A Unified Comparison of Tabular and Graph-Based Feature Representations in Machine Learning for Malware Detection

Samy Bettaieb*†, Serena Lucca*†, Charles-Henry Bertrand Van Ouytsel†, Axel Legay‡, Etienne Rivière†
†*UCLouvain, Louvain-la-Neuve, Belgium*
‡*Nexova, Libin, Belgium*
Email: {samy.bettaieb, serena.lucca, charles-henry.bertrand, etienne.riviere}@uclouvain.be

2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) | 979-8-3315-9546-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/EUROSPW67616.2025.00012

***Abstract*—Feature representation is a key factor in machine learning-based malware detection, affecting the information expressed and used for detection, the choice of the classifier, and computational efficiency. While both tabular and graph-based feature representations have been widely studied, we lack a systematic comparison under unified conditions. This study compares tabular and graph-based features extracted through static and dynamic analysis for malware detection. We evaluate these representations using state-of-the-art models on a unified dataset of Windows PE32 files. Our analysis focuses on three aspects: computational time required for feature extraction, detection performance, and robustness against adversarial attacks. To ensure a fair evaluation, we assess detection performance on both in-distribution and out-of-distribution datasets, highlighting the trade-offs between feature complexity, model accuracy, and real-world applicability. We find that tabular features perform best in most scenarios, making the cost of building graphs not always justified.**

***Index Terms*—Malware Detection, Feature representation, Machine learning**

1. Introduction

In recent years, the cybersecurity threat landscape has evolved significantly, with sophisticated malware increasingly targeting individuals and organizations [1], making robust and efficient malware detection tools necessary.

Traditional signature-based detection methods struggle to keep up with emerging malware variants, making machine learning (ML)-based approaches an attractive alternative. By learning patterns from observed malicious and benign samples, ML models can generalize to unseen threats. However, their effectiveness heavily depends on how we represent the features extracted from the samples. Indeed, the feature representation impacts ML performance, interpretability, robustness against adversarial attacks, and overall computational efficiency.

Consequently, determining the most informative way to represent malware data has become a major challenge in ML-based malware detection [2]. Features can be represented in many different ways, with two prominent approaches being tabular and graph-based representations. Tabular features represent malware samples as structured numerical data extracted from static attributes (e.g., file metadata, opcode sequences) or dynamic behