Simplicity Performs, But Should It? Evaluating Malware Detection Benchmarks

Samy Bettaieb[1], Laurens D'hooge[2], Charles-Henry Bertrand Van Ouytsel[1], Axel Legay, Etienne Rivière[1], Miel Verkerken[2], and Bruno Volckaert[2]

[1] UCLouvain
{samy.bettaieb, charles-henry.bertrand, etienne.riviere}@uclouvain.be
[2] UGent, IDLab-imec
{laurens.dhooge, miel.verkerken, bruno.volckaert}@ugent.be

Abstract. Deep Learning (DL) has become a dominant approach in malware detection, often favored for its ability to capture complex patterns. However, this study highlights a critical issue: simple non-deep learning models, such as One-Rule and AdaBoost, perform surprisingly well on several state-of-the-art malware detection datasets, achieving results comparable to those of more complex models, including LightGBM and DL approaches like FastAI's Tabular Learner. Rather than suggesting that simple models are inherently superior, our findings indicate that a small subset of features often dominates the classification process, allowing models to achieve high accuracy with only a small fraction of the feature space. This behavior appears to be closely tied to the limited variability in some feature values across academic datasets. Across the eleven datasets that we analyze, classification scores reach above 99% and the performance gap between simple and complex models is only 5.6% in absolute terms and 6% in relative terms. We provide evidence that complex models are often overvalued in academic malware detection benchmarks and demonstrate the value of simple baselines such as the 1R model and AdaBoost. We also present examples of dominant features that bias classification across datasets, with an in-depth analysis of EMBER features and their impact on classification. Finally, we release cleaned, reusable datasets to support further research.

Keywords: Machine Learning, Malware Analysis, Baselines, Data Quality

1 Introduction

Machine learning (ML) has become a cornerstone of malware detection, providing an efficient approach to analyzing vast amounts of binaries and distinguishing between malicious and benign software. Traditional signature-based methods, while effective against known threats, struggle to detect new malware variants, making ML-based approaches popular for identifying previously unseen attacks. Among ML approaches, deep learning (DL) has gained considerable attention, mainly due to its success in image recognition and natural language processing, where capturing complex patterns is key. Techniques using assembly code

embeddings, pre-trained language models, and even combinations of multiple Graph Neural Networks architectures are now frequently used to represent malware samples [6, 15, 30, 31, 34]. This growing trend of applying deep learning models in malware detection research is often based on the assumption that DL's complexity will yield superior results.

Additionally, the size and quality of training data are crucial factors in model performance; however, academic malware datasets may not always accurately reflect real-world diversity. Unlike image data, where individual pixels do not hold semantic content, malware data can often be represented with features that carry specific semantic meaning (e.g., file size, API calls, byte distributions). Sets of those features end up together to create tabular dataset formats. In these cases, simpler models like decision trees can effectively use this structured information, and intricate DL architectures may not always provide significant improvements over simpler, more interpretable models.

At the same time, evaluation reproducibility remains a significant challenge in malware detection research. Unlike image classification, which benefits from standardized, public benchmark datasets such as CIFAR-10, ImageNet, or MNIST [10, 11, 29], the rapidly evolving nature of malware samples introduces variability, making it difficult to maintain reliable datasets over time. It is a common practice among researchers to rely on curated or adapted academic datasets to validate their detectors. However, such datasets may not reflect the variability observed in real-world malware or may not be sufficiently documented or even available. On top of that, if the chosen feature representation relies on access to the binary files, researchers may face legal or practical restrictions that prevent them from sharing cleanware binaries. This often results in either incomplete or entirely unavailable datasets, further worsening the reproducibility issue.

This study aims to address two key research questions (RQ) while providing transparent and accessible resources for malware detection research.

- RQ1: How do simple models perform on public academic malware datasets?
- RQ2: Do certain features overly influence the classification process, potentially biasing classification results?

To answer these questions, we evaluate simple models such as One Rule and AdaBoost against more complex models such as LightGBM (a gradient boosting model) and FastAI's Tabular Learner (a DL model) on several state-of-the-art malware datasets. Our findings show that simple models can perform similarly to more complex models. This suggests that the baseline classification performance achieved by simple models is already quite high, resulting in only minor improvements in performance for more complex models. We interpret this not as evidence of the sufficiency of simple models, but as an indication that some public academic datasets can be solved with limited features that dominate decision-making. This raises concerns about the robustness of evaluations based on current benchmarks and the generalization beyond them to the real world.

This work encourages a closer examination of the role of model complexity in this field. It addresses crucial questions about whether academic datasets may lack real-world variability or introduce biases that overstate model performance.

We hope this study encourages the community to look at model behavior and dataset construction more closely, aiming for malware detection benchmarks that better reflect real-world conditions, practicality, and reproducibility, rather than excessive architectural complexity.

This study makes the following contributions to malware detection research:

- **Comprehensive model performance evaluation**: We assess the performance of simple models (One Rule [19], AdaBoost [16, 17]) alongside more complex ones (LightGBM [26], FastAI's Tabular Learner [14, 20]) on eleven academic malware datasets (Section 4).
- **EMBER features mapping and analysis**: We provide a detailed mapping and critique of EMBER features [3], analyzing their influence on model performance and identifying correlations that could distort classification results (Section 5).
- **Reproducible experiments in data and implementation**: All datasets used in our experiments are cleaned, preprocessed, and publicly provided on Kaggle in a standardized format. All experimental code resides on the same platform alongside the associated datasets. Pinned execution environments further strengthen the reproducibility and longevity of our experiments.

2 Background & Related work

This section provides an overview of the application of machine learning for malware detection. We begin by discussing standard features extracted from binaries, particularly the EMBER (version 2) features. We then provide an overview of common challenges and pitfalls identified in recent literature that hinder the deployment of machine-learning approaches. Finally, we describe datasets commonly used in the literature.

2.1 Feature extraction

Cybersecurity researchers generally distinguish static features (extracted without executing the binary) and dynamic features (extracted from an execution). Dambra *et al.* [9] conducted a comprehensive study on machine learning in malware classification using one of the largest balanced malware datasets to date. Their findings reveal that static features outperform dynamic features and that combining both only marginally improves performance over static features alone. While dynamic features could better express the semantic characteristics of the analyzed samples (file creation, modified registry keys, ...), they generally result in sparse representations due to the high variability among malware families. Additionally, they also show that models trained on datasets with balanced class distributions generalize better to unseen data.

The EMBER features [3] are a set of static features commonly used in the literature [23–25, 33]. The authors of the EMBER dataset provide scripts to extract raw features from binaries, including a vectorization step for the considerable amount of textual (string) data, resulting in 2,381 features per malware

sample. These features are divided into nine categories: Byte Histograms, Byte Entropy Histograms, String Counts, General File Info, Header File Info, Section Info, Import Info, Export Info, and Data Directories Info. We detail the list of different features in Appendix A. While histograms and string count features are format-agnostic, other features are extracted by parsing PE files using the LIEF library. For the string vectorization, the authors of EMBER use the hashing trick [40] strategy to encode the full-text collection into a finite number of buckets. This operation was applied to multiple string-type descriptors in the header information, imported function names, exported function names, entry point section names, and more.

2.2 Pitfalls in machine learning for malware detection

Smith *et al.* [38] discuss the gap between the ML and infosec communities. They observe that malware classifiers often achieve near-perfect (99.9%) accuracy, raising concerns about overfitting and poor real-world generalization. To address this, they advocate for improving feature representation by prioritizing semantic over syntactic information.

Arp *et al.* [4] review existing work on machine learning applied to computer security and highlight common pitfalls that hinder their evaluation. In particular, they highlight the need to establish baseline performance with simple models to expose unnecessary complex models. They also recommend using explanation techniques to inspect model decisions and avoid spurious correlations. D'hooge *et al.* [12] demonstrate that in network intrusion detection (NIDS) research, increasing model complexity has taken hold without providing substantial benefits in classification scores. The lack of variability in the 17 analyzed state-of-the-art NIDS datasets lets elementary models perform equally well, fully eroding the claimed benefits of complicated models.

Shwartz *et al.* [36] compare DL models and traditional tree-based methods on tabular data. Their study reveals that XGBoost consistently outperforms DL approaches regarding generalization and hyperparameter tuning efficiency. They challenge the notion that deep learning is always better by showing the relevance of classical models in tabular data applications and highlighting the importance of considering performance, computational costs, and optimization time when selecting models for real-world data science problems.

2.3 Datasets

Our analysis incorporates eleven datasets, each contributing to a comprehensive evaluation of malware detection performance. The dataset's characteristics are presented in Table 1.

EMBER [3] contains PE files scanned by VirusTotal in 2017 and serves as a benchmark for malware detection. It is split into train/test sets based on observation time. An updated 2018 version focused on .NET, packed samples, and removed duplicates, while keeping the same split strategy. EMBER is widely used for learning, benchmarking, and adversarial attack research.

Name	Year	#Samples	C	Features	Focus
EMBER [3]	2017	1,100,000	✓	E	Detection
EMBER [3]	2018	1,000,000	✓	E	Detection
Packware [2]	2020	55,327	✓	E (+ 68,325)	Influence of packing
BODMAS [41]	2021	134,435	✓	E	Temporal aspects
ERMDS [23]	2023	15,300	✓	E	Robustness against obfuscation
MOTIF [25]	2023	3095	✗	E	Large corpuses
BenchMFC [24]	2024	223,000	✗	E	Classification under concept drift
CCCS-CIC-AndMal [27,35]	2020	357,805	✓	9,504	Android
Kronodroid [18]	2021	63,681	✓	489	Concept drift & Android
CIC-MalMem [18]	2022	58,058	✓	58	Memory dumps across categories
CIC-Evasive-PDFMal [21]	2022	10,023	✓	32	Malicious PDF documents

Table 1. Presentation of the datasets. Presence of cleanware (C), Ember features (E)

Packware [2] targets static detection on packed binaries, offering "wild" samples (from the wild) and "lab" variants (packed in controlled settings). It includes 70,706 features, among them the EMBER set. In this work, we focus on the wild subset, as our goal is malware detection rather than packer classification.

BODMAS [41] includes malware and cleanware samples collected randomly between August 2019 and September 2020, covering 581 malware families across 14 categories. EMBER features are provided and used in our analysis.

ERMDS [23] evaluates the robustness of ML-based detection systems against obfuscation. It also uses EMBER features, enabling comparisons with other datasets. It contains 10,000 malware samples (VirusShare, 2022) and cleanware from GitHub and SourceForge. We focus on BOS (binary-obfuscated) and POS (packed) subsets.

MOTIF [25] offers 3,095 disarmed malware samples from 454 families, enriched with high-confidence family labels derived from threat reports (20162021). Although it contains no cleanware, its valuable for studying malware classification and family evolution.

BenchMFC [24] is a benchmark for malware classification under concept drift. It includes 223,000 malware samples from 526 families (2012-2022), derived from 9.7 million VirusShare samples. EMBER features are utilized, and the dataset focuses on malware evolution over time.

To enable detection tasks with **BenchMFC** and **MOTIF**, we add 15,135 cleanware samples: 983 from Portable Freeware [39] and 14,152 from Chocolatey [1]. These two sources are commonly utilized in the literature [5,9]. After extraction, EMBER features are computed for all cleanware.

CCCS-CIC-AndMal-2020 [27,28,35] is an Android dataset from CIC with 195,624 malware samples (147 families, 12 categories) and 162,181 benign samples, described by 9,504 static features.

Kronodroid-2021 focuses on concept drift in Android malware detection. We use the real-device subset, combining 289 dynamic and 200 static features. After cleaning, the dataset includes 28,511 malware and 35,170 benign samples.

CIC-MalMem2022 [7,8] is built for obfuscated malware detection using memory dumps. It includes 2,916 malware samples (executed in VMs) and simulated benign behavior, yielding 58,058 memory records described by 58 features.

CIC-Evasive-PDFMal2022 [21,22] provides 5,555 malicious and 4,468 benign PDF files, all using 32 features, with a focus on evasion techniques.

3 Methodology

In this section, we introduce the four machine learning algorithms used in our comparison. Next, we describe the dataset cleaning and sampling process, followed by important background on our evaluation methodology.

3.1 Machine Learning Algorithms

The *One Rule* [19] model (1R) is the simplest ML model of the four used in this study. It generates a single decision rule for one feature based on an optimal numerical comparison that leads to the lowest error rate. While extremely straightforward, it is a valuable baseline for understanding how well a single feature can predict the target class.

AdaBoost [16,17] is a meta-estimator that combines multiple weak learners to create a strong classifier. It starts by training an initial classifier on the dataset, then sequentially adds new classifiers, each focusing more on the instances misclassified by previous iterations. The default classifier inside the Adaboost model is the 1R model.

