Decoding the
Secrets of Machine Learning in Windows Malware Classification: A Deep Dive into Datasets, Features, and Model Performance

Anonymous Author(s)

ABSTRACT

Many studies have proposed machine-learning (ML) models for malware detection and classification, reporting an almost-perfect performance. However, they assemble ground-truth in different ways, use diverse static- and dynamic-analysis techniques for feature extraction, and even differ on what they consider a malware family. As a consequence, our community still lacks an understanding of malware classification results: whether they are tied to the nature and distribution of the collected dataset, to what extent the number of families and samples in the training dataset influence performance, and how well static and dynamic features complement each other.

This work sheds light on those open questions by investigating the key factors influencing ML-based malware detection and classification. For this, we collect the largest balanced malware dataset so far with 67K samples from 670 families (100 samples each), and train state-of-the-art models for malware detection and family classification using our dataset. Our results reveal that static features perform better than dynamic features, and that combining both only provides marginal improvement over static features. We discover no correlation between packing and classification accuracy, and that missing behaviors in dynamically-extracted features highly penalise their performance. We also demonstrate how a larger number of families to classify makes the classification harder, while a higher number of samples per family increases accuracy. Finally, we find that models trained on a uniform distribution of samples per family better generalize on unseen data.

1 INTRODUCTION

Modern Windows malware analysis has to cope with a large number of samples that have been steadily increasing for two decades. In 2022, both the AV-TEST Institute and Kaspersky registered over 400,000 new malicious programs daily [19, 25]. In order to counter such numbers, research and industry have begun to rely on Machine Learning (ML)-driven malware classification models. They can be applied over a large number of files and offer more flexible classification mechanisms than signature-based methods. Nevertheless, they have to contend with human attackers' imagination, which consistently produces new variants to fly under the radar. At their core, ML techniques capture the statistical correlation between training data and classification targets. As a result, such statistics-based classification models lose their effectiveness when going beyond the knowledge encoded in the training data. Human attackers aware of this limitation can thus always be one step ahead to choose attacks unseen in the training data, in order to evade the detection of ML-based methods. Moreover, ML-based classification models are often performed in a pipeline [17, 30, 56]. For example, given a suspicious file, a typical ML pipeline should first figure out whether it is malicious (*binary classification*), and then find out whether it belongs to a known family (*family classification*). Even though these classification tasks achieve high accuracy in previous literature [17, 30, 56], most of these works have been carried out with unrealistic assumptions, mainly because of how the dataset was constructed.

In addition, a ground-truth of malware families is hard to obtain. Antivirus companies will not likely use the same name for the same family. Although the CARO (Computer Antivirus Research Organization) naming convention has been proposed to mitigate this issue, it still faces usage obstacles. Scientific research tackled this problem and produced AVClass [49]: given a list of AV labels (e.g., from a VirusTotal JSON report), the tool returns the *single most likely* family name. However, even if AVClass returns a single family name according to a consensus algorithm by default, it can also output a ranking of all alternative family names. Thus, the problem is that AVClass is often used to carry out studies using its default output as ground truth, even though it is probabilistic in nature.

Moreover, while it is straightforward to collect a high number of samples for popular families, collecting a large diverse malware dataset remains difficult and time-consuming [8, 27, 37, 51]. In this work, we collect PE malware executables from the VirusTotal (VT) feed [55], a real-time stream of JSON-encoded reports of samples submitted to VirusTotal. Despite the appearance of more than 44M VT reports over a period of nearly three months and the collection of 227k samples from 13.8k families, only 780 malware families of those contain at least 100 samples.

To further complicate the matter, malware authors often use off-the-shelf packers and protectors [32, 34]. Both modify a program to hinder its analysis while still preserving its original behavior. Based on their design, different malware that undergo the packing or protection procedures may generate executables that share a highly similar structure. This easily makes a ML classifier trained over these malware samples overfit the packed or protected file structure, rather than capturing its true malicious component.

Therefore, in this work, we put considerable effort to create four heterogeneous datasets for a total of 118,111 samples to perform a large-scale measurement study. Three of them are composed of malicious samples with varying numbers of families, while the fourth contains benign samples. We devoted particular attention during the construction of the datasets, trying both to reproduce the datasets usually used in research, but also considering real-world scenarios typical of malware analysis. Such datasets allowed us to create well-controlled experiments for studying how the effectiveness of ML-based binary and family classification change under different testing scenarios.

Finally, there is also another crucial aspect that influences ML algorithms that we further explored: *feature* extraction. The methods by which one can analyze executable files fall into two main categories, depending on what facets one wants to study, namely *static* properties and *dynamic* behavior; nonetheless, the previous

Table 1: Dataset summary

Dataset	Samples	Families
Malware Balanced (M_B)	67,000	670
Benign (B)	16,611	-
Malware Unbalanced (M_U)	18,000	1,500
Malware Generic (M_G)	16,500	-
All	118,111	-

two can also be *combined*. Since we wanted to study existing ML state-of-the-art solutions and **not** design new ones, we build our static and dynamic feature extraction approaches on what was described in recent papers [7, 9]. Therefore, this means that we have statically analyzed and dynamically executed in a sandbox more than a hundred thousand samples used in this study.

With this in mind, our work aims to contribute by answering the following research questions, in both binary and family classification tasks.

⟨**R1**⟩ **How do static, dynamic, and combined models perform on different malware families/classes in binary and family classification?**

⟨**R2**⟩ **Do missing feature values in the run-time behaviors negatively impact the classification results?**

⟨**R3**⟩ **Does the presence of off-the-shelf packers and protectors bring harm to classification accuracy?**

⟨**R4**⟩ **Is the AVClass2 confidence score correlated with ML-based decisions?**

⟨**R5**⟩ **What is the contribution of static and dynamic feature classes to the classification performance and does their contribution change when joining the two sets?**

⟨**R6**⟩ **On which families and classes of malware does each model fail to produce accurate classification?**

⟨**R7**⟩ **How does the training dataset construction strategy affect the model performance?**

⟨**R8**⟩ **How does the ML-driven malware classifier perform over the families unseen in the training data?**

2 DATASET COLLECTION

To conduct our experiments we collected 118,111 Windows PE32 executables, divided in four datasets, as summarized in Table 1. This section describes the process for building those datasets.

2.1 Malware Samples

We collect PE malware executables from the VirusTotal (VT) feed [55]. The VT feed is a real-time stream of JSON-encoded reports. Each report contains the analysis results of a sample submitted to Virus-Total – including file hashes, filetype, size, and the detection labels assigned by a large number of antivirus (AV) engines. These reports are generated both by new samples submitted by VT users, as well as by user-requested re-analysis of files already known to VT. Samples in the feed can be of various filetypes (e.g., PE, APK, PDF), but our collection focuses on Windows PE executables. Samples that appear in the feed can be downloaded within 7 days from the moment they appear in the feed.

We want our dataset to be as diverse as possible in terms of number of families, but also to be balanced, so that no malware family

is over-represented or under-represented. Our initial target was to collect 1,000 malware families with a hundred samples each. The threshold of 100 samples per family was chosen to have enough samples per family for performing multi-class classification experiments, taking into account that samples are split into 60% training, 20% validation, and 20% testing. However, due to the collection, filtering, and reclassification process described below, we ended up with 670 families satisfying that threshold, as shown in Table 1.

To the best of our knowledge, this is the most diverse labeled malware dataset in terms of families to date. The most recent dataset was Motif [22] with 454 families. While the number of families in Motif is also large, it is 21 times smaller than our balanced dataset with 3,095 samples, and is unbalanced with a median of 3 samples per family and a standard deviation of 10.4. Only one family in Motif reaches 100 samples and 29% of the families have only one sample. Such small number of samples for most families does not allow building an accurate multi-class classifier, as we will show in our evaluation.

Initial collection from VT feed. We collected reports and samples from the VT feed for 82 non-consecutive days between August 2021 and March 2022. We only retained reports of samples detected by at least one AV engine, and with a trID [39] filetype identification field (available in the report) equal to '32-bit non-installer PE executable'. We excluded 64-bit PE executables, dynamic-link libraries (DLLs), and executables generated by popular installer software (e.g., NSIS, InnoSetup). These restrictions are placed by our dynamic analysis sandbox, described in Section 3.2, which currently does not support running 64-bit PE executables or DLLs, and does not interact with GUIs in order to complete the installation of other programs. However, an analysis of the whole VT feed during the 82 collection days shows that from all malicious PE samples in the feed, 87.6% are 32-bit executables, 8.2% are DLLs (32-bit or 64-bit), 3.9% are 64-bit executables, and the remaining 0.3% are other PE types (e.g., OCX, CPL, SCR).

The retained reports are fed to the AVClass2 malware labeling tool [49], which outputs the most likely family name for the sample as well as a confidence factor that captures the number of AV engines assigning that family to the sample (after removing duplicates due to AV engines that copy each other). For each family reported by AVClass2, our system downloaded 100 distinct samples. Each downloaded sample was then checked again to exclude any remaining non-32-bit PE executables and installers that were missed by trID. In particular, samples are removed if their PE header does not indicate that it is 32-bit executable, or if they are detected as installers using public Yara rules by Avast [6]. As stated, our initial target was to collect 1,000 malware families with 100 samples each. However, when this target was reached, many other families had been collected with less than 100 samples, resulting in an initial dataset of 239,417 PE32 malware samples from 23,555 families.

Reclassification and family filtering. The AV labels of a sample may change over time as AV vendors refine their detection rules. These label changes may in turn change the family that AVClass2 outputs for a sample. To account for such changes, we re-collect the updated VT report for our samples 54 days after the end of our collection process, and feed the new reports to AVClass2 to obtain the (possibly) updated family. From the 239,417 samples, 9.7% (23,171) were at this point re-classified as a different family. AVClass2 uses a taxonomy to identify a wide range of non-family

tokens that may appear in the AV labels. These include file properties (e.g., *FILE:packed:asprotect*, *FILE:exploit: gingerbreak*), malware classes (e.g., *CLASS:virus*, *CLASS:worm*), behaviors (e.g., *BEH:ddos*, *BEH:filedelete*), and generic tokens (e.g., *GEN:malicious*, *GEN:behaveslike*). However, the AVClass2 taxonomy is assumed to be incomplete by design [49]. Thus, it may output a label for a sample that does not correspond to a real family, but rather to a previously unknown instance of the above categories. To address this issue, we manually inspected the collected family labels and conservatively filtered out any labels that may not correspond to real family names. This step identified 86 likely non-family tokens not in the AVClass2 taxonomy, such as *gametool*, *testsample*, *nsismod*, *dllinject*, and *processhijack*. We also removed all random-looking labels (e.g., *005376ae*) that AVClass2 fails to automatically identify. As byproduct of our effort, we will contribute our extended AVClass2 taxonomy to the open-source AVClass2 project.

After reclassification and family filtering, the dataset contained 227,296 samples from 13,894 families, out of which 780 families had at least 100 samples.

Despite collecting more than 44M VT reports over a period of nearly 3 months, we were unable to reach our goal of 1,000 families with 100 samples. In fact, out of over 14K families that appeared in the VT feed, very few included a significant number of unique samples, making the collection of a diverse malware dataset difficult and time-consuming.

Feature filtering. We performed static and dynamic feature extraction (as detailed in Section 3) for all samples of the 780 families with at least 100 samples. This required to execute each sample in a sandbox to obtain a behavioral report. We then excluded samples with feature extraction errors, e.g., those that did not exhibit any runtime behavior, and sub-sampled families to keep only 100 samples each. The result is a balanced dataset (hereinafter M_B) that contains 67,000 samples from 670 families. According to AVClass2, those families belong to 13 malware classes: 36% (282) of the families are classified as grayware (including its adware subclass), 15% (120) as downloaders, 11% (87) as worms, 10% (78) as backdoors, 5% (41) as viruses, and the remaining 23% (62) includes ransomware, rogueware, spyware, miners, hacking tools, clickers, and dialers.

Dataset statistics. Over 93% of the samples in the M_B dataset are detected by at least 20 AV engines, while only 0.3% have a VT score less or equal to 3. It is worth noting that the minimum number of detections for samples in the dataset is two since AVClass2 requires at least two AV engines to assign a label to a sample.

Samples on the VT feed can be new (i.e., collected and scanned for the very first time by VT) or resubmitted (i.e., first submitted in the past but re-scanned on the day they were collected). We compute the freshness of samples in the M_B dataset as the number of days between a sample's collection date and its VT first seen date. We observe that 53.4% of the samples were collected within a day of being first observed by VT, 7.6% within a year, and 37.8% are old samples first seen over one year before our study.

Packer and protector detection. To hamper analysis, malware authors may use packers that compress a sample and de-compress it at runtime, as well as more sophisticated protectors that may combine different obfuscations such as packing, encryption, and code virtualization. To evaluate the impact of packers and other protectors on malware classification, we determine whether a sample uses an off-the-shelf packer or protector by using the signature-based Detect It Easy (DIE) [2] tool, as well as the well-maintained Yara rules of Avast RetDec [6]. Overall, 22% of the samples in M_B use a packer or protector. The most popular packer is upx detected on 14.0% the samples, followed by aspack (3.2%) and pecompact (1.0%). The most popular protectors are vmprotect (1.9%) and asprotect (0.4%).

2.2 Testing Datasets

We create two other disjoint malware datasets, which we use in Section 4 to test the ability of ML classifiers to generalize beyond the M_B dataset they were built upon. The first dataset, referred as Malware Unbalanced (or M_U) in Table 1, contains 18K samples from 1.5K families. These samples were part of the initial VT feed collection, passed the filtering and re-classification steps, but their families never reached the threshold of 100 samples and thus were excluded from M_B. All samples are detected by at least 20 AV engines and none of the samples nor their families are part of M_B.

The second dataset, Malware Generic (M_G), contains 16.5K samples for which AVClass2 was unable to output a family, due to AV engines using only generic labels. These samples were separately collected from the VT feed between June 23rd and July 6th 2022 and underwent the filtering steps to keep only 32-bit non-installer PE executables. All samples are detected by at least 20 AV engines and none of the samples are part of M_B.

2.3 Benign Samples

Building a benign dataset by just relying on the number of AV detections in the VT report is prone to errors due to the presence of malicious files that are still unknown to AV engines. Therefore, we took a more conservative strategy and decided to build a benign dataset by using a fresh installation of all the community-maintained packages (which undergo a rigorous moderation review process to avoid pollution) of Chocolatey [1] in a clean machine running Windows 10. After each package was installed, we extracted all the executable files present on the hard disk, which may correspond to Windows system files or third-party publishers.

We exclude files that are not 32-bit PE executables and those with more than three detections on VT. This allowed us to discard borderline cases, i.e., benign files with characteristics very similar to malware, like hacking and scanning tools. Using this procedure we collected a dataset B of 16,611 benign samples. The code signatures of those samples indicate a large diversity of publishers with over 1.4K different signers – including both small companies and large software publishers such as Microsoft, Oracle, and Google.

3 METHODOLOGY

Our work aims to answer the 5 research questions raised in Section 1. These questions are all closely associated with the applicability of ML-based malware detection (i.e., binary classification) and family classification (i.e., multi-class classification) in practice. Notably, we aim to explore the performances of ML-driven malware classifiers that use features extracted statically, dynamically, or a combination of both with varied coverage of malware families and changed volumes of training samples. It is worth noting that developing novel ML-based malware classification models is beyond the scope of our study.

Table 2: Feature classes used in the classifiers.

ID	Class	Extraction	Features
s-headers	PE headers	static	29
s-sections	PE sections	static	590
s-dll	DLL imports	static	131
s-imports	API imports	static	3,732
s-bytegrams	Byte n-grams	static	13,000
s-opcodegrams	Opcodode n-grams	static	2,500
s-strings	Strings	static	10,402
s-file	File Generic	static	2
d-network	Network activity	dynamic	438
d-file	File activity	dynamic	60,555
d-mutex	Mutexes used	dynamic	7
d-registry	Registry operations	dynamic	60
d-service	Services activity	dynamic	736
d-process	Process activity	dynamic	28,198
d-thread	Thread actitivy	dynamic	7

Instead, we focus on discussing and evaluating the analysed issues using state-of-the-art ML models applied for malware classification.

3.1 Static Features

Hojjat et al. [7] performed a literature review to identify the static features that carry the most useful information for binary classification. We implement their feature extraction methodology to extract the same classes of static features. However, the final number of features in each class may differ as some include a feature selection step that retains only the features that provide more information, which is dependent on the training dataset.

The upper half of Table 2 summarizes the static feature classes (prefixed by s-). The *s-headers* class captures 29 integer features from the *Optional* and *COFF* headers of the executable [13]. The *s-sections* class captures 590 Boolean features from the 32-bit characteristic field of each section in the executable. The *s-dll* and *s-imports* capture 131 and 3,732 Boolean features, respectively, extracted from the import table. *s-dll* features capture the imported libraries while *s-imports* features capture the imported functions in those libraries. The *s-bytegrams* class capture 4-grams, 5-grams, and 6-grams extracted from the raw sequence of bytes of 1,000 randomly chosen files in our dataset, and filtered to keep only the 13,000 n-gram values appearing in more than 1% of the samples and with higher information gain. The *s-opcodegrams* class captures 3-grams and 4-grams built from the sequence of opcodes disassembled using Capstone [12] and filtered to keep the top 2,500 n-gram values with higher information gain. The *s-strings* class comprises of 10,402 Boolean features capturing strings of at least 4 characters that appear in over 1% of the files. Finally, the *s-file* features capture the file size in bytes and the whole file Shannon entropy [31]. We refer the interested reader to [7] for a more detailed description of the features and the works that originally proposed them.

3.2 Sandbox

We have built a sandbox for executing malware using the best practices proposed by previous works [32, 34, 44, 57]. We configured a Windows 10 Pro 32-bit virtual machine (VM) with 2 CPUs (Intel Xeon Platinum 8160 @ 2.10GHz) and 2 GiB of RAM. We installed popular

apps and populated the file system with common file types to resemble a legitimate desktop workstation as suggested by Miramirkhani et al. [34]. Malware runs on clones of this VM orchestrated using Proxmox VE [5]. To improve performance, we stored all virtual disk images and VM snapshots in a RAM disk. As recommended by Rossow et al. [44], each machine runs on its own isolated local network with full Internet access through an ADSL line of our institution dedicated for this purpose. Recent works have measured that 40%–80% of modern malware use at least one evasive technique [15, 32]. To limit the impact of such evasions, we base our analysis on the Intel PIN-based *JuanLesPIN* tool [3, 32], which handles common evasive techniques, thereby increasing the likelihood that malware detonates. Unfortunately, it does not support 64-bit Windows executables, so we focus on 32-bit malware. We modified *JuanLesPIN* to monitor Windows APIs responsible for network, processes, services, registry, mutexes, file system, and DLL loading. We tested our analysis environment with the Al-Khaser [4] tool to confirm that our sandbox could not be identified. To analyze sandbox detection techniques that measure overhead [32] we executed 1,000 malware samples randomly chosen among those that: (i) terminate the execution, (ii) use at least one evasive technique, and (iii) call at least 50 Windows API (a threshold proposed in [26] to determine whether a sample has detonated). We measured their execution time with and without instrumentation observing a percentage increase of $\mu = 125$, $\sigma = 31$, $min = 26$, $med = 106$, $max = 206$. This overhead is in line with that in [32]. Kuechler et al. [26] recently showed that the amount of code executed by malware samples plateaus after two minutes, and little additional information can be obtained thereafter. We took a conservative approach and ran each sample for up to five minutes.

