layer to output the classification results. Specifically, we set the learning rate as 0.001, and training epoch as 100. In order to alleviate the deviation caused by the train and test data contingency, we eliminate the latent influence by obtaining the average value of performance metrics through the 4-fold cross validation.

C. Feature Extraction

We extract both static and dynamic features from malware samples to construct feature vectors as training examples for the machine learning algorithm.

Static Feature. We follow the method proposed by Saxe et al. [2] to extract four groups of static features, including byte entropy (B for short), printable strings (S), import function (I), and PE metadata (M). The static features construct a vector of 1024 dimensions, and the details are shown in APPENDIX A-A. For convenience, we call the complete features of the four groups as the INVINCEA features (including B, S, I and M) according to the authors' organization.

Dynamic Feature. In order to obtain the samples' dynamic behavior features, we leverage sandbox-based malware analysis to execute the PE samples and record the traces to extract features, similar to [4]. Specifically, referring to work [65], we collect the dynamic behavior of the sample during the first two minutes of execution, i.e., the appropriate execution time. And we construct a 1024-dimensional feature vector with the top 1024 3-gram sub-sequence in the whole API sequence based on the information gain of each 3-gram sub-sequence, similar to [66]. The details are shown in APPENDIX A-B.

D. Evaluation Metrics

As the dataset is unbalanced, it's not fair to use accuracy for performance evaluation that the model may focus on detecting the majority classes and ignore the minor classes, but the accuracy metric still keeps a high score. Due to the accuracy paradox, we choose F_1 score as the primary metric to evaluate the effectiveness of trained models, which is the harmonic mean of the $Precision$ and $Recall$, as shown with Equation 1.

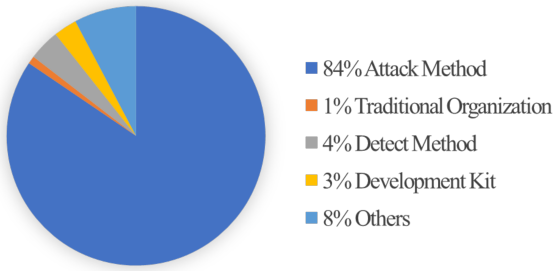

Figure 2: Different name rules on BODMAS dataset.

TABLE III: Four categories of label naming conventions.

Category	Examples
Tradition	cosmicduke
Att Tech	Sormattack, Vtflooder, Zbot
Detec Meth	Razy, Zusy
Dev Kit	Delphi, Delf
Others	unknown, malware

The higher F_1 score means the better performance of the model.

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (1)$$

IV. Empirical Study

In this section, we present the details of the empirical study to answer the three research questions introduced in Section Introduction. For each RQ, we introduce the experiment method first, then present the results and analysis.

A. RQ1: What are the label bias problems in ML-based malware classification?

1) Chaotic naming rules: The directed manifestation of label bias is the casual malware family names. Although there are some naming conventions [16, 17], AV engines give chaotic names. We study this problem based on the BODMAS dataset as it provides the ready-made family names. We split the raw labels to the sequence of consecutive non-alphanumeric characters by the tokenization module of AVCLASS. Then we check the tokens manually for their name categories according to the paper [67].

Results and analysis. As shown in Figure 2, there are mainly four categories of label naming convention, including traditional organization, attack techniques, detection methods and develop kits. We also list the examples of four categories in Table III. According to our analysis, 84% of the labels are named by attack technique (e.g., vtflooder and FakeAV), 4% by detection method (e.g., Agent), 3% by development kit (e.g., Delf and autoit), and 1% stem from the traditional organization (e.g., CosmicDuke, belonging to MiniDuke APT). There are some generic or random labels that we classify as others, such as "unknown" and "win32".

For example, a malware sample is named as *Gen:Variant.Razy.476334* and the name includes the

TABLE IV: Data to generalization validation on different MAIN and BODMAS. The shared family have common family labels on both two dataset. # of sub dataset: shows the number of samples belongs to the corresponding shared family.

Label Source	# of family label		# of shared family	# of sub dataset	
	BODMAS	MAIN		subset-B	subset-M
AVG	74	60	13	3233	93123
Avria	63	55	12	3259	100339
BitDefender	191	114	9	268	9381
F-Secure	64	55	12	3259	100341
Kaspersky	77	50	9	2215	80295
Microsoft	105	80	14	1503	68348
Symantec	40	30	13	3307	140033

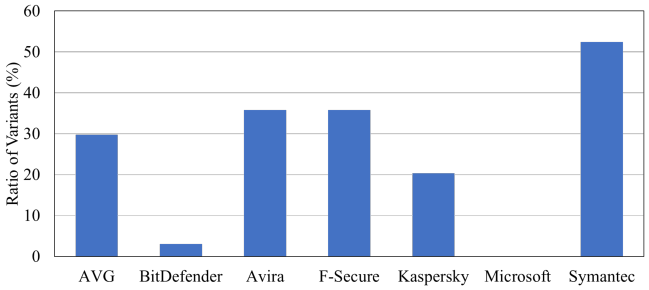

Figure 3: Ratio of malware variants in subset of MAIN dataset. Microsoft is the baseline. The blue bar is the ratio of variation in specified dataset.

tokens of *Rzay* and *476334*. Razy represents a unique detection technique of BitDefender, while *476334* is a random number for marking malware variants. When we adopt Razy as the label, the label granularity is too coarse as all the variants are considered as a cluster. In the opposite, if the name comes from the traditional family information directly, such as *PonyStealer* of a malware sample *Gen:Heur.PonyStealer.qq0@CGd9nvki*, taking the characters *qq0@CGd9nvki* is too specific.