LightGBM [26] is an efficient gradient-boosting framework that uses tree-based learning algorithms. Gradient-boosted decision tree (GBDT) models are widely used in malware classification research using tabular features [9,42]. Notably, the state-of-the-art performance on EMBER features is achieved by a GBDT model using LightGBM with its default parameters [3].

FastAI's Tabular Learner [14,20] is a deep learning model explicitly designed for tabular data. It uses neural networks with embedding layers for categorical variables and fully connected layers.

In terms of complexity, we consider One Rule and AdaBoost as simple models due to their straightforward nature: One Rule relies on a single feature, while AdaBoost sequentially enhances weak classifiers. In contrast, we classify LightGBM and FastAI's Tabular Learner as complex models. While LightGBM is a gradient-boosting decision tree model, and the Tabular Learner is a relatively small neural network, they still represent a significant increase in complexity compared to One Rule or Adaboost.

3.2 Dataset Cleaning and Sampling

Below, we outline the process we followed to clean and ready the datasets for training and evaluation:

- **Data Loading:** Load the dataset (e.g., from CSV or JSON files) and ensure column names and order match the provided documentation.
- **Handle Missing Values:** Identify rows with missing values (NaN). If no logical fill-in value exists, remove the incomplete samples.

- **Remove Duplicates:** Identify and remove duplicate rows to prevent inflated training and test scores.
- **Metadata:** Detect and remove metadata columns (e.g., source, sha256) that could lead to shortcut learning and artificially high performance.
- **Optimize Data Types:** Downsize feature data types whenever possible (e.g., float64 → float32, int64 → int8) to save memory and improve efficiency.
- **Store in Optimized Format:** Save the cleaned dataset in Apache Parquet, a binary format for tabular data with typed columns, which reduces storage size by approximately 90% and dramatically improves loading speed compared to text-based formats like CSV.

To ensure reproducibility, all datasets with their cleaning and classification notebooks that contributed to the results in this article are available on Kaggle. A repository which showcases our blueprint analysis applied on the EMBER2017 dataset is available at this link[3]. The code for the other datasets are on the same profile.

3.3 Additional Methodological Notes

The analyses include a few notable deviations from standard ML practice that should be mentioned.

Data Splits The train-test split is 20% train, 80% test instead of the usual 80-20, unless the dataset had pre-supplied train-test splits, in which case those were use. This split will show that even with these heavily reduced training sets, performance on the substantially larger test sets suffers virtually no penalty in classification scores.

One Rule Ensemble This article's baseline model of choice is the 1R model. We calculate the classification performance of a single, optimal comparison for every feature. To construct one classifier out of them, a simple ensembling procedure is added whereby singular 1R models with better-than-random classification ability (AUROC >0.5) get to make the predictions for the test set. Those predictions (0 or 1) are summed up and divided by the number of contributing 1R models, resulting in a single prediction value between 0.0 and 1.0. To make the final binary prediction, an optimal cut-off point is determined by letting the ensemble predict the training set and applying Youden's J statistic to set an optimal decision threshold. This threshold is re-used as the cut-off point for the final binary classification of the averaged test set predictions.

Tabular Learner The Tabular Learner uses learning rate scheduling for faster convergence according to the one-cycle policy described in the work by Smith and Topin [37]. This optimization lets us use at most 10 epochs for the training run. Additionally, the Tabular Learner utilizes early stopping with low patience. Combined with the fast convergence method, the neural network was trained for 1-3 epochs per dataset before early stopping terminated training.

[3] https://www.kaggle.com/datasets/dhoogla/ember-2017-v2-features

4 Results

In this section, we present the performance of four models: One Rule, AdaBoost, LightGBM, and FastAI's Tabular Learner, evaluated across all datasets. We report AUROC, F1-score, precision, and recall. These results support our investigation of **RQ1** and **RQ2** by revealing whether increasing model complexity is necessary for high performance, and whether specific features or datasets drive strong results.

4.1 Comparison of the different classifiers

The results for the different datasets using our classifiers are presented in Table 2. Mentions in the table to the One Rule model refer to the One Rule ensemble described in the methodology under subsection 3.3.

Table 2: Performance Summary across Datasets

Dataset	Model	AUROC	Precision	Recall	F1-score
EMBER 2017	One Rule	0.8791	0.8435	0.7075	0.7696
	Adaboost	0.9473	0.9492	0.9451	0.9472
	LightGBM	0.999	0.9891	0.9839	0.9865
	Tabular Learner	0.9875	0.9817	0.9761	0.9789
EMBER 2018	One Rule	0.765	0.6755	0.8199	0.7407
	Adaboost	0.8415	0.8165	0.8811	0.8475
	LightGBM	0.9853	0.9305	0.9529	0.9416
	Tabular Learner	0.9582	0.9152	0.9188	0.917
BODMAS	One Rule	0.9038	0.7899	0.7789	0.7844
	Adaboost	0.9791	0.9644	0.9853	0.9747
	LightGBM	0.9997	0.9968	0.9888	0.9927
	Tabular Learner	0.9878	0.9817	0.9632	0.9724
ERMDS	One Rule	0.9037	0.9559	0.7743	0.8556
	Adaboost	1.0	1.0	1.0	1.0
	LightGBM	1.0	1.0	0.9995	0.9998
	Tabular Learner	0.9577	0.9635	0.9015	0.9315
BenchMFC with Cleanware	One Rule	0.9999	1.0	0.9987	0.9993
	Adaboost	0.9999	1.0	0.9998	0.9999
	LightGBM	1.0	1.0	0.9991	0.9996
	Tabular Learner	0.98	0.9852	0.983	0.9841
MOTIF with Cleanware	One Rule	0.9999	1.0	0.9992	0.9996
	Adaboost	0.9992	1.0	0.9984	0.9992
	LightGBM	0.9996	1.0	0.9988	0.9994
	Tabular Learner	0.9869	0.9163	0.9814	0.9477

Continued on next page...

Dataset	Model	AUROC	Precision	Recall	F1-Score
Ember-features-merged	One Rule	0.760	0.713	0.758	0.735
	Adaboost	0.862	0.837	0.909	0.872
	LightGBM	0.995	0.966	0.964	0.965
	Tabular Learner	0.986	0.954	0.952	0.953
Packware wild	One Rule	0.9074	0.9213	0.7624	0.8344
	Adaboost	0.936	0.9553	0.9438	0.9495
	LightGBM	0.9966	0.9774	0.9813	0.9793
	Tabular Learner	0.9701	0.9579	0.9418	0.9498
CCCS-CIC-AndMal-2020	One Rule	0.9509	0.9509	0.8641	0.9054
	Adaboost	0.9569	0.9725	0.9461	0.9591
	LightGBM	0.9963	0.9838	0.9691	0.9764
	Tabular Learner	0.9919	0.9789	0.9558	0.9672
Kronodroid-2021	One Rule	0.905	0.826	0.861	0.843
	Adaboost	0.952	0.963	0.944	0.953
	LightGBM	0.998	0.980	0.982	0.981
	Tabular Learner	0.993	0.972	0.976	0.974
CIC-MalMem2022	One Rule	0.9958	0.9739	0.9933	0.9835
	Adaboost	0.9997	0.9995	0.9999	0.9997
	LightGBM	1.0	0.9999	0.9989	0.9994
	Tabular Learner	0.9985	0.9986	0.9976	0.9981
CIC-Evasive-PDFMal-2022	One Rule	0.982	0.9805	0.9297	0.9545
	Adaboost	0.9859	0.9876	0.9872	0.9874
	LightGBM	0.9995	0.9998	0.9581	0.9785
	Tabular Learner	0.9962	0.9851	0.9665	0.9757

The One Rule model achieves an F1-Score ranging from 0.74 to perfect score among the datasets, dropping only to 0.735 when considering the merge of all datasets using EMBER features. While increasing the model complexity generally improves classification performance, the Tabular Learner does not seem to outperform LightGBM.

EMBER 2017 yields higher classification accuracy than **EMBER 2018**, as expected, due to its less challenging samples. Among the studied datasets, EMBER 2018 appears to be the most challenging dataset to classify. While the best One Rule models for EMBER 2017 are based on byte histograms or specific header values (e.g., the presence of Graphical User Interface), the best One Rule model for EMBER 2018 is based on byte entropy histograms, specific header values, and properties of the entry point section (e.g., whether it is writable).

BODMAS results exhibit the desired pattern where increasing model complexity is necessary for good classification. The ensemble 1R model still yields 0.90 AUROC, but its malware recognition is clearly lacking at 0.78 F1-score. Still, the jump that is needed in model complexity is not that high because a bet-

ter ensembling method as in adaboost closes a large part of the gap to great classification, performing on par with a moderately large neural network.

ERMDS has two partial datasets with BOS (binary obfuscated samples) and POS (packed obfuscated samples). Results on the individual parts have been gathered, but the results in the table are for the entire dataset regardless of the obfuscation method (BOS+POS). We can consider ERMDS solved because the adaboost model achieves perfect 1.0 scores across all metrics. It is the only dataset in the collection for which the result is that clear-cut.

BenchMFC and MOTIF (with Cleanware): Since the malware and cleanware datasets are from different sources and were collected at different time periods, some bias is already expected. This is clearly visible in the numerical results, with perfect classification immediately achieved by the 1R model for both datasets.

Packware wild results support increasing model complexity with the 1R ensemble, setting a high but imperfect baseline (0.91 AUROC). As a large portion of the samples are packed versions of malware samples from the EMBER datasets, it is good to see that this obfuscation step has little impact on the ability to classify the samples well. However, it should be noted that at 55,000 samples, the packware-wild dataset is significantly smaller than either EMBER dataset.

CCCS-CIC-AndMal-2020 showed a small room for improvement with the 1R ensemble baseline starting at 0.95 AUROC and 0.91 F1-score. It should be noted that during the baseline classification with the 1R model, just 809 out of 9503 features positively contributed to the ensemble (defined as having an individual AUROC above 0.5). More than four fifths of the margin to perfect classification was consumed by the more complicated models (i.e. LightGBM and Tabular Learner at 99+% AUROC compared to 1R and Ada at 95%).

Kronodroid-2021 exhibits a medium difficulty, leading to a baseline score with the ensemble 1R model in the same range as BODMAS and ERMDS (~0.1 margin to perfect classification). The upward trend in model complexity neatly aligns with increases in classification scores to shrink the margin down to <0.3%.

CIC-MalMem2022: The 1R model proves to be as capable of classifying this dataset with a <0.3% margin compared to the other models. From a practical standpoint, this dataset is solved and, therefore, not usable. It is also relatively small, with just under 60,000 samples and no plans for a new version.

CIC-Evasive-PDFMal2022, a dataset from the same research lab, suffers a similar fate, whereby the 1R model sets a very high baseline with <2% margin to reach perfect classification. This margin gets eaten up by both the Tabular Learner and LightGBM. Standing at only 10,000 samples and with substantial erroneous mismatches between the documentation and the actual dataset, it should not be considered for further use.

Figure 1 illustrates the gap between the models in precision and recall, according to the dataset size. The results reveal that the performance gap between complex and simple models becomes more apparent as the dataset size increases. This finding reinforces the claim that insufficiently diverse datasets can artificially inflate model performance.

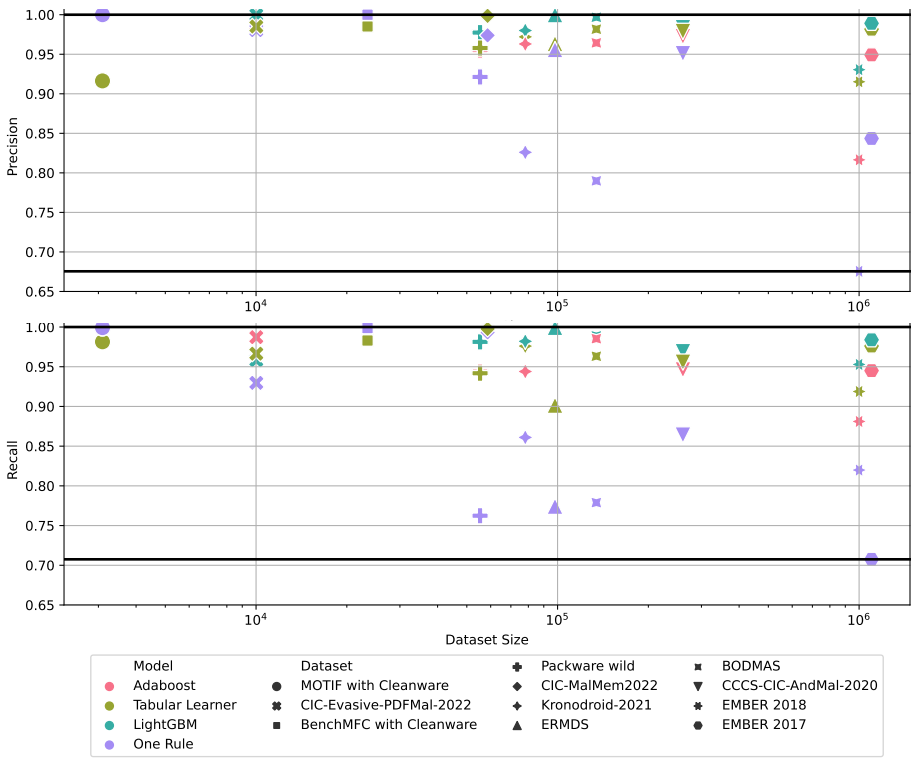


Fig. 1. Performance gap between models according to the dataset size (in log scale). Bold lines indicate the minimum and maximum values of the scores.

Detection performance alone doesn't fully reflect model quality. Generalization and robustness against adversarial attacks are also important. Simple models, like the One Rule classifier, which depend on a limited set of features, are particularly vulnerable to attacks that target those specific features. This study doesn't claim simple models are superior. Instead, their unexpectedly strong performance on academic datasets highlights potential biases in the data and features, suggesting these benchmarks may oversimplify the real-world complexity of malware detection.

5 Discussion

In this section, we discuss our results to answer our different research questions.

Efficiency of simple classifiers Simple 1R ensembles deliver surprisingly strong malware detection performance, especially on datasets built using cleanware from external sources like BenchMFC or MOTIF, or from public repositories as seen in ERMDS. These sources are often used when cleanware isn't available, but as previous work has shown [32], they can lead to overly optimistic

results. Interestingly, this effect appears even when cleanware is collected using methods similar to those used for malware.

As datasets grow, more complex models tend to become relevant. However, LightGBM already achieves excellent results on the merged EMBER dataset, which questions the added value of deep learning–especially considering its higher training cost and reduced interpretability for only modest performance gains.

In response to **RQ1**, the findings show that simple models can be highly effective on academic datasets. This highlights the importance of establishing baselines with simple models and evaluating them on more diverse, realistic datasets than those typically used in research. On average, simple models (1R and Ada) reached 0.92 AUROC on EMBER-feature datasets, compared to 0.99 for more complex models (LightGBM and Tabular Learner)–a 7% absolute and 7.6% relative difference.

Global feature importance From Figure 2, we can observe that datasets share similarities in their feature ranking. However, some features appear to be specific to precise datasets, which deserve investigation.

- **Histogram features.** While histogram features seem decisive for the MOTIF dataset, generally ranking among their top features, these are less represented in other datasets.
- **String features.** Histogram based on strings seems quite specific to some datasets. However, the number of URL strings is used by different models. Among the non-EMBER features extracted for the Packware dataset, the bigram "ww" appears as an efficient feature that could be related to the URL feature. The number of MZ bigrams (possibly indicating embedded executables into the binary) is also important for both EMBER datasets.
- **General file information and headers.** Different headers have high impact in specific datasets, like the WINDOWS_GUI header for BODMAS and both EMBER datasets. This could indicate an imbalance between console and GUI-based executables among malware. We also observe that the timestamp is used in BODMAS, ERMDS, and EMBER as one of the top features. Although the timestamp field could contain artifacts related to specific compilers or obfuscation tools, its usage should be monitored in machine learning models to avoid being used as a shortcut to classify malware and cleanware from different time periods.
- **Section information.** Among others, the code section size and the number of sections readable and executable appear as important features in BODMAS and EMBER. When observing features related to the entry point section name, we discovered a bug in the original script used to extract EMBER features. Each character of the entry point section name is mapped to a different bin, rather than the entire name being mapped to a single bin. This problem is observed in BODMAS, MOTIF, and BenchMFC, which appear to be derived from the original EMBER script with an old sklearn version. This issue has been addressed in sklearn version 1.3, which requires users to provide a list of strings to the function employed for the hashing trick.

- **Import information.** Most of the best features across datasets correspond to imported API calls. In the 1,024 hashed bins of EMBER features, many collisions are observed–likely due to C++ name mangling, the use of ordinals, or the presence of numerous benign functions (e.g., related to GUI handling). Still, some API calls clearly stand out, especially those tied to evasion techniques and environment checks (e.g., `GetConsoleCP`, `GetComputerNameA`, `NtQueryPerformanceCounter`). Other relevant API include `GetProcAddress` (1839) for dynamic API loading or registry and crypto-related calls like `SHRegGetValueA` (1980) and `CryptAcquireContextA` (1999). While these features are less prominent in ERMDS, the model continues to leverage evasion-related imports–such as `EnumWindows` (1697), used to enumerate open windows and detect security tools.
- **Export information.** The impact of those features seems limited across datasets, although export table size is used for BODMAS and BenchMFC.
- **Data directories information.** Features related to certificates, imports, and resource tables seem particularly interesting for the BODMAS and EMBER datasets. The presence of a certificate was already highlighted in literature [38] as a possible bias that could inflate performance (i.e., Dataset with only legitimate software with certificates) and should therefore be monitored. Properties of the Import Address Table (IAT) and resource able are expected since they are generally influenced by packing and obfuscation.

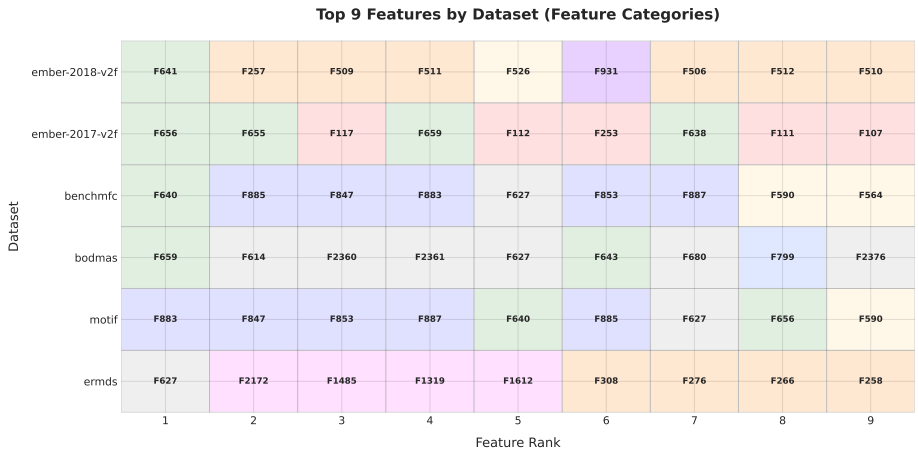


Fig. 2. Matrix representation of the top-9 features identified by the 1R model for the EMBER-based datasets. Categories have been derived from the large feature table in Appendix A.

Constructing stable feature hierarchies for EMBER2017 An additional experiment was performed on the EMBER2017 dataset, given that it and its successor are the highest quality datasets in this analysis. A feature selection method called hierarchical feature block ranking (HFBR) was executed to get a stable feature hierarchy [13]. The algorithm is a two stage process of multi-

collinear feature removal through the variance inflation procedure, followed by repeat unbiased measurements of feature importance through cross-validated forests of extremely randomized decision trees. Through use of the Friedman chi-square test, it can be established if at least one of the features is significantly more predictive than the others. If that is the case then pairwise Wilcoxon signed rank tests (one-sided) are performed on the feature importance measures in descending order. Given the high number of tests, the Benjamini-Hochberg procedure is used to control the false discovery rate. The corrected sequence of Wilcoxon signed rank test p-values places features in blocks, with the null hypothesis being: the estimations of the feature importances of the higher average feature could have been drawn from the same distribution as the estimations for the lower average feature. A lack of rejection indicates no significant difference. This sequence of (non-)rejections of the null hypothesis breaks the feature set into blocks. Features within the same block are considered interchangeable in predictive power. Features in lower-ranked blocks are consistently dominated by features in higher-ranked blocks. Through repeats of seconds step of the HFBR algorithm at different data sampling rates, a set of block rankings is created which is ultimately reduced by placing features in their most common block as the final feature block hiearchy. For the EMBER dataset 226 features were identified as multicollinear and removed prior to further analysis. The remainder (over 2100 features) landed in 4 blocks. A comparison with the top 9 features discovered through simpler methods in the previous paragraph (figure 2) revealed a similar selection of top features. Features 656, 659 and 638 are also in the top-9. However, the simpler method did not remove multicollinear features so it kept features 655, 117, 112 and 111 in the top-9. The first few feature blocks for EMBER2017 did not contain many features so they are listed here. Block 1: F638, block 2: F622 and F619, block 3: F656, F659, F666 and F640, block 4: all remaining features. The distribution of features F638, F622 and F619 are represented in Figure 3. It appears simple classifier could be built by considering DLL indicator in COFF header (F638), presence of relocation (F622) and debug info (F619). An initial test with this selection of 7 features inside the Adaboost model with 1R sublearners reached an AUROC of 0.82. The same model with access to the full feature set of 2381 features reached an AUROC of 0.95. This article will not dive deeper into classification performance with feature selection, but given our preliminary results, we expect that there is significant margin to create much leaner models with similar classification performance.

Comparison with discriminant feature with 1R Figures 4 illustrates the distribution of the three top features used by the 1R classifier on BODAMS. The 1R rule enables us to identify features that effectively split the data space and highlight potential shortcuts employed by the models in classification. While complex classifiers like LightGBM use some of these features, there are also features used by the best 1R classifier that do not seem to have high importance in other classifiers. In general, histogram features (byte, entropy, and string-based) allow the building of effective 1R ensembles and are highly represented for most datasets (especially EMBER2018). The header features related

to GUI use (MOTIF, EMBER 2017, BODMAS) and the timestamps (MOTIF, BenchMFC, BODMAS), are also represented in the most important features, allowing for easy discrimination of malware from cleanware. Another interesting bin (Feature 659) concerning the program header is linked to DLL characteristics (`'DYNAMIC_BASE'`, `'HIGH_ENTROPY_VA'`, `'NO_ISOLATION'`). While the options `'DYNAMIC_BASE'` and `'HIGH_ENTROPY_VA'` are related to address space layout randomization (more represented in cleanware), `'NO_ISOLATION'` forces using system DLLs to ensure compatibility on more systems (more represented in malware). The number of URL strings present in the binary is also a good discriminant for several datasets: BODMAS, MOTIF, but also Packware (since the `'ww'` bigram is likely related to a URL). While cleanware samples used with MOTIF and BenchMFC are extracted from public resources commonly used in literature, cleanware samples for BODMAS were extracted from a company database. In general, some of the best features are consistent with malicious behavior (e.g., the *dll* characteristics), but others, like the timestamp, seem to bring excessive discriminative power for the classifier.

Regarding **RQ2**, we observe that certain features disproportionately influence the classification process across multiple datasets. Features such as timestamps and specific string patterns (e.g., URLs) are often strongly discriminating features, but while they effectively separate malware from cleanware, they may reflect dataset-specific characteristics rather than actual malware properties. This raises the question of whether such features introduce biases, a topic that warrants further investigation. A deeper investigation into their potential to introduce biases, particularly in real-world settings, would be a valuable direction for future research.

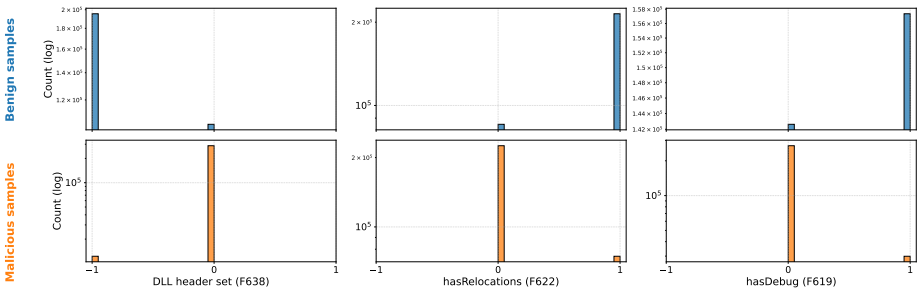


Fig. 3. Distributions of three best features for EMBER2017 according to hierarchy.