3.3 Dynamic Features

We extract 7 classes of dynamic features from the API calls (including their arguments) invoked by the malware during execution in the sandbox. The features were chosen to cover those used in previous works that built classifiers from malware executions (e.g., [9, 17, 18]).

The lower half of Table 2 summarizes the 7 dynamic feature classes (prefixed by d-). Categorical features such as filenames and domains are one-hot encoded to Boolean features. To encode each feature, we count all its possible values and exclude those appearing less than five times in the training set. The *d-network* class (438 features) captures the HTTP, TCP, and UDP traffic. Of those, 430 features capture unique domains contacted by the malware and HTTP User-Agent strings used; three count the number of HTTP requests, TCP connections, and UDP pseudo-sessions; and 5 randomness-related features capture the mean/median/min/max/std likelihood of domain names and URLs contacted according to a recently proposed Markov Chain model [9]. The *d-file* class features (60,555) capture the name and extension of 60,547 files created or accessed by the malware, the number of files read, written, and deleted; and 5 capture the randomness of the filenames. The *d-mutex* class features (7) capture the number of mutex objects created and the randomness of the mutex names. The *d-registry* class features (60) capture 55 unique registry keys written, and the count of registry keys created, opened, read, written, and deleted. The *d-service* class features (736) capture the count, randomness, and names of services and service managers

created, started, and halted. The *d-process* class features (28,198) capture the count of processes created, processes terminated, and shell commands invoked, as well as 28,195 unique process names. The *d-thread* class features (7) capture the number of the threads opened, created, resumed, terminated, and suspended, as well as the number of the interactions with the context of a given thread and the number of asynchronous procedure calls (APC) queued to a thread. The last two features help capture suspicious behaviours.

Missing features. When a dynamic feature cannot be computed (e.g., due to lack of activity), we assign it a default place-holder value that does not belong to the domain of the feature. We refer to such features as *missing* features. For example, if a sample has no file system activity, we cannot compute the *d-file* filename randomness features. As a result, the 5 statistical features related to the randomness of the file names are thus not available. To facilitate the following analysis over the impact of the missing features, we define the *feature missing rate* (FMR) of a malware family as the fraction of family samples that have missing values in the file, registry, service, and process features (which, among the seven dynamic features classes we consider, are the most relevant for classification according to Table 4). Missing values over all these four feature classes considerably degrades both the amount and quality of useful information available to the classifier. According to our analysis, over 54% of the malware families studied in our work contain on average 77% of the malware samples per family with missing feature values in these four dynamic feature classes. Missing observations can negatively impact ML classifiers by overfitting the data and reducing the model's accuracy. Recently, Aonzo et al. [9] showed that classifier models tend to focus on static features, rather than dynamic ones, precisely because static features are rarely missing. In Section 4.6 we analyze the impact of missing features in the classification results.

3.4 Models

We train multiple models to capture different axis: classification task (i.e., binary or family classification), features (i.e., static, dynamic and combined), dataset construction (i.e., distribution of families in training dataset), and number of families and samples. All the models are *Random Forest* classifiers whose hyper-parameters are each time tuned by using a validation set. Random forests are intrinsically capable of handling categorical features (e.g., unique filenames accessed during execution) and continuous features (e.g., filename mean randomness).

Classification task. We build models for binary and family classification tasks. The binary classification models detect whether a given sample is malicious (positive class) or benign (negative class). The family classification models identify the family of a given malicious sample, that is, there is one class per malware family and no goodware class. We prefix the name of a model with *binary-* or *family-* to indicate the classification task.

Features. We build models that use all static features, all dynamic features, and all combined features (i.e., all static and all dynamic). The name of a model includes *-static-*, *-dynamic-*, or *-combined-* to indicate the features used. Dynamic and combined models only include samples that called at least 50 APIs during their execution, a threshold previously proposed to determine whether a sample has detonated [26].

Dataset construction. For the binary classification task, we experiment with two ways of building our dataset, namely *uniform* and not *nonUniform*. The *uniform* approach builds datasets that balance the number of goodware and malware, using a sampling-with-replacement approach, as follows. We uniformly select from each family in M_B a number of samples so that the total number of malicious samples matches the size of the benign dataset (i.e., each family in M_B provides 24–25 samples for a total of 16,611 malware samples). We repeat the process five times avoiding repetitions (i.e., each time selecting a different set of malware samples from each family in M_B), as to completely cover all the malicious samples in each family. These steps produce 5 balanced datasets. Each dataset is split into 60% of samples for training, 20% for validation, and 20% for testing. To evaluate a model, for each of the five datasets, we perform a 10-fold cross validation to ensure that all the samples equally contribute to the training and testing datasets. We report average results across the five rounds and their respective folds. Thus, obtaining the accuracy results from one model requires us to train and test 50 times.

The *nonUniform* approach replicates the unbalanced distribution of samples per family in the Motif dataset [22]. The motivation for this dataset is to study whether the family distribution in the training set of a binary classification task (where family labels are not used) affects the detection accuracy. In Motif, 29% of families have only one sample, 41% have 2-5 samples, 12% 6-10, 10% 11-20, 4% 21-30, 2% 31-40, 1% 41-50, and 1% has over 142 samples. We replicate this distribution on the 670 families in M_B. For example, we select one sample from 29% (randomly-chosen without replacement) of the 670 M_B families and 142 samples from one randomly-chosen family. The resulting dataset comprises all 16,611 benign samples and 4,821 samples from 670 families that follow the per-family sample distribution in Motif.

Number of families and samples. To measure the impact that the number of families to classify and the available samples for each family have on the results, we build multiple ML-based classifiers for the family classification task by uniformly sampling 70, 170, 270, 370, 470 and 570 families from the total 670 families. For each of them, we also experiment with a version trained and tested on 50, 60, 70 and 80 malware samples for each family. As indicated above, we have 20% samples used as the validation data. Therefore, at maximum, there are 80 malware samples for training and testing use.

4 EXPERIMENTAL STUDY

This section presents the results of the experiments we conducted to answer the research questions presented in the introduction. We have adopted the following structure for ease of reading: the reader will find the discussion to ⟨**Rx**⟩ in Section 4.x and a summary with the answer ⟨**Ax**⟩ at the end of each subsection.

4.1 Overall Classification Results

In this section, we examine how static, dynamic, and combined features impact binary and family classification. Table 3 summarizes the accuracy results for the three feature sets. It also reports the fraction of malware families with 100% family-wise accuracy. In binary classification, 100% family-wise accuracy for a family denotes that the family can be perfectly differentiated from goodware. In family classification, 100% family-wise accuracy instead means that samples from a malware family are not misclassified as another malware

Table 3: Overall results

Task	Features	Precision	Recall	F1-score	Families 100% classified
Binary	Static	0.956	0.957	0.957	242 (36.12%)
Binary	Dynamic	0.945	0.892	0.926	465 (69.40%)
Binary	Combined	0.963	0.934	0.948	450 (67.16%)
Family	Static	0.856	0.850	0.848	68 (10.15%)
Family	Dynamic	0.734	0.708	0.704	114 (17.17%)
Family	Combined	0.874	0.867	0.865	138 (20.60%)

family. The results correspond to the uniform dataset construction approach. Each line of the table report the averaged precision, recall and F1 score of 10-fold cross validation. Besides, we also give the fraction of malware families that are 100% classified using different features and in different tasks.

The static features achieve higher F1 score than the dynamic features in both binary and family classification. However, the fraction of perfectly classified malware families is higher for dynamic features, suggesting that dynamic features work very well for some malware families, but poorly on others. The combination of static and dynamic features brings marginal improvements in F1 score over static-only features. It improves it by 1% for family classification, but decreases it by 2% for binary classification. On the other hand, adding dynamic features increases the percentage of perfectly classified families over the static case, although for binary classification the fraction reduces compared to dynamic-only features. The accuracy reduction with more features might seem counter-intuitive, but it can happen when the two feature sets are not independent and bring different strengths and weaknesses that lead to mistakes on different parts of the input space. It is well known as the curse-of-dimensionality in machine learning [53]. Adding more features does not necessarily improve the overall accuracy, more features may bring unexpected variance and noise into the classification module [28].

Our results may raise concerns on the value of dynamic analysis. On the one hand, dynamic features outperform static features for a fraction of families, significantly raising the number of perfectly classified families (e.g., nearly doubling it for binary classification). This confirms the value of dynamic analysis, for example when researchers are interested to build behavioural signatures for specific malware families. On the other hand, the overall impact of adding dynamic features to static features is unclear. This might be the consequence of malware families for which dynamic features do not work well, because of intrinsic properties of the malware family (or malware class), but also because the sandbox might fail to stimulate samples adequately (e.g., due to evasion techniques or to the lack of a live command-and-control server).