2) annotation inconsistency: The annotation inconsistency is mainly related to the expertise level, i.e., presents as the detection ability or the preference for different features in malware analysis tasks. If an AV engine cannot detect one of the features or prefer one of them, it will give a different family name from other AV engines. As there is no baseline for this problem, we select one of the reputable AV engines to study the consistency of other AV engines. We find the shared examples based on the results of one engine and check if other engines keep the same labels for the variant in a family. The experiment is based on two datasets, i.e., BODMAS and MAIN dataset, and the details of shared samples show in Table IV.

Specifically, there is a sample A_{bodmas} in BODMAS, the baseline label is Microsoft (M) and the comparison engine is Symantec (S). Thus, the label for A_{bodmas} is M-A_{bodmas} based on Microsoft and the label S-A_{bodmas} stems from Symantec. For a sample B_{main} is the variant of A_{bodmas}, if we assume M-A_{bodmas} is the same as M-B_{main} based on the baseline tool Microsoft, S-A_{bodmas} should be the same as S-B_{main}. Otherwise, this indicates that one of the AV engines has changed its detection strategy and caused the annotation to be inconsistent.

TABLE V: F1 scores of classifiers trained on BODMAS dataset. Feature is INVINCEA, and labels are from 7 reputable AV engines.

Label Source	F1	Label Source	F1
AVG	0.7112	F-Secure	0.9073
Avira	0.9152	Kaspersky	0.8820
BitDefender	0.7162	Microsoft	0.7587
Symantec	0.9541	–	–
AVCLASS	0.8483	BODMAS	0.7190

Results and analysis. We select the Microsoft engine as the baseline label source and show the inconsistency ratios of other reputable AV engines in Fig. 3. BitDefender shows no more than 3%, but the other engines have variation ratios of more than 20%. As Symantec has an inconsistency ratio exceeding 52%, we take a further study and find it utilize a hybrid detection method of the fusion of traditional techniques and the ML-based method. As a result, it names some samples as the name combinations as "*ML.Attribute.HighConfidence*", and the left token may change after tokenization, leading to a surge in the inconsistency ratio.

B. RQ2: Do existing methods solve the label bias problems?

1) Effect of existing methods: To select a reputable AV engine, we follow the experimental design in Section III to train the models based on different label sources, respectively and get seven classification models, as shown in Table V. And we follow the details in the papers [6, 20] to process the label sources and train another two classification models marked by the methods, i.e., AVCLASS and BODMAS in Table V. We conduct all the experiments on the BODMAS dataset.

Results and analysis. According to the F1 scores of different classification models, we find that the models' performances vary a lot when adopting different results of AV engines, from 0.9541 (i.e., Symantec) to 0.7112 (i.e., AVG). It means that not all reputable engines are suitable for training malware classification models, and it is hard for users to select reputable annotators. We also find that some results of single AV engines are better than the performance of the improvement methods (i.e., 0.8483 of AVCLASS and 0.7190 of BODMAS). It shows that the label aggregation solution cannot beat all reputation engines for better classification performance. Therefore, we take further studies on the two improvement methods.

2) Further study for selecting reputable engines: We observe that the engines may prefer some features and name the samples based on more consideration for the preferred features. To verify our observation of feature preferences, we conduct an experiment like the ablation study. Specifically, we first test the situation of dropping one type of feature, and then test the effect of only taking the dropped feature. Thus, for static features, we have nine groups of feature combinations, i.e., SIM, BIM, BSM, BSI, B, S, I, M and the complete INVINCEA as shown in Table VI. We also compare the

TABLE VI: Annotation Bias: F_1 scores of multi-classifiers trained Main dataset.

Label Source	INVINCEA	SIM	BIM	BSM	BSI	Bytes	Strings	ImpFunc	Metadata	API
AVG	0.9367	0.8785	**0.9430**↑	0.9373↑	0.9276	**0.9135**	0.6951	0.6037	0.7173	0.7745
Avira	0.8468	0.7734	0.8341	**0.8392**	0.8354	**0.8228**	0.5911	0.7017	0.6920	0.6748
BitDefender	0.7153	0.6737	**0.7075**	0.6931	0.6978	**0.6853**	0.4853	0.5389	0.5422	0.5476
F-Secure	0.8267	0.8184	**0.8439**↑	0.8280↑	0.8389↑	**0.8267**	0.5873	0.6935	0.7090	0.6758
Kaspersky	0.8318	0.8273	**0.8319**↑	0.8224	0.8076	0.7667	0.6069	0.7388	**0.7694**	0.7093
Microsoft	0.9530	0.9496	**0.9537**↑	0.9418	0.8747	0.8761	0.6246	0.7464	**0.9114**	0.8506
Symantec	0.9109	**0.9136**↑	0.9108	0.9049	0.8938	0.8529	0.3658	0.6652	**0.8926**	0.8349

* B: bytes feature, S: printable strings feature, I: PE import function feature, M: PE metadata feature, API: dynamic API call sequence. INVINCEA is equal to BSIM. ↑: represents an improvement in performance compared to the baseline (i.e., INVINCEA).

static feature with the dynamic feature (i.e., the API sequence feature).

We construct feature vectors based on different groups of features and annotate the feature vectors with the labels from each AV engine to construct training examples, respectively. Finally, we train the models based on different training examples in the MAIN dataset, and the effectiveness of the trained models presents the preference of different engines.

Results and analysis. The performance results of trained models based on different features are presented in Table VI. From each row of Table VI, we can see the F_1 score changes of dropping a feature and only taking the dropped feature to train a model, according to the baseline of the INVINCEA features.

Taking the first row as an example, the F_1 score of the trained model is 0.9367 based on the training examples constructed by the INVEINCEA features and the AVG labels. When dropping the string feature (i.e., the training examples are feature vectors containing BIM features, also annotated by AVG), the F_1 score increases from 0.9367 to 0.9430, which means the string feature has a negative contribution to the model's effectiveness. The model trained only with the string feature performs more poorly (0.6951) than the models with bytes (0.9135) or metadata (0.7173), which confirms the string feature is not an effective feature when taking AVG labels. The same situation happens on the import function feature, while the byte and metadata features present the opposite situation. Especially, just taking the byte feature could get a comparable result to the INVINCEA features. Additionally, the API feature performs worse than the complete static features but better than some separately.

Combining the results of all the test AV engines in each row, we can see that the F_1 scores decrease when dropping one feature or taking only one feature. But dropping the string feature will not bring much influence, and even increases the model's effectiveness sometimes. In conclusion, AV engines are biased toward different features, and various AV engines have different detecting abilities for specific features.

3) Further study for label aggregation based on majority voting: We observe that, during the training process of selecting reputable AV engines as ground truth labels, some AV engines get close F1 scores, and their training logs are pretty similar (e.g., Avira and F-Secure). It may indicate the existence of homologous AV engines. To verify our observation, we use a vector $\vec{v}$ to represent the distribution of each family label in a label source and adopt the cosine similarity algorithm to measure the similarity of different engines based on their label results.

Specifically, the dimension of $\vec{v}$ is the sum of all engines' label types, and the value of each dimension is the number of samples belonging to the specified family label. The $\vec{v}$ can be represented as Eq. (2), where n_i and n_j represent the numbers of samples belonging to ith and jth family label in a specified label source.

$$\vec{v} = \left[\cdots, \quad n_i, \quad \cdots, \quad n_j, \quad \cdots\right] \tag{2}$$

Thus, for a dataset, if we measure the label similarity of two engines $\boldsymbol{A}$ and $\boldsymbol{B}$, $\vec{v}_a$ and $\vec{v}_b$ can be mapped by label distribution from $\boldsymbol{A}$ and $\boldsymbol{B}$, and the similarity of $\boldsymbol{A}$ and $\boldsymbol{B}$ will be represented by Eq. (3).

$$\textit{Cos\_Sim}(\text{A, B}) = \cos(\vec{v}_a, \vec{v}_b) = \frac{\vec{v}_a \cdot \vec{v}_b}{\|\vec{v}_a\| \, \|\vec{v}_b\|} \tag{3}$$

The Cos_Sim is much closer to 1, and the engines are much closer to homologous. To find all homologous relationships, we research the similarity of each pair among the 67 AV engines in VT based on the MAIN dataset.

Result and Analysis. We present the results of the independence experiment in Fig. 4, and find that homologous label sets exist in high-reputation AV engines, presenting as the consumer-supplier relationships. While a report [61] provides some pairings between consumer and third-party (supplier) engines, we found more consumer-supplier relationships. Although the labels of consumer-supplier engines appear to be normalized and consistent, the "false" normalization may cause researchers to blindly trust consumer engines as the ground truth, which engines are not suitable for the label aggregation method.

Specifically, there are 67 AV engines in our experiments on the MAIN dataset. We calculate the similarity of each pair of engines, according to Eq. (3), and get a matrix of size 67 by 67. We transform the matrix to the axisymmetric heat map shown in Fig. 4. Ideally, if each pair of engines are independent, the heat map should show a dark blue diagonal line containing only individual points, no region of points. However, we discover seven regions of correlated engines and many independent engines.

First, the engines in the red dotted box at the upper right

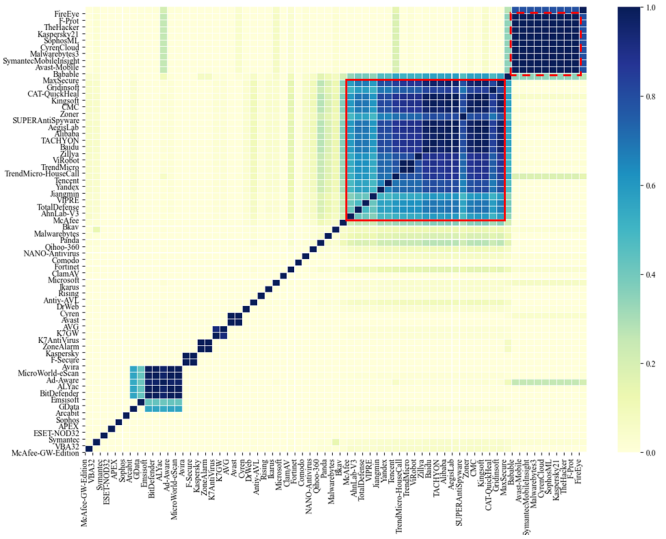

Figure 4: Heatmap of the similarities of AV engines based on MAIN dataset. The AV Engine names on x axis has been shown in APPENDIX B.

TABLE VII: List of supplier engines and consumer engines. Consumer engines incorporate supplier engines to detect malware.

Supplier Engine	Cosumer Engine
Avast	AVG
Avira	F-Secure
BitDefender	Ad-ware, ALYac, Arcabit,* G Data,* MicroWorld-eScan, Emsisoft, VIPRE,† Quick Heal,† Tencent†
Kaspersky	ZoneAlarm
K7GW	K7AntiVirus

* integrate more than one detecting engine.
† consumer engines defined by AV Comparatives[61].

corner of Fig. 4 are inoperative engines, such as Symantec Mobile Insight which specializes in Android and cannot detect PE-format programs. Second, engines in the solid red box annotate lots of samples as benign, resulting in the presentation as a cluster. For example, TrendMicro gives over 40% benign labels to the samples, and Alibaba annotates more than 83.5% samples as benign. Finally, the other five groups at the lower left corner are homologous engines after manual verification, also listed in Table VII. It is noted that Arcabit and G Data integrate not only one type of engine. That's why they look lighter blue than the other engines in the BitDefender region in Fig. 4.

To explore the details of the consumer-supplier relationship, we design some metrics to evaluate the quantification relationship and take the BitDefender group in Table VII as an example. As shown in Table VIII, we measure the eight AV engines with the baseline of BitDefender, and use Kaspersky for comparison. CL and BL represent labels from the consumer engine and BitDefender, respectively. Thus, $\frac{CL \cap BL}{CL}$ and $\frac{CL \cap BL}{BL}$ represent the proportion of shared labels in the consumer engine and BitDefender label sets. CS means

TABLE VIII: Metrics of consumer engines integrating BitDefender as third-party engines.

AV engines	$\frac{CL \cap BL}{CL}$	$\frac{CL \cap BL}{BL}$	$\frac{CS \cap BS}{BS}$	Cos_Sim
BitDefender	100.00%	100.00%	100.00%	1.000000
Ad-Aware	88.28%	99.12%	99.44%	0.999646
ALYac	67.26%	99.12%	91.93%	0.966256
Arcabit	14.40%	15.79%	32.61%	0.556163
eScan	91.94%	100.00%	99.96%	0.999999
Emsisoft	82.71%	96.49%	96.23%	0.995013
G Data	19.17%	85.09%	57.57%	0.425818
Kaspersky	0.00%	0.00%	0.00%	0.000000

CL: label set from the specified current consumer engine. BL: label set from BitDefender. CS: samples belong to CL. BS: samples belongs to BL. Cos_Sim: cosine similarity between consumer engine and BitDefender.

TABLE IX: The preference of different AV engines

	AVG	Avira	BitD	F-S	Kas	Micro	Sym
B	**1**	1	1	1	2	2	2
S	**4**	5	5	5	5	5	5
I	5	2	4	3	3	**4**	4
M	3	3	3	2	1	**1**	1
A	2	4	2	4	4	**3**	3

the samples belongs to CL, and BS represents the samples belongs to BL. Thus, $\frac{CS \cap BS}{BS}$ represents the proportion of samples, belonging to each consumer engine's shared labels in the MAIN dataset.

As the Table VIII shows, all the 6 consumer engines integrate BitDefender results with the Cos_Sim value from 0.4258 to 0.9999, and cover samples in the MAIN dataset from 32.61% to 99.96% respectively. Meanwhile, all results of Kaspersky are 0 because BitDefender and Kaspersky are orthogonal on the label set.

C. RQ3: Can we improve the label bias problems better?

1) Recommendation for selecting reputable engines: According to our observation that AV engines have feature preference problems, we have the following recommendation for selecting reputable engines. To make the preferences of AV engines clear, we rank the results by taking each single feature and show the order in Table IX. For example, the results of AVG are presented as bytes, API, metadata, strings and import function in order from good to bad. When performing ML-based malware classification, users should select reputable engines based on the features they extract, thus the trained models could get better performance.

From the data of Table IX, we can also see the annotation inconsistency problem and have some suggestions. If the ranking orders are similar, we can treat the annotation criteria as consistent. For example, supposing a malware family has unique characters in metadata and import function, an AV engine that prefers the metadata feature might give a different family name from the one that picks the import function feature. Thus, in addition, users could consider the concrete

TABLE X: The influence of consumer engines to AVCLASS method. The minority family represents which family have no more than 5 samples. F1 score means the scores from multi-classifiers trained on INVINCEA features.

BODMAS Dataset		*AVCLASS*	*AVCLASS**
original	# of labels	1491	2554
	# of samples	30852	30852
	# of minority family	3	12
	# of discrete samples	1406	2466
reduced	# of labels	82	76
	# of samples	29429	28345
	F1 score	0.8483	0.9262

F1 scores of classifiers shown in Table VI, which presents the engine's detection ability for this feature.

For example, although Microsoft takes the API feature in third place, it gets the highest F1 score when taking only the API feature, as marked with a bold number in Table IX. If a malware sample has advanced countermeasure techniques against the static analysis, the other AV engines may extract less information and annotate it with general names. Still, Microsoft will give more details and an informative family name based on its dynamic analysis ability.

2) Recommendation for label aggregation: For label aggregation in malware classification tasks, we need to pay attention to the pitfalls of the consumer engine, which may create the illusion of majority for the label aggregation method, according to our observation. We also perform a study on the improvement of eliminating the consumer engine. Specifically, we streamline the engine list of each sample's VT report before AVCLASS generates labels based on the results of Table VII. After the elimination, a new label set called *AVCLASS** is generated based on the remaining 54 AV engines. We train a new multi-classifier by combining INVINCEA features and *AVCLASS**.

Our intuitive idea is that a set of *AVCLASS* labels generated without preprocessing cannot be trusted. Due to the consumer AV engines, we speculate that the *AVCLASS* label set may be the duplicated counting results. To measure the influence of consumer engines, we streamline the engine list of each sample's VT report before AVCLASS generates a label by eliminating all latent consumer engines, as shown in Table VII. After the elimination, a new label set called *AVCLASS** is generated based on the remaining 54 AV engines. We train a new multi-classifier by combining INVINCEA features and *AVCLASS**. The experiment details are shown in Table X. Compared with classifiers trained by *AVCLASS* (84.83%), the classifiers based on *AVCLASS** have a higher F1 score of 92.62% with a 7.79% increase.

Further study shows that the cost is to discard much more discrete samples. *AVCLASS** has 2,466 singleton labels and 12 minority families, which are 1.75 and 4 times *AVCLASS*. The extra 1,060 discrete samples and 8 minority families are generated by removing the consumer engines, which indicates that they are double-count and not real label aggregation results in *AVCLASS*. The disappeared family labels contain *Zusy*, *Delphi*, etc. AVCLASS can still be a better choice if we consent to such costs.

We must state that our motivation for excluding these samples is not to disregard minorities. On the contrary, we think they are essential, and it is just that such minorities do not help us train the model and draw conclusions. At the same time, we believe that when the number of these discrete and minority family samples reaches a specific size, it is possible to use them as training data.

V. Discussion

A. Threats to Validity

Our experiment results show the label bias in malware classification and the limitations of existing two methods for label bias problems. However, there are some threats to validity.

The first threat of validity is related to the dataset. We used two datasets to perform the empirical study of label bias, i.e., an open source dataset (i.e., BODMAS) and a newly-collect dataset (i.e., MAIN). As the scale of the datasets could affect the generalization of our conclusions, we collect more than 153k real-world malware samples in the MAIN dataset. Both datasets appear to be unbalanced, and we have to clean the datasets to make the trained models work properly and facilitate outcome measurement. Here, we only reduce the size of the dataset by filtering out the extreme minority of samples and trying to ensure that the data distribution of the imbalanced dataset is consistent with the cybersecurity malicious sample distribution. Therefore, we believe that these two datasets can help us understand the label bias well.

Another threat of validity is related to the model training. The extracted features and selected models could affect the answers to the RQs. For this threat, we extract 5 types of features according to the paper [2] and the details are presented in the APPENDIX. We also choose the DNN algorithm to train the classification models as it is verified by several works [1, 2, 6]. We perform all experiments using 4-fold cross-validation and give averages to mitigate unexpected errors in the results. The case studies are performed and verified by the security experts in the cooperative security company.

B. Discussion and Future Work

According to the results of Table V, labels from reputable AV engines beat the label aggregation. However, a single reputable label source inevitably might have a relatively high rate of false positives, which makes researchers favor label aggregation. But for label aggregation, although we propose to eliminate the consumer AV engines, we have to pay the price of discarding some discrete samples. At the same time, these may be correctly classified in reputable AV engines. It does not interfere with binary identification, but it does affect multi-classification. Therefore, researchers need to make decisions based on their mission objectives, and we convince that minority families do matter in practice. Considering the need to classify minority families, we will pay more attention to them in our future work, such as few-shot learning [68, 69].