6 Conclusion

This study evaluated the performance of simple and complex machine learning models across multiple academic malware datasets. Our findings demonstrate that simple models, particularly the One-Rule classifier, can achieve comparable performance to more complex approaches such as LightGBM and deep learning

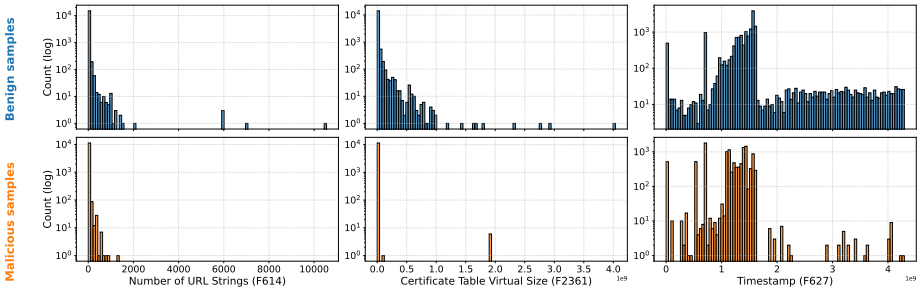


Fig. 4. Feature distributions of three best 1R model for BODMAS

models. This observation holds across various datasets, including well-established benchmarks such as EMBER and newer collections like BODMAS and ERMDS.

Our analysis revealed that certain features, such as timestamps and GUI-related headers, can provide strong discriminative power but may introduce biases in the classification process. The consistent high performance of simple models across academic datasets suggests that these collections might not fully capture the complexity and variability of real-world malware detection scenarios.

These findings have important implications for malware detection research. First, they emphasize the importance of establishing baselines with simple models before exploring more complex approaches. Second, they emphasize the need for more diverse and challenging datasets that more accurately represent real-world scenarios. Third, they suggest that the current trend toward increasingly complex models may not always be justified, especially given the higher computational costs and reduced interpretability.

Future work should focus on developing more diverse and challenging datasets that better represent the real-world distribution of malware. Additionally, research should investigate methods to reduce bias in feature selection, ensuring that models rely on meaningful properties over superficial patterns. Ultimately, there is a need for more studies that evaluate model performance in real-world deployment scenarios, rather than relying only on academic benchmarks.

This work challenges the assumption that complexity always leads to better performance. We urge model researchers to be more thoughtful in their model selection and dataset development for malware detection research.

Acknowledgments

This research is supported by the Walloon regions CyberExcellence program (grant #2110186). This research was also partially supported by the AIDE project funded by the Belgian SPF BOSA with reference number 06.40.32.33.00.10.

References

1. Chocolatey package manager, https://chocolatey.org/

2. Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., Kruegel, C.: When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In: Network and Distributed Systems Security (NDSS) Symposium 2020 (2020)
3. Anderson, H.S., Roth, P.: Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
4. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium. pp. 3971–3988 (2022)
5. Bertrand Van Ouytsel, C.H., Legay, A.: Malware analysis with symbolic execution and graph kernel. In: Nordic Conference on Secure IT Systems. pp. 292–310. Springer (2022)
6. Bolun Wu, Yuanhang Xu, F.Z.: Malware classification by learning semantic and structural features of control flow graphs. In: 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 540–547. IEEE (2021)
7. Carrier, T., Victor, P., Tekeoglu, A., Lashkari, A.H.: Detecting obfuscated malware using memory feature engineering. In: Icissp. pp. 177–188 (2022)
8. Carrier, T., Victor, P., Tekeoglu, A., Lashkari, A.H., D'hooge, L.: Cic-malmem2022 (2023), https://www.kaggle.com/dsv/6248429
9. Dambra, S., Han, Y., Aonzo, S., Kotzias, P., Vitale, A., Caballero, J., Balzarotti, D., Bilge, L.: Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance. In: Proceedings of the 2023 Conference on Computer and Communications Security. pp. 60–74 (2023)
10. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
11. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
12. D'hooge, L., Verkerken, M., Wauters, T., De Turck, F., Volckaert, B.: Castles built on sand: Observations from classifying academic cybersecurity datasets with minimalist methods. In: 8th International Conference on Internet of Things, Big Data and Security (IoTBDS). pp. 61–72. SCITEPRESS (2023)
13. Dhooge, L., Verkerken, M., Wauters, T., Volckaert, B., De Turck, F.: Hierarchical feature block ranking for data-efficient intrusion detection modeling. Computer Networks **201**, 108613 (2021). https://doi.org/https://doi.org/10.1016/j.comnet.2021.108613, https://www.sciencedirect.com/science/article/pii/S1389128621005107
14. fastai: Tabular learner - fastai. https://docs.fast.ai/tabular.learner.html (2023), accessed: [6 Jan 25]
15. Feng, P., Gai, L., Yang, L., Wang, Q., Li, T., Xi, N., Ma, J.: Dawngnn: Documentation augmented windows malware detection using graph neural network. Computers & Security p. 103788 (2024)
16. Freund, Y., Schapire, R., Abe, N.: A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence **14**(771-780), 1612 (1999)
17. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55**(1), 119–139 (1997)
18. Guerra-Manzanares, A., Bahsi, H., Nõmm, S.: Kronodroid: time-based hybrid-featured dataset for effective android malware detection and characterization. Computers & Security **110**, 102399 (2021)

19. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Machine learning **11**, 63–90 (1993)
20. Howard, J., Gugger, S.: Fastai: a layered api for deep learning. Information **11**(2), 108 (2020)
21. Issakhani, M., Victor, P., Tekeoglu, A., Lashkari, A.H.: Pdf malware detection based on stacking learning. In: ICISSP. pp. 562–570 (2022)
22. Issakhani, M., Victor, P., Tekeoglu, A., Lashkari, A.H., D'hooge, L.: Cic-evasive-pdfmal2022 (2022), https://www.kaggle.com/dsv/4059725
23. Jia, L., Yang, Y., Tang, B., Jiang, Z.: Ermds: A obfuscation dataset for evaluating robustness of learning-based malware detection system. BenchCouncil Transactions on Benchmarks, Standards and Evaluations **3**(1), 100106 (2023)
24. Jiang, Y., Li, G., Li, S., Guo, Y.: Benchmfc: A benchmark dataset for trustworthy malware family classification under concept drift. Computers & Security **139**, 103706 (2024)
25. Joyce, R.J., Amlani, D., Nicholas, C., Raff, E.: Motif: A malware reference dataset with ground truth family labels. Computers & Security **124**, 102921 (2023)
26. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems **30** (2017)
27. Keyes, D.S., Li, B., Kaur, G., Lashkari, A.H., Gagnon, F., Massicotte, F.: Entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In: 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS). pp. 1–12. IEEE (2021)
28. Keyes, D.S., Li, B., Kaur, G., Lashkari, A.H., Gagnon, F., Massicotte, F., Rahali, A., Taheri, L.: Cccs-cic-andmal-2020 (2022), https://www.kaggle.com/dsv/4177444
29. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
30. Li, C., Cheng, Z., Zhu, H., Wang, L., Lv, Q., Wang, Y., Li, N., Sun, D.: Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning. Computers & Security **122**, 102872 (2022)
31. Ling, X., Wu, L., Deng, W., Qu, Z., Zhang, J., Zhang, S., Ma, T., Wang, B., Wu, C., Ji, S.: Malgraph: Hierarchical graph neural networks for robust windows malware detection. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications. pp. 1998–2007. IEEE (2022)
32. Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX security symposium. pp. 729–746 (2019)
33. Ponte, A., Trizna, D., Demetrio, L., Biggio, B., Ogbu, I.T., Roli, F.: Slifer: Investigating performance and robustness of malware detection pipelines. Computers & Security **150**, 104264 (2025)
34. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K.: Malware detection by eating a whole exe. In: Workshops at the thirty-second AAAI conference on artificial intelligence (2018)
35. Rahali, A., Lashkari, A.H., Kaur, G., Taheri, L., Gagnon, F., Massicotte, F.: Didroid: Android malware classification and characterization using deep image learning. In: Proceedings of the 2020 10th International Conference on Communication and Network Security. pp. 70–82 (2020)
36. Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you need. Information Fusion **81**, 84–90 (2022)