⟨**A1**⟩ For both binary and family classification tasks, models trained on static features alone provide higher accuracy than the models trained only on dynamic features. The latter are able to perfectly classify more families, but perform poorly on others, producing an overall lower classification accuracy.
Adding dynamic features on top of the static features brings marginal improvement for family classification and even negatively affects binary classification.

4.2 Impact of Missing Dynamic Feature Values

A possible explanation for the worse results of dynamic features compared to static features is that a sandbox may fail to stimulate samples adequately to cause them to 'detonate', or simply sample may not work properly due to missing local or remote components. As a result, the classifier might need to take a decision only based on a limited snapshot of the malware runt-time behavior.

We computed the Pearson correlation coefficient between the family-wise recall score of binary classification and the FMR to study the link between the two. Interestingly, the correlation is not statistically significant for the binary classification task (pearson -0.1 and p-value 0.11). However, there is a clear negative correlation (-0.43, p-value of $7.61 * 10^{-16}$) for the family classification task. In this case, as the fraction of samples with missing feature values for a family increases, its classification accuracy decreases. This is also confirmed by looking at the malware families that are the most difficult to classify with dynamic features , i.e., those for which the classifier has the lower accuracy (see Table.11 and Table.12 in Section.4.6). Among the top-10 all have an FMR > 65%.

This outcome demonstrates that the ML classifier might still be able to identify signs of malicious behavior in incomplete dynamic analysis reports, but more feature values are needed to precisely distinguish among different families (in particular for those, like downloaders, that might have similar behavioral profiles). In addition, binary classification is also affected by the quality of the behavior collected from benign samples, while family classification accuracy is solely associated with the feature completeness of malware samples in each family.

⟨**A2**⟩ Globally, a statistically significant inverse correlation in the family classification task between the family-wise classification accuracy using dynamic features and the amount of missing dynamic feature values exists. The correlation is instead not significant for the binary classification task.

4.3 Impact of Packers and Protectors

This section evaluates whether the presence of off-the-shelf packers and protectors harms the classification accuracy when considering static features. In this experiment, we try to understand whether the models overfit the packing/protection technology structure or are able to capture data that allows them to classify samples correctly.

To answer this question, we first compute the family-wise classification accuracy for both binary and family classification using static features. We then compute the Pearson correlation scores between the family-wise accuracy scores and the rate of packed samples in each family. If packing negatively affects the ability to classify a sample, we would expect lower accuracy for families where packing is more prevalent.

However, the correlation scores are 0.015 (p-value 0.5) and 0.0001 (p-value 0.98) respectively for binary and family classification. Thus, we can reject the hypothesis that classification accuracy is correlated with the packing rate.

This might seem surprising, as one might expect a high correlation between packing and misclassification rate at least for models that rely only on static features. After all, packing was one of the main reasons that led researchers to introduce malware analysis

sandboxes and dynamic analysis. However, this is a common misconception. In fact, while packing is very effective at impeding static *analysis* (i.e., the ability to examine a sample and statically derive its behavior), other works [7] have shown that common packers leave certain areas of the binary untouched, thus having a limited effect on the ability of a ML *classifier* to identify a sample.

> ⟨**A3**⟩ Packed or protected samples (with off-the-shelf tools) do not significantly correlate with their classification accuracy using static features. This means that although these technologies function well to deter static analysis (in particular reverse engineering), do not significantly affect ML classifiers.

4.4 Impact of Ground Truth Confidence

To assign a family to a sample AVClass2 computes a list of (tag, confidence) pairs, e.g., (FAM:sality, 5), (CLASS:virus, 4), (FAM:zpevdo, 1). Then, it selects as family the highest confidence tag that is either a family in its taxonomy or an unknown tag not in its taxonomy. The confidence score roughly represents the number of AV engines that assign a tag to the sample, after accounting for aliases and discounting groups of AV engines that copy their labels. This section examines whether the AVClass2 confidence score for the selected family impacts the classification accuracy.

To examine this issue, we first compute the confidence score for each family. For each sample, we obtain a normalized confidence in the [0,1] range by dividing the confidence score of the assigned family over the sum of the confidence scores for all family and unknows tags for the sample. In the case above, this step returns 0.83 as the *FAM:sality* confidence was 5, but *FAM:zpevdo* also appeared in the output. Then, we average the normalized confidence factor across all samples in the family to produce a family confidence score.

Next, we compute the correlation between the family-wise classification accuracy and the family confidence score. The hypothesis is that higher family confidence scores correlate with higher family classification accuracy, i.e., the more agreement AV engines have when tagging the sample, the easier it should be to classify the sample. The pearson correlation coefficient is 0.083 for static features (p-value 0.03) and 0.062 for dynamic features (p-value 0.01). The correlation is positive, but extremely small. Thus, we can conclude that poor family classification is not influenced by a low AVClass2 confidence score, and the result is statistically significant. This is further confirmed by examining the 10 families with lowest classification accuracy using either static-only or dynamic-only features (Table 9 and Table.11 in the Appendix). Of those 20 families, all have a confidence score above 0.5 and 15 have a confidence score above 0.8. This suggests that even when the AV engines do not fully agree on the name of a sample, the majority vote likely selects the correct family, which provides further confidence on our AVClass2-based ground truth generation approach.

> ⟨**A4**⟩ The accuracy of family classification is not correlated with the AVClass2 confidence score, which captures the agreement between different AV vendors on the family name of a sample. This observation supports that AVclass2 is a valid tool for getting ground truth when it is necessary to obtain the family name of malware.

Table 4: Feature class importance using MDI score.

Feature Class	Binary classification			Family classification		
	Comb.	Static	Dyn.	Comb.	Static	Dyn.
s-bytegrams	40.88	51.38	-	38.60	41.67	-
d-registry	17.19	-	25.00	0.51	-	0.60
s-opcodegrams	13.44	21.08	-	23.48	20.87	-
s-strings	9.09	15.27	-	17.62	19.27	-
d-file	7.74	-	29.70	3.16	-	56.20
s-sections	3.05	6.73	-	5.62	6.48	-
s-imports	2.48	4.17	-	7.87	9.30	-
d-thread	2.06	-	7.34	0.16	-	5.26
d-network	1.51	-	3.50	0.35	-	3.70
d-process	1.47	-	32.90	0.87	-	30.70
s-headers	0.34	0.72	-	0.73	0.96	-
d-mutex	0.25	-	0.16	0.03	-	1.19
d-service	0.19	-	1.40	0.07	-	2.39
s-dll	0.17	0.28	-	0.52	0.57	-
s-file	0.13	0.35	-	0.39	0.87	-

4.5 Feature Class Importance

This section examines the importance of the static and dynamic feature classes for binary and family classification. We measure the importance of each feature class using the average Mean Decrease Impurity (MDI) score. In a tree-based classifier, the MDI score of a feature captures how often the feature was used in the tree. The more a feature is used, the more important it is to distinguish different classes. We average the MDI Score across all the features belonging to the same feature class and over all trees in the Random Forest model.

Table 4 summarizes the feature class importance. Overall, static features are ranked higher than dynamic features, especially for family classification. This matches results in Section 4.1 where dynamic features provide marginal improvements over static features. This observation is in line with recent findings that although humans prefer dynamic features, ML algorithms rely more on the *always present* static features [9].

The most contributing static feature classes for both classification tasks are *s-bytegrams*, *s-opcodegrams*, and *s-strings*. This confirms what was previously observed in the literature, with raw and opcode ngrams dominating over other static features [7]. On the other hand, the most contributing dynamic feature classes for both classification tasks are *d-file* and *d-process*. It is interesting to note that even expert human analysts used widely file and processes operations to identify malicious behaviours [9].

In our dataset, over 50% of the malware samples contain missing features values in the *d-network* and *d-service* feature classes, thus missing feature values is likely the reason for their low importance. It is interesting that *d-registry* ranks second for binary classification, but only 10th for family classification. This means that registry operations are useful to differentiate malware from goodware, but they do not provide enough diversity to separate different malware families. This likely happens because multiple malware families operate on the same registry keys such as those related to achieving persistence (e.g., auto-start) and those that disable OS security features. In contrast, goodware does not need to operate on those keys.

Table 5: Classification accuracy for malware classes.

| Class	Binary class. Recall			Family class. F1 score		
	Static	Dyn.	Comb.	Static	Dyn.	Com.
Adware	0.905	0.915	0.981	0.926	0.761	0.925
Backdoor	0.966	0.943	0.996	0.830	0.730	0.838
Clicker	0.971	0.929	1.000	0.817	0.692	0.821
Dialer	0.994	0.875	1.000	0.988	0.888	0.984
Downloader	0.974	0.899	0.996	0.864	0.695	0.874
Grayware	0.932	0.895	0.986	0.832	0.675	0.852
Miner	0.989	0.972	0.999	0.927	0.807	0.962
Ransomware	0.967	0.945	0.997	0.839	0.580	0.853
Rogueware	0.984	1.000	0.992	0.616	0.401	0.663
Spyware	0.972	0.829	0.998	0.869	0.704	0.879
Tool	0.992	0.929	1.000	0.864	0.778	0.830
Virus	0.885	0.939	0.971	0.819	0.719	0.809
Worm	0.978	0.899	0.996	0.922	0.721	0.921
Average	**0.967**	**0.920**	**0.9907**	**0.848**	**0.704**	**0.865**

⟨**A5**⟩ Static features are more important than dynamic features for both classification tasks, but specially for family classification. Raw and opcode n-grams are the most important feature classes in both classification tasks. The importance of a feature class may depend on the classification task. For example, *d-registry* is important to distiguish malware from goodware, but is not relevant for family classification.