At the same time, features are also important, and it's not realistic to analyze each sample manually due to our limited ability to analyze samples. Even through automated means, it is not easy to obtain complete features, such as opcodes, instructions, etc., especially when the dataset is in a large scale. It will be better if much more datasets with diverse feature types are shared with the cyber security community, and we will maintain the malware dataset of MAIN open-source including static and dynamic features. We believe that vendors should share and maintain a common naming rule to facilitate research work because this mutual understanding can reduce unnecessary costs.

Although we offer some effective advice that seems soft, our main idea is to provide an inside view of label bias. We will make more practical solutions based on these suggestions in the future.

VI. Conclusion

In this paper, we conduct an empirical study of the label bias problems in machine learning based malware classification and existing solutions for the problems. We collect the MAIN dataset including 153,811 malware samples, and leverage a open-source dataset BODMAS for experiments. We present the existence of label bias problems including the chaotic naming rules and annotation inconsistency. And we find the limitations of existing solutions, i.e., selecting reputable AV engines faces the feature preference problem, and label aggregation based on majority voting ignores the independence problem. Then, we recommend users to select reputable engines according to the extracted features and eliminate consumer engines when aggregating labels. The results of improvements show that we get a performance increase.

Acknowledgment

We would like to thank the anonymous reviewers for their constructive comments. This research was supported in part by the National Nature Science Foundation of China (Grant No. 62232016, 61902384, and 62102406), Frontier Science and Technology Innovation Project (Grant No. 2019QY1403) and Outstanding Science and Technology Talent Program of ISCAS.

Appendix A
Feature Extraction

A. Static Feature Extraction

We follow the work[2] to extract the static features, including bytes entropy, printable strings, PE import function, and PE metadata. The static feature is a 1024-dimension vector, each type of feature has 256 dimensions.

Bytes Entropy. For each binary sample, compute the entropy $H(X)$ with a 1024-length sliding window at a step size of 256, and pair with the value of each byte within the window into a list. The $H(X)$ can be represented as Eq. (4), where

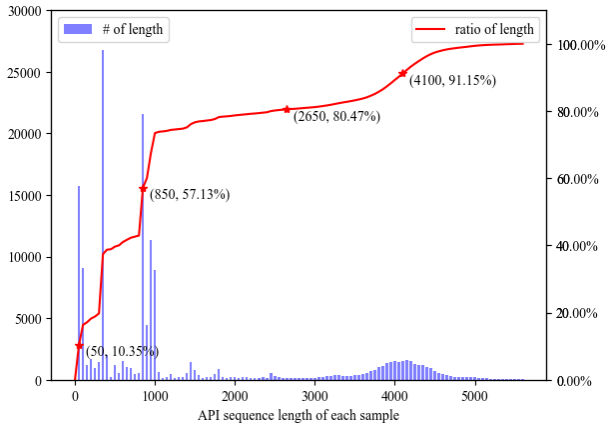

Figure 5: Distribution of dynamic API sequence length on MAIN dataset. Each blue bar represents the number of samples with specified API sequence length. The red curve represents the percentage of the MAIN dataset that contains samples of up to a specified length.

$P(x_i)$ is the probability of the byte x_i to appear in the current window.

$$H(X) = \sum_{i=1}^{n} P(x_i) I(x_i) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \quad (4)$$

Then, map the pairing list into a two-dimensional histogram, which will be divided into 16×16 bins. Finally, concatenate the each row of histogram to a 256-dimensional vector.

Printable Strings. Extract the printable strings which length is no less than 6, and hash the printable strings into the range $[0, 16)$. Then, take the log base 1.25 of each string's length, and pair the log of length and hash value into a list. Finally, map the strings pairing list into a histogram and concatenate the 256-dimensional vector as Bytes Entropy does.

PE Import Function and Metadata. Initialize two 256-dimensional vectors with all-zero. Utilize the 'pefile' (a python parsing library) to extract the DLL name and import funtions, and pair each import function with its DLL. Meantime, extract PE structure field information from the binary, such as NumberOfSections, TimeDateStamp, etc.. Then, hash the two groups of information into the 256-dimensional vectors, respectively.

B. Dynamic Feature Extraction

We first serialize every three consecutive APIs for preserving behavioral semantics as a 3-gram feature, for a dynamic behavior sequence with a length of L. Thus, we get a sequence of 3-gram features, which length is $L-2$. Second, according to the different label sources, we count the frequency of each 3-gram feature to calculate its information gain [66] in the dataset, respectively. Then, we get a ranking of 3-gram feature based on different label sources. Finally, for different label sources, we select different 1024-dimensional feature vectors to represent the samples of the whole dataset, whose 3-gram features information gain value is the top 1024 largest in the dataset according to the corresponding label source. We think the 1024 is enough as we take a further study of the

samples' execution traces of APIs and the results are presents as Figure 5.