37. Smith, L.N., Topin, N.: Super-convergence: Very fast training of residual networks using large learning rates (2018), https://openreview.net/forum?id=H1A5ztj3b
38. Smith, M.R., Johnson, N.T., Ingram, J.B., Carbajal, A.J., Haus, B.I., Domschot, E., Ramyaa, R., Lamb, C.C., Verzi, S.J., Kegelmeyer, W.P.: Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In: 13th ACM Workshop on Artificial Intelligence and Security. pp. 49–60 (2020)
39. Sébastien, C.: Portable freeware dataset, https://doi.org/10.5281/zenodo.2528209
40. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th annual international conference on machine learning. pp. 1113–1120 (2009)
41. Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., Wang, G.: Bodmas: An open dataset for learning based temporal analysis of pe malware. In: 4th Deep Learning and Security Workshop (2021)
42. Zhou, S., Li, H., Fu, X., Han, D., He, X.: Novel multi-classification dynamic detection model for android malware based on improved zebra optimization algorithm and lightgbm. Sensors (Basel, Switzerland) **24**(18), 5975 (2024)

A Feature index table

This appendix provides the complete feature index table used throughout the experiments (EMBER feature set). It serves as a reference for interpreting the feature importance results discussed in Section 5. Each feature is listed with a consistent index and description to aid reproducibility and cross-dataset comparison. Having it as part of the body of the article also serves as a long-term reference for the EMBER feature set because the datasets themselves never encoded the names as table headers.

Index	Feature Name	Description
1-256	byteHistogram	Byte value distribution histogram (256 bins)
257-512	byteEntropyHistogram	Entropy histogram of byte values (256 bins)
513	numStrings	Number of strings in the file
514	averageStringLength	Average length of strings
515-610	printableStringDistribution	Histogram of printable characters in strings (96 bins)
611	numPrintables	Number of printable characters in strings
612	entropyStrings	Entropy of extracted strings
613	numPathsStrings	Number of path-related strings
614	numUrlsStrings	Number of URL-related strings
615	numRegistryStrings	Number of registry-related strings
616	numMzString	Number of "MZ" header occurrences in strings
617-618	size/vsize	File size and virtual size in bytes
619	hasDebug	Presence of debug information
620	exports	Number of exported functions
621	imports	Number of imported functions
622	hasRelocations	Presence of relocation entries
623	hasResources	Presence of resource section

624	hasSignature	Presence of digital signature
625	hasTls	Presence of TLS callbacks
626	symbols	Number of symbols in the file
627	timestamp	Compilation timestamp
628-637	headerFileInfoHashed	COFF machine header info (10 bins)
638-647	headerFileInfoHashed	COFF header characteristics (10 bins)
648-657	headerFileInfoHashed	Optional header subsystem info (10 bins)
658-667	headerFileInfoHashed	Optional header dll characteristics (10 bins)
668-677	headerFileInfoHashed	Optional header magic info (10 bins)
678-679	major/minorImageVersion	Major and minor image version
680-681	major/minorLinkerVersion	Major and minor linker version
682-683	major/minorOSVersion	Major and minor OS version
684-685	major/minorSubsystemVersion	Major and minor subsystem version
686	sizeofCode	Size of code section
687	sizeofHeaders	Size of headers
688	sizeofHeapCommit	Size of committed heap
689	numSections	Number of sections
690	numSectionsZeroSize	Number of zero-sized sections
691	numSectionsEmptyName	Number of unnamed sections
692	numSectionsRx	Number of executable sections
693	numSectionsW	Number of writable sections
694-743	sectionSizeHashed	Hashed section sizes (50 bins)
744-793	sectionEntropyHashed	Hashed section entropy (50 bins)
794-843	sectionVsizeHashed	Hashed virtual section sizes (50 bins)
844-893	entrySectionNameHashed	Hashed entry section name (50 bins)
894-943	charactSectionHashed	Hashed section characteristics (50 bins)
944-1199	importsDLLHashed	Hashed DLL imports (256 bins)
1200-2223	importsAPIHashed	Hashed API imports (1024 bins)
2224-2351	exportsFuncHashed	Hashed exported functions (128 bins)
2352-2353	exportTableSize/vsize	Export table size and virtual size
2354-2355	importTableSize/vsize	Import table size and virtual size
2356-2357	resourceTableSize/vsize	Resource table size and virtual size
2358-2359	exceptionTableSize/vsize	Exception table size and virtual size
2360-2361	certificateTableSize/vsize	Certificate table size and virtual size
2362-2363	baseRelocationTableSize/vsize	Base relocation table size and virtual size
2364-2365	debugSize/vsize	Debug section size and virtual size
2366-2367	architectureSize/vsize	Architecture data size and virtual size
2368-2369	globalPtrSize/vsize	Global pointer table size and virtual size
2370-2371	tlsTableSize/vsize	TLS table size and virtual size
2372-2373	loadConfigTableSize/vsize	Load configuration table size and virtual size
2374-2375	boundImportSize/vsize	Bound import table size and virtual size
2376-2377	iatSize/vsize	Import Address Table (IAT) size and virtual size
2378-2379	delayImportDescriptorSize/vsize	Delay import descriptor size and virtual size
2380-2381	clrRuntimeHeaderSize/vsize	CLR runtime header size and virtual size

Table 3: EMBER feature set mapping feature number to feature name.