4.6 Hard-to-Detect Malware

In this section, we illustrate which malware classes and families pose a greater challenge for classifiers based on static and dynamic features. Table 5 shows Recall and F1-scores for each malware class in binary and family classification respectively. In binary classification, the recall value is defined as the number of correctly classified samples in the class over the total number of samples in the class. The numbers differ from those in Table 3 because Table 5 only considers the classification results of malware samples, while Table 3 covers the classification of both goodware and malware samples (thus taking also false positives into account).

As we can see, the recall and F1 score are not uniform across all classes and can widely vary depending on the task and the features used. Static features are considerably better at detecting downloaders, dialers, and worms. In contrast, dynamic features perform better on rogueware, miner and ransomware.

These results are confirmed also if we look at individual families. We provide in the appendix Table 9, Table 10, Table.11 and Table.12 to show the 10 families with the lowest accuracy in both classification tasks using static and dynamic features. For instance, among the 10 malware families for which the static classifier makes more mistakes, we count four *viruses* (i.e., file infectors) and six *grayware*. This is even more remarkable if we consider the fact that there are only 40 families of Viruses in our entire dataset. The fact that *viruses* typically append their code to benign files results in a wide variation in terms of static features among samples of the same family, and this can explain why it is hard for a static classifier to differentiate them from *goodware* and from other families. Similarly, *grayware* is defined as undesirable code, which is not outright malicious per se, therefore making it difficult to find a clear boundary to isolate these families. In the worst 10 families using dynamic features, we can observe a similar pattern: grayware and virus dominate the list.
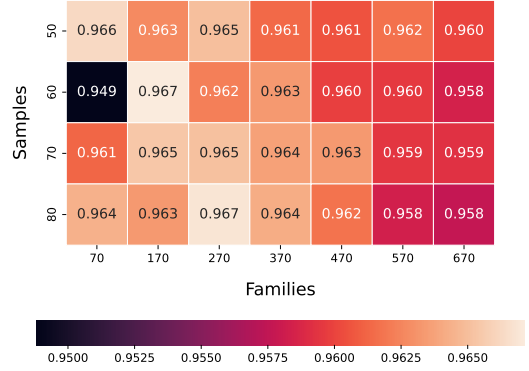
Figure 1: F1 score heatmap for binary classification using static model.

Besides, adware and spyware are also among the worst families. Malware samples in each of the class have similar behaviours.

⟨**A6**⟩ Models employing static features find more difficult to classify *grayware* and *viruses*. Dynamic features can identify ransomware, spyware and adware as malware, but they have great difficulty in properly identifying their families, probably due to very similar runtime behaviours of different families in these classes.

4.7 Impact of Training Dataset Construction

This section evaluates the effect of the construction of the training dataset on classification accuracy. We specifically investigate the impact of the size of the training dataset, the variety of malware families represented, and the uniformity of the sample-family selection. To the best of our knowledge, the question of how the diversity in terms of families impact binary classification has not been studied before.

To study this aspect we plot a number of heatmaps. In each experiment, as described in Section.3, we reserved randomly 20 samples in each family for validation (e.g., hyper-parameter tuning) and we choose p samples from the remaining 80 samples and use them for training and testing. To study the impact of number of available samples, we vary p from 50 to 80. To study instead the impact of the number of different families in the dataset, we progressively vary the number of families involved in both binary and family classification from 70 to 670. For each combination of number of families and number of samples per family, we conduct a 10-fold cross validation test and report the averaged F1 score in the corresponding cell of each heatmap.

Figure 1 and Figure 2 present heatmaps of the F1 score for binary classification, using static features and dynamic features respectively. Figure 3 shows the heatmap for the combined model, for brevity only showing the variation with the number of families. Figure 4, Figure 5, and Figure 6 are similar but for the family classification task.

Overall, the results indicate that as the number of samples per family increases, the classification accuracy also increases. The exception is for the binary classification using static features, where increasing the samples per family may cause a decrease in overall accuracy. For example, when using 50 samples for each of the 670 families the F1 score is 0.960, but when using 80 samples it slightly decreases to 0.958. However, the trend is different if we consider
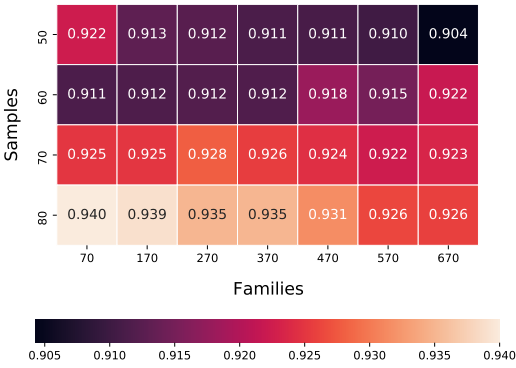
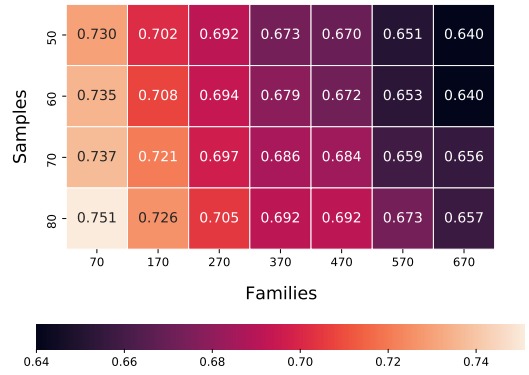**Figure 2: F1 score heatmap for binary classification using dynamic model.**

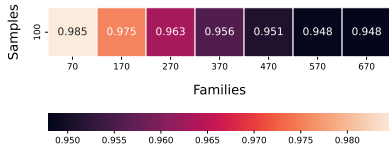**Figure 3: F1-score heatmap for binary classification with combined model.**

Figure 4: F1 score heatmap for family classification using Random Forest on static analysis features.

Figure 5: F1 score heatmap for family classification using Random Forest on dynamic analysis features

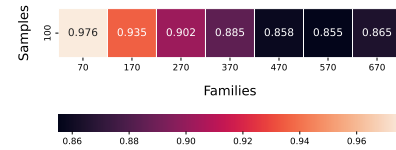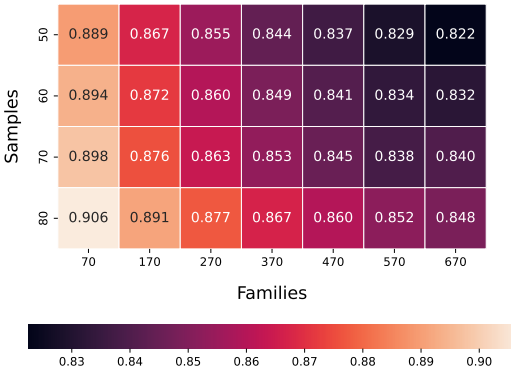**Figure 6: F1-score heatmap for family classification when combining features derived from static and dynamic analysis**

more families. We consider this very small changes as fluctuations due to the randomness of the sample selection process.

With respect to family diversity, the results confirm that the more families in the training dataset the more difficult their classification is. As expected, the decrease in classification accuracy is more marked for the family classification task, where intuitively the higher the number of classes the more difficult the classification becomes. The decrease is also more marked for the dynamic features than for the static ones, likely due to their lower discriminatory power as discussed in Section 4.1.

Non-uniform sampling. We also evaluate the impact of a non-uniform downsampling strategy for binary classification. For this purpose, we mimic the distribution of the recently-proposed MOTIF

dataset [22], which contain 3,095 PE malware samples from 454 families with an unbalanced distribution (e.g., the median is 3 samples per family and 29% of families have a single sample). We create a new dataset by applying the MOTIF distribution to M_B. This new MOTIF-like dataset comprises of 4,821 samples from all 670 families with the following distribution: 29% of families are singletons, 41% have 2-5 samples, 12% 6-10, 10% 11-20, 4% 21-30, 2% 31-40, 1% 41-50, and 1% has over 50 samples (up to 100).

We use this to compare two sampling approaches: the *uniform* approach (which is the one we adopted so far in the paper) where we keep a balanced number of samples for each family, versus a *nonUniform* approach, where we consider a real-world case in which the number of available samples varies from one family to another, as captured by the MOTIF-like dataset. Table 6 show the results for both approaches and different feature sets. We could not identify any significant difference between the two approaches, thus suggesting that training a classifier with a non-uniform amount of samples does not significantly impact its performance, under the important assumption that the testing dataset also follows the same distribution.

> ⟨**A7**⟩ Increasing the number of malware families in the training set makes the classification more complex and generally results in a lower accuracy. While not surprising, this is very important because previous studies were often performed on only few dozens of families, with the risk of reporting inflated results that do not generalize to larger and more realistic datasets.
> Increasing the number of samples per family can help to increase the classification accuracy, in particular for models based on dynamic analysis. Finally, the choice between a non-uniform and a uniform downsampling strategy does not significantly affect the binary classification accuracy.