APPENDIX B
ANTI-VRIUS ENGINE LIST

The AV engines appearing in Fig. 4, from left to right, are McAfee-GW-Edition, VBA32, Symantec, ESET-NOD32, APEX, Sophos, Arcabit, GData, Emsisoft, BitDefender, ALYac, Ad-Aware, MicroWorld-eScan, Avira, F-Secure, Kaspersky, ZoneAlarm, K7AntiVirus, K7GW, AVG, Avast, Cyren, DrWeb, Antiy-AVL, Rising, Ikarus, Microsoft, ClamAV, Fortinet, Comodo, NANO-Antivirus, Qihoo-360, Panda, Malwarebytes, Bkav, McAfee, AhnLab-V3, TotalDefense, VIPRE, Jiangmin, Yandex, Tencent, TrendMicro-HouseCall, TrendMicro, ViRobot, Zillya, Baidu, TACHYON, Alibaba, AegisLab, SUPERAntiSpyware, Zoner, CMC, Kingsoft, CAT-QuickHeal, Gridinsoft, MaxSecure, Babable, Avast-Mobile, SymantecMobileInsight, Malwarebytes3, CyrenCloud, SophosML, Kaspersky21, TheHacker, F-Prot, FireEye.

REFERENCES

[1] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[2] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[3] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[4] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 399–418. Springer, 2016.

[5] Junjie Liang, Wenbo Guo, Tongbo Luo, Vasant Honavar, Gang Wang, and Xinyu Xing. Fare: Enabling fine-grained attack categorization under low-quality labeled data. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2021.

[6] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *Proceedings of Deep Learning and Security Workshop (DLS), in conjunction with IEEE Symposium on Security and Privacy (IEEE SP)*, 2021.

[7] Mahinthan Chandramohan, Hee Beng Kuan Tan, and Lwin Khin Shar. Scalable malware clustering through coarse-grained behavior modeling. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–4, 2012.

[8] Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su. X-force: Force-executing binary programs for security applications. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 829–844, 2014.

[9] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. Goldeneye: Efficiently and effectively unveiling malware's targeted environment. In *International Workshop on Recent Advances in Intrusion Detection*, pages 22–45. Springer, 2014.

[10] 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.

[11] Robert McCluskey, Amir Enshaei, and Bashar Awwad Shiekh Hasan. Finding the ground-truth from multiple labellers: Why parameters of the task matter. *arXiv preprint arXiv:2102.08482*, 2021.

[12] Chengbin Pang, Tiantai Zhang, Ruotong Yu, Bing Mao, and Jun Xu. Ground truth for binary disassembly is not easy. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2479–2495, 2022.

[13] Jing Zhang, Victor S Sheng, Tao Li, and Xindong Wu. Improving crowdsourced label quality using noise correction. *IEEE transactions on neural networks and learning systems*, 29(5):1675–1688, 2017.

[14] Omar Alonso. Challenges with label quality for supervised learning. *Journal of Data and Information Quality (JDIQ)*, 6(1):1–3, 2015.

[15] Gaurav Sood. *virustotal: R Client for the virustotal API*. R package version 0.2.1.

[16] A new virus naming convention (1991). http://www.caro.org/articles/naming.html Accessed Aug. 30, 2022.

[17] Current status of the caro malware naming scheme. https://bontchev.nlcv.bas.bg/papers/naming.html Accessed Aug. 30, 2022.

[18] Bo Li, Phani Vadrevu, Kyu Hyung Lee, Roberto Perdisci, Jienan Liu, Babak Rahbarinia, Kang Li, and Manos Antonakakis. Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions. In *NDSS*, 2018.

[19] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, et al. Ad injection at scale: Assessing deceptive advertisement modifications. In *2015 IEEE Symposium on Security and Privacy*, pages 151–167. IEEE, 2015.

[20] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International symposium on research in attacks, intrusions, and defenses*, pages 230–253. Springer, 2016.

[21] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 45–56, 2015.

[22] Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1357–1365, 2013.

[23] 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.

[24] 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.

[25] Qi-anxin technology group inc. https://www.qianxin.com/ Accessed Sep. 10, 2022.

[26] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[28] Ziyun Zhu and Tudor Dumitraş. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 767–778, 2016.

[29] Jiayun Xu, Yingjiu Li, and Robert H Deng. Differential training: A generic framework to reduce label noises for android malware detection. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*, 2021.

[30] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining*, pages 1723–1732, 2017.

[31] 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.

[32] Ding Yuxin and Zhu Siyi. Malware detection based on deep learning algorithm. *Neural Computing and Applications*, 31(2):461–472, 2019.

[33] Mehadi Hassen and Philip K Chan. Scalable function call graph-based malware classification. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 239–248, 2017.

[34] Kesav Kancherla and Srinivas Mukkamala. Image visualization based malware detection. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 40–44. IEEE, 2013.

[35] Amine Boukhtouta, Serguei A Mokhov, Nour-Eddine Lakhdari, Mourad Debbabi, and Joey Paquet. Network malware classification comparison using dpi and flow packet headers. *Journal of Computer Virology and Hacking Techniques*, 12(2):69–100, 2016.

[36] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15(1):15–28, 2019.

[37] Daniel Gibert, Carles Mateu, and Jordi Planes. A hierarchical convolutional neural network for malware classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[38] Andrew Davis, Matthew Wolff, Michael Wojnowicz, Derek A Soeder, and Xuan Zhao. Neural attention mechanisms for malware analysis, July 11 2017. US Patent 9,705,904.

[39] Ida pro. https://hex-rays.com/ida-pro/ Accessed Aug. 30, 2022.

[40] Cuckoo sandbox. https://cuckoosandbox.org/ Accessed Aug. 30, 2022.

[41] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5(4):283–293, 2009.

[42] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14(1):1–20, 2018.

[43] Cheng Wang, Zheng Qin, Jixin Zhang, and Hui Yin. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 481–487. IEEE, 2016.

[44] Robert Searles, Lifan Xu, William Killian, Tristan Vanderbruggen, Teague Forren, John Howe, Zachary Pearson, Corey Shannon, Joshua Simmons, and John Cavazos. Parallelization of machine learning applied to call graphs of binaries for malware detection. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 69–77. IEEE, 2017.

[45] Stavros D Nikolopoulos and Iosif Polenakis. A graph-based model for malware detection and classification using system-call groups. *Journal of Computer Virology and Hacking Techniques*, 13(1):29–46, 2017.

[46] Shamsul Huda, Jemal Abawajy, Mamoun Alazab, Mali Abdollalihian, Rafiqul Islam, and John Yearwood. Hybrids of support vector machine wrapper and filter based framework for malware detection. *Future Generation Computer Systems*, 55:376–390, 2016.

[47] Mahboobe Ghiasi, Ashkan Sami, and Zahra Salehi. Dynamic vsa: a framework for malware detection based on register contents. *Engineering Applications of Artificial Intelligence*, 44:111–122, 2015.

[48] Çağatay Yücel and Ahmet Koltuksuz. Imaging and evaluating the memory access for malware. *Forensic Science International: Digital Investigation*, 32:200903, 2020.

[49] Jedidiah R Crandall, Gary Wassermann, Daniela AS De Oliveira, Zhendong Su, S Felix Wu, and Frederic T Chong. Temporal search: Detecting hidden malware timebombs with virtual machines. *ACM SIGOPS Operating Systems Review*, 40(5):25–36, 2006.

[50] Lok Kwong Yan and Heng Yin. Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 569–584, 2012.

[51] Harry Halpin and Roi Blanco. Machine-learning for spammer detection in crowd-sourcing. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[52] Amazon sagemaker. https://aws.amazon.com/sagemaker/ Accessed Aug. 30, 2022.

[53] Willem-Jan van den Heuvel and Damian A Tamburri. Model-driven ml-ops for intelligent enterprise applications: vision, approaches and challenges. In *International Symposium on Business Modeling and Software Design*, pages 169–181. Springer, 2020.

[54] Cedric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlaš, Wentao Wu, and Ce Zhang. A data quality-driven view of mlops. *arXiv preprint arXiv:2102.07750*, 2021.

[55] John Eric Steephen, Samyak Raj Mehta, and Raju Surampudi Bapi. Do we expect women to look happier than they are? a test of gender-dependent perceptual correction. *Perception*, 47(2):232–235, 2018.

[56] Bo Sun, Akinori Fujino, and Tatsuya Mori. Poster: Toward automating the generation of malware analysis reports using the sandbox logs. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1814–1816, 2016.

[57] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference*, pages 42–53, 2020.

[58] Aziz Mohaisen and Omar Alrawi. Av-meter: An evaluation of antivirus scans and labels. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 112–131. Springer, 2014.

[59] Virustotal += cylance. https://blog.virustotal.com/2017/07/virustotal-cylance.html Accessed Aug. 30, 2022.

[60] Virustotal += max. https://blog.virustotal.com/2017/07/virustotal-max.html Accessed Aug. 30, 2022.

[61] List of consumer av vendors (pc). https://www.av-comparatives.org/list-of-consumer-av-vendors-pc/ Accessed Aug. 30, 2022.

[62] Omar Alrawi, Charles Lever, Kevin Valakuzhy, Kevin Snow, Fabian Monrose, Manos Antonakakis, et al. The circle of life: A large-scale study of the iot malware lifecycle. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3505–3522, 2021.

[63] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *NDSS*, 2020.

[64] Daniel Berrar. Cross validation., 2019.

[65] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. Does every second count? time-based evolution of malware behavior in sandboxes. In *Proceedings of the Network and Distributed System Security Symposium, NDSS. The Internet Society*, 2021.

[66] Swati Jadhav, Hongmei He, and Karl Jenkins. Information gain directed genetic algorithm wrapper feature selection for credit rating. *Applied Soft Computing*, 69:541–553, 2018.

[67] Felipe N Ducau, Ethan M Rudd, Tad M Heppner, Alex Long, and Konstantin Berlin. Automatic malware description via attribute tagging and similarity embedding. *arXiv preprint arXiv:1905.06262*, 2019.

[68] Yude Bai, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Duoyuan 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)*, pages 1560–1571. IEEE, 2020.

[69] Candong Rong, Gaopeng Gou, Chengshang Hou, Zhen Li, Gang Xiong, and Li Guo. Umvd-fsl: Unseen malware variants detection using few-shot learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.