Table 6: Impact of uniform and non-uniform sample selection in training dataset.

Model	Precision	Recall	F1-score	Accuracy
binary-static-uniform	0.956	0.957	0.957	0.957
binary-dynamic-uniform	0.962	0.892	0.926	0.929
binary-combined-uniform	0.963	0.934	0.948	0.948
binary-static-nonUniform	0.961	0.960	0.961	0.960
binary-dynamic-nonUniform	0.959	0.886	0.921	0.924
binary-combined-nonUniform	0.955	0.927	0.940	0.927

Table 7: Binary classification accuracy on singletons and unseen families datasets.

Model	Singletons	Unseen
binary-static-uniform	0.943	0.815
binary-dynamic-uniform	0.805	0.898
binary-combined-uniform	0.985	0.908
binary-static-nonuniform	0.810	0.653
binary-dynamic-nonuniform	0.328	0.855
binary-combined-nonuniform	0.758	0.637

4.8 Model Generalization
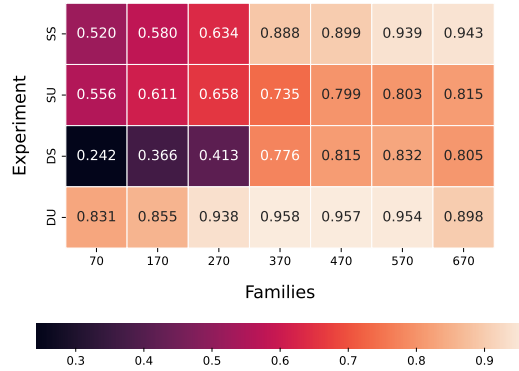
Figure 7: Binary classification accuracy on singletons and unseen families of the uniform dynamic and static models. (SS: Static Singleton. SU: Static Unseen. D is for Dynamic)

In this section, we test how well our models for binary and family classification generalize on unseen data. To this extent, we validate the performance of the previously-trained models on the singleton and unseen datasets introduced in Section 2.2, which include new families and have different distributions from the training data. This scenario is known as the "out-of-distribution" (OOD) test [29], where training and testing data have different distributions in the feature space. The distribution gap between the training and testing data has been frequently witnessed in malware analysis [21], as malware families evolve rapidly over time. Theoretically, one should expect the performance of a ML model to drop drastically in this more realistic scenario, as OOD samples directly violate the IID assumption of ML techniques.

Binary Classification. Table 7 summarizes the binary classification results over the singletons and unseen families using the static, dynamic, and joint feature pool. "Uniform" and "non-uniform" in the table denote training with the 670 families with uniformly and non-uniform dataset construction methods (§ 3.4) The empirical measurements shown in Table 7 can be summarized around three main observations.

First, the accuracy of binary classification using only static or dynamic features deteriorates significantly over singleton and unseen family files. Using the combined feature set, the binary classification accuracy with the uniform setting augments over the singleton samples, whereas it deteriorates over the unseen families. In the non-uniform setting, we can observe the same tendency of accuracy drop over the OOD samples. The observations echo closely to the out-of-distribution challenge of machine learning raised in [29].

Second, the accuracy deterioration over the out-of-distribution samples is more significant in the non-uniform setting of training than that in the uniform setting, regardless of the used features.

This is different from the results of the in-distribution evaluation in Table 6, where we observe no major difference in accuracy between the uniform and non-uniform settings. These results show an important point: classifiers built on very unbalanced datasets may perform equally well when tested on samples with the same unbalanced distribution, but generalize more poorly to other testing datasets, likely because many families were underrepresented in the training and thus the model failed to properly capture them.

Third, we can notice that static features generalize poorly to unseen families, while dynamic features perform better in this scenario. This is due to the nature of the features themselves: static information can precisely pinpoint only known samples, while dynamic behavior can better generalize also to unknown ones. Thus, compared to static features, dynamic features may provide more rich information to capture new types of malicious behaviors that never appear in the training phase.

We investigate this aspect in more detail by varying the number of families we used for training. In Figure 7, we can see that dynamic features perform poorly when the number of malware families for training is low (as there was not enough example of behaviors to learn from) but, with a sufficient number of families, they offer better classification results than static features. Dynamic features usually have a high dimensional and highly sparse feature representation. For example, some files or processes only appear a few times in the training set for specific malware families. A smaller number of families may aggravate the curse of dimensionality, which results in an overfitting of the classifier. Furthermore, we can observe the
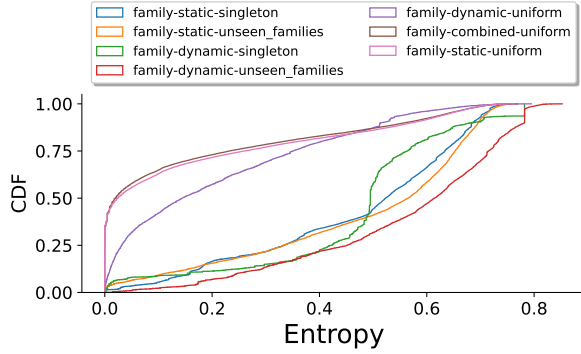
Figure 8: Entropy distribution comparison

Table 8: Related work on ML-based Detection and family Classification of Windows malware (S=Static, D=Dynamic)

		Goal		Features		Dataset	
Work	**Year**	**D**	**C**	**S**	**D**	**#**	**Fam.**
Rieck et al. [43]	2008	✗	✓	✗	✓	10K	14
McBoost [38]	2008	✓	✗	✓	✓*	5.5K	-
PE-Miner [50]	2009	✓	✗	✓	✗	16K	-
Nataraj et al. [35]	2011	✗	✓	✓	✓	67K	561*
OPEM [47]	2012	✓	✗	✓	✓	1K	-
Santos et al. [46]	2013	✓	✗	✓	✗	1K	-
Dahl et al. [14]	2013	✓	✗	✗	✓	1.8M	-
Kancherla et al. [23]	2013	✓	✗	✓	✗	25K	-
Saxxe et al. [48]	2015	✓	✗	✓	✗	350K	-
Miller et al. [33]	2016	✓	✗	✓	✓	1.1M	-
MtNet [18]	2016	✓	✓	✗	✓	2.8M	98
MAAR [45]	2017	✓	✗	✗	✓	3K	-
MalConv [41]	2018	✓	✗	✓	✗	284K	-
EMBER [8]	2018	✓	✗	✓	✗	400K	-
Rhode et al. [42]	2018	✓	✗	✗	✓	5.1K	-
MalDy [24]	2019	✓	✓	✗	✓	20K	15
NeurLux [20]	2019	✓	✗	✗	✓	34K	-
MalInsight [17]	2019	✓	✓	✓	✓	3.5K	5
MalDAE [16]	2019	✓	✗	✓	✓	5.5K	-
MALDC [58]	2020	✓	✗	✗	✓	54K	-
IMCFN [54]	2020	✓	✗	✓	✗	9.4K	-
Zhang et al. [59]	2020	✓	✗	✗	✓	27.7K	-
Rabadi et al. [40]	2020	✓	✗	✗	✓	7.1K	-
Joyce et al. [22]	2022	✗	✓	✓	✗	3K	454
This work	2023	✓	✓	✓	✓	67K	670

classification accuracy over unseen samples improves as the number of families increase, regardless of the features used in the test.

Family Classification. So far, we only tested the generalization of our models in a binary classification scenario. We now apply our family classifier trained using the 670 families over the singleton and unseen families as another out-of-distribution test scenario. Achieving high or low classification accuracy over these out-of-distribution samples is not interesting, as most of these samples share no common families as the training data and we don't have the ground truth family labels for these samples. Thus, the purpose of organizing this test is only to study how the uncertainty level of the family classifier changes over the out-of-distribution malware samples.

To measure the uncertainty difference, we define the Relative Entropy Score (*RES*) of the classifier's output as $\frac{\sum_{k=1}^{C} p_k \log p_k}{T}$, where $T = \sum_{k=1}^{C} 1/C \log 1/C$ and C is the number of the families covered by the training data building the classifier. In this experiment, C is therefore set to 670. For an input sample, the output of the family classifier is a 670-dimensional probability-valued vector $\{p_k\}$ (k=1,2,3,...,C=670). Each p_k gives the probabilistic confidence that the sample belongs to the corresponding family. By definition, the numerator $\sum_{k=1}^{C} p_k \log p_k$ provides the entropy of the classifier's output. The denominator $\sum_{k=1}^{C} 1/C \log 1/C$ denotes the maximum entropy that the classifier's classification output may have. As a result, the magnitude of *RES* is strictly normalized between 0 and 1. Higher/Lower *RES* denotes that the classifier shows higher/lower uncertainty level over the classification output.

In Figure 8, we demonstrate the empirical cumulative distribution function (CDF) of RES-based uncertainty distribution of the family classifier's output on the testing malware samples of the 670 families (in-distribution samples) and those belonging to singleton / unseen families. Consistently with theoretical studies [29], we can find that the uncertainty level of the family classification output over the singleton and malware samples of previously unseen families increases significantly, compared to those derived with the testing samples sharing the same families of the training data.

⟨**A8**⟩ Our experiments confirm a significant performance drop in binary classification over out-of-distribution samples, both in the case of singleton and unseen families. At the same time, the confidence of the ML-based classifier decreases significantly over these out-of-distribution samples. This implies that ML-based models tend to be less certain over malware samples drifted from the training samples. Our results also show that models trained on a very unbalanced dataset generalize more poorly, and that dynamic features generalize better than static over new families. Overall, as the distribution gap between training and testing malware samples is common in practice, these results raise concern over the utility of ML-based malware classification for real-world scenarios.

5 RELATED WORK

Table 8 presents a categorization of previous works on Windows malware classification, according to their goal (binary detection or family classification), features (static or dynamic), and dataset size (both in terms of malware executables and malware families). Among the approaches in Table 8, the choice of the models varies widely including classical models like Support Vector Machine, GradientBoost and Random Forest, as well as neural networks. Most approaches perform feature extraction, e.g. extract n-grams of bytes, opcodes, or system calls, but a couple work directly operate on raw bytes and API sequences [20, 41].

MalInsight [17] is the only study so far to provide a comprehensive coverage over the choice of features and classification tasks. However, their dataset includes only 5 families. At the other end of the spectrum, Nataraj et al. [35] studied only family classification on a unbalanced dataset with over 500 classes. However, the authors consider a each full AV label a different class, so that number does not correspond to real malware families. In contrast, our study investigates the factors impacting the performance of ML classifiers using a large-scale balanced dataset with 670 families.

ML challenges and pitfalls. In cyber security research, two major challenges are raised in the practices of ML-based analysis. First of

all, the issue of missing observations affecting the prediction accuracy, e.g., in network intruion detection [36, 52]. Secondly, most of ML models follow a core assumption: the training and test data of a ML model should be drawn identically and independently from the same underlying distribution, i.e. the I.I.D. assumption. However, the I.I.D assumption does not hold in practices. Highly diversified and quickly evolving malware technologies make the implementations and behaviours of malware vary significantly and frequently. New variants of malware arise to exploits novel vulnerabilities and evade detection of anti-virus services. Once a machine-learning-driven malware classifier is deployed in practical security applications, the fast changing profiles of malware samples breaks the I.I.D, assumption and causes the deterioration of the classification accuracy [11]. The design of a robust classifier for frequently drifting malware profiles is still an open problem.

Arp et al. [10] review the use ML-based classification in cyber security published over the past 10 years. The study summarizes the existing issues at the different stage of the ML-based pipelines for cyber security data analysis . For example, the authors demonstrate that the statistical bias introduced by training sample sampling and inaccurate class label tagging may introduce spurious correlation into the ML classifiers. In addition, employing inappropriate performance metrics ignoring the class imbalance in the testing phase may lead to incorrect interpretation to the quality of ML-based predicative analysis. In general, according to [10], the performance metrics of a ML-based analysis pipeline in cyber security practices should be defined by considering the characteristics of the security data collected and the requirements raised in the concerned applications. Otherwise, the pipelines may produce unrealistic performance and interpretations to security incidents. In our work we focus instead on the bottlenecks of ML-based malware classification encountered in practices, which may obstruct accurate classification of malware. For instance, we focus on the impact of the coverage of malware families for training and we dive into the potential reasons causing failure of ML-based models over certain malware samples. We also explore how the classifier behave over out-of-distribution malware samples, which is an interesting problem in the practical deployment of ML-driven pipelines.

Dataset construction. In 2015, the Microsoft Malware Classification Challenge [51] was developed as a Kaggle competition to conduct malware family classification. The corresponding dataset is composed of disassembly and bytes of 20K Windows malware samples from 9 families. It was released in the Kaggle competition and has since been used in several studies. [27, 37] built larger-scale Android malware datasets for evaluating the performances of ML-driven classification models. More specifically, [37] evaluates the spatial and temporal bias of binary classification accuracy over 129,728 Android apps. [27] explores the variance-bias trade-off of malware clustering on 134,698 Android apps. By comparison, our work focuses on the measurement study over large-scale Windows malware collections. Our goal is to characterize the applicability and limits of ML-driven malware classifiers for practical use. In [22], Joyce et al. built a multi-family dataset containing 3,095 malware samples collected from from 454 families. This work offers the most diversified coverage over different malware families in public malware datasets with manually verified labels. Interestingly, this dataset has a highly skewed distribution over the number of malware samples per family. Over half of the families contain less than 5 samples per family, which poses a few-shot learning challenge to ML-driven malware classification. In our study tried to we mimic this distribution to assess the impact of the skewed distribution of malware samples over the accuracy of the trained ML-based classifiers. We also compare the impact of the skewed distribution and that of varying malware coverage regarding the classification accuracy. The empirical study helps identify the limits of ML-based classification methods in practical malware analysis.

6 FINAL RECOMMENDATIONS

The goal of this work was to understand the key factors that influence the performance of machine-learning models for malware detection and classification. Based on the results of the individual experiments that we conducted, we can draw some general recommendations on the use of ML for malware classification:

1. Experiments on malware classification (both binary and family) should be performed on hundreds of different families. Classifiers trained on a few families (like the ones on the popular Microsoft dataset) can provide misleadingly high accuracy scores.
2. Each family in the dataset should contain a sufficient number of samples. In particular, results from experiment performed on very unbalanced datasets (e.g., where many families only contain a handful of samples) tend to generalize poorly when tested over different distributions.
3. Static features dominate detection and classification of samples from *known* families, by relying on signature-like information extracted from sequences of bytes and opcodes. Packing, in its current widespread implementation, does not seem to have a considerable negative effect on this. The addition of dynamic features, which are much more time-consuming and error-prone to extract, has only a marginal impact on the classification accuracy and therefore its use should be carefully considered if the goal is to detect known families. However, static features are unable to capture samples from *unknown* families, where instead models based on dynamic behavior show a better ability to generalize.
4. The performance of *all* ML models drop drastically when tested on OOD samples. Therefore, the completeness and the regular update of the training data (which are both difficult to achieve in the real world) are key to obtain good results. ML models, especially if integrating dynamic features, might still be used to flag suspicious previously-unseen samples, but with much less accuracy and higher false positive rates.

REFERENCES

[1] Accessed January 20, 2023. Chocolatey, the Package Manager for Windows. https://chocolatey.org/. (Accessed January 20, 2023).

[2] Accessed January 20, 2023. Detect-It-Easy. https://github.com/horsicq/Detect-It-Easy. (Accessed January 20, 2023).

[3] Accessed January 20, 2023. JuanLesPIN. https://github.com/Maff1t/JuanLesPIN-Public. (Accessed January 20, 2023).

[4] Accessed January 20, 2023. LordNoteworthy/al-khaser. https://github.com/LordNoteworthy/al-khaser. (Accessed January 20, 2023).

[5] Accessed January 20, 2023. Proxmox Virtual Environment. https://www.proxmox.com/en/proxmox-ve. (Accessed January 20, 2023).

[6] Accessed January 20, 2023. Yara patterns of RetDec. https://github.com/avast/retdec/tree/master/support/yara_patterns. (Accessed January 20, 2023).

[7] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin'Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In *Network and Distributed Systems Security (NDSS) Symposium 2020.*

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

[9] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. 2022. Humans vs. Machines in Malware Classification. In *To appear in Usenix Security 2023.*

[10] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp

[11] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift. In *IEEE Symposium on Security and Privacy (Oakland).* https://arxiv.org/abs/2010.03856

[12] Capstone. 2022. Capstone - The ultimate disassembly framework. (2022). https://www.capstone-engine.org/.

[13] Microsoft Corporation. 2022. PE Format. (2022). https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

[14] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. 2013. Large-Scale Malware Classification using Random Projections and Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing.*

[15] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security* 113 (2022), 102550.

[16] Weijie Han, Jingfeng Xue, Yong Wang, Lu Huang, Zixiao Kong, and Limin Mao. 2019. MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *computers & security* 83 (2019), 208–233.

[17] Weijie Han, Jingfeng Xue, Yong Wang, Zhenyan Liu, and Zixiao Kong. 2019. MalInsight: A systematic profiling based malware detection framework. *Journal of Network and Computer Applications* 125 (2019), 236–250.

[18] Wenyi Huang and Jack W. Stokes. 2016. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment.*

[19] AV-TEST Institute. 2023. New Malware. (2023). https://www.av-test.org/en/statistics/malware/.

[20] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. 2019. Neurlux: Dynamic Malware Analysis without Feature Engineering. In *Annual Computer Security Applications Conference.*

[21] Roberto Jordaney, Kumar Sharad, Santanu Kumar Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17).* USENIX Association, USA, 625–642.

[22] Robert J Joyce, Dev Amlani, Charles Nicholas, and Edward Raff. 2022. MOTIF: A Large Malware Reference Dataset with Ground Truth Family Labels. In *Workshop on Artificial Intelligence for Cyber Security.*

[23] Kesav Kancherla and Srinivas Mukkamala. 2013. Image visualization based malware detection. In *IEEE Symposium on Computational Intelligence in Cyber Security.*

[24] ElMouatez Billah Karbab and Mourad Debbabi. 2019. MalDy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. *Digital Investigation* 28 (2019), S77–S87.

[25] Kaspersky. 2023. PCybercriminals attack users with 400,000 new malicious files daily. (2023). https://www.kaspersky.com/about/press-releases/2022_cybercriminals-attack-users-with-400000-new-malicious-files-daily---that-is-5-more-than-in-2021.

[26] Alexander Kuechler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In *Network and Distributed System Security (NDSS) Symposium (NDSS 21).*

[27] Shinho Lee, Wookhyun Jung, Wonrak Lee, Hyung Geun Oh, and Eui Tak Kim. 2021. Android malware dataset construction methodology to minimize

[28] bias–variance tradeoff. *ICT Express* (2021).

[28] Chia Chin Lip and Dzati Athiar Ramli. 2012. *Comparative Study on Feature, Score and Decision Level Fusion Schemes for Robust Multibiometric Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 941–948. https://doi.org/10.1007/978-3-642-27552-4_123

[29] Weitang Liu, Xiaoyun Wang, John D. Owens, and Yixuan Li. 2020. Energy-Based out-of-Distribution Detection. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20).* Curran Associates Inc., Red Hook, NY, USA, Article 1802, 12 pages.

[30] Nicola Loi, Claudio Borile, and Daniele Ucci. 2021. Towards an Automated Pipeline for Detecting and Classifying Malware through Machine Learning. (2021). https://doi.org/10.48550/ARXIV.2106.05625

[31] Robert Lyda and James Hamrock. 2007. Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy* 5, 2 (2007), 40–45.

[32] Lorenzo Maffia, Dario Nisi, Platon Kotzias, Giovanni Lagorio, Simone Aonzo, and Davide Balzarotti. 2021. Longitudinal Study of the Prevalence of Malware Evasive Techniques. *arXiv preprint arXiv:2112.11289* (2021).

[33] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, Anthony D. Joseph, and J. D. Tygar. 2016. Reviewer Integration and Performance Measurement for Malware Detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment.*

[34] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. 2017. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 1009–1024.

[35] Lakshmanan Nataraj, Vinod Yegneswaran, Phillip Porras, and Jian Zhang. 2011. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence.*

[36] Marek Pawlicki, Michał Choraś, Rafał Kozik, and Witold Hołubowicz. 2021. Missing and Incomplete Data Handling in Cybersecurity Applications. In *Intelligent Information and Database Systems*, Ngoc Thanh Nguyen, Suphamit Chittayasothorn, Dusit Niyato, and Bogdan Trawiński (Eds.). Springer International Publishing, Cham, 413–426.

[37] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *USENIX Security Symposium.*

[38] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. 2008. McBoost: Boosting Scalability in Malware Collection and Analysis using Statistical Classification of Executables. In *Annual Computer Security Applications Conference.*

[39] Marco Pontello. 2021. TrID - File Identifier. (2021). http://mark0.net/soft-trid-e.html.

[40] Dima Rabadi and Sin G Teo. 2020. Advanced windows methods on malware detection and classification. In *Annual Computer Security Applications Conference.*

[41] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware Detection by Eating a Whole EXE. In *Workshops at the AAAI Conference on Artificial Intelligence.*

[42] Matilda Rhode, Pete Burnap, and Kevin Jones. 2018. Early-stage malware prediction using recurrent neural networks. *computers & security* 77 (2018), 578–594.

[43] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. 2008. Learning and Classification of Malware Behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment.*

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

[45] Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. 2017. MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values. *Engineering Applications of Artificial Intelligence* 59 (2017), 93–102.

[46] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences* 231 (2013), 64–82.

[47] Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. 2012. OPEM: A Static-Dynamic Approach for Machine-learning-based Malware Detection. In *International joint conference CISIS'12-ICEUTE´ 12-SOCO´ 12 special sessions.*

[48] Joshua Saxe and Konstantin Berlin. 2015. Deep Neural Network Based Malware Detection using two Dimensional Binary Program Features. In *International Conference on Malicious and Unwanted Software.*

[49] Silvia Sebastián and Juan Caballero. 2020. AVclass2: Massive Malware Tag Extraction from AV Labels. In *Annual Computer Security Applications Conference.* 42–53.

[50] M Zubair Shafiq, S Momina Tabish, Fauzan Mirza, and Muddassar Farooq. 2009. Pe-miner: Mining structural information to detect malicious executables in realtime. In *International workshop on recent advances in intrusion detection.* Springer, 121–141.

[51] Michael R Smith, Nicholas T Johnson, Joe B Ingram, Armida J Carbajal, Bridget I Haus, Eva Domschot, Ramyaa Ramyaa, Christopher C Lamb, Stephen J Verzi, and W Philip Kegelmeyer. 2020. Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. 49–60.

[52] Nazgol Tavabi, Andres Abeliuk, Negar Mokhberian, Jeremy Abramson, and Kristina Lerman. 2020. Challenges in Forecasting Malicious Events from Incomplete Data. In *Companion Proceedings of the Web Conference 2020 (WWW '20)*. Association for Computing Machinery, New York, NY, USA, 603–610. https://doi.org/10.1145/3366424.3385774

[53] G. V. Trunk. 1979. A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1, 3 (1979), 306–307. https://doi.org/10.1109/TPAMI.1979.4766926

[54] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. 2020. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks* 171 (2020), 107138.

[55] vtfeedapi Accessed January 20, 2023. VirusTotal API 2.0 Reference: File Feed. https://developers.virustotal.com/v2.0/reference/file-feed. (Accessed January 20, 2023).

[56] jiezhong xiao, qian han, and yumeng gao. 2022. Hybrid Classification and Clustering Algorithm on Recent Android Malware Detection. In *2021 5th International Conference on Computer Science and Artificial Intelligence (CSAI 2021)*. Association for Computing Machinery, New York, NY, USA, 249–255. https://doi.org/10.1145/3507548.3507586

[57] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M Blough, Elissa M Redmiles, and Mustaque Ahamad. 2021. An Inside Look into the Practice of Malware Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3053–3069.

[58] Hao Zhang, Wenjun Zhang, Zhihan Lv, Arun Kumar Sangaiah, Tao Huang, and Naveen Chilamkurti. 2020. MALDC: a depth detection method for malware based on behavior chains. *World Wide Web* 23, 2 (2020), 991–1010.

[59] Zhaoqi Zhang, Panpan Qi, and Wei Wang. 2020. Dynamic malware analysis with feature engineering and feature learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

APPENDIX

6.1 Impact of Missing Dynamic Feature Values

Figure 9 compares the family-wise classification accuracy (F1 score) for a family with its FMR for the family classification task. The figure shows that a lower FMR tends to produce higher F1 scores and vice versa.
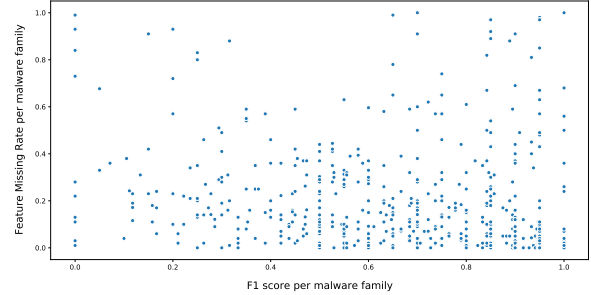

Figure 9: F1 score for family classification using dynamic features versus Feature Missing Rate (FMR) for the family.

6.2 Hard-to-Detect Malware Families

Tables 9 and 10 list the 10 malware families that have the lowest binary and family classification accuracy when only static features are used. Tables 11 and 12 show the 10 malware families with the lowest binary and family classification accuracy using only dynamic features.

Table 9: Top-10 malware families with the lowest binary classification accuracy using the static features (i.e., highest mispredictions as goodware).

| | Static binary classification | | |
Family	Class	Avg Recall	% packed
pioneer	virus	0.401	6%
asparnet	grayware	0.410	5%
systweak	grayware	0.458	19%
shopper	grayware	0.500	1%
sality	virus	0.516	4%
vitro	virus	0.553	3%
installcore	grayware	0.596	10%
slugin	virus	0.598	4%
elex	adware	0.603	9%
passview	grayware	0.617	35%

Table 10: Top-10 families with the lowest family classification accuracy using static features (i.e., highest mispredictions to other families)

Static family classification			
Family	Class	Avg F1	% packed
zpevdo	grayware	0.150	15%
vitro	virus	0.240	3%
uwamson	grayware	0.252	15%
gendal	grayware	0.280	62%
dumpex	grayware	0.290	40%
alman	virus	0.293	11%
sality	virus	0.328	4%
pasta	grayware	0.346	28%
cobra	grayware	0.381	60%
copidmbe	virus	0.387	9%

Table 11: Top-10 malware families with the lowest binary classification accuracy using dynamic features (i.e., highest mispredictions as goodware).

Dynamic binary classification				
Family	Class	Avg Recall	Packed	FMR
tasker	grayware	0.0	11%	0.77
malex	downloader	0.0	1%	0.77
rostpay	grayware	0.0	96%	0.76
constructor	grayware	0.0	13%	0.78
atcpa	virus	0.0	0%	0.78
mocrt	spyware	0.0	73%	0.80
mokes	backdoor	0.0	1%	0.65
bingoml	grayware	0.0	22%	0.72
safebytes	grayware	0.0	99%	0.81
trymedia	adware	0.0	73%	0.70

Table 12: Top-10 families with the lowest family classification accuracy using dynamic features (i.e., highest mispredictions to other families)

Dynamic family classification				
Family	Class	Avg F1	% packed	FMR
bancos	spyware	0.0	44%	0.76
kovter	grayware	0.0	0%	0.78
safebytes	grayware	0.0	99%	0.80
winner	grayware	0.0	0%	0.80
umbra	downloader	0.0	0%	0.80
ulise	grayware	0.0	2%	0.80
contenedor	virus	0.0	0%	0.80
cobra	grayware	0.0	60%	0.79
kuaizip	adware	0.0	1%	0.80
zpevdo	grayware	0.0	15%	0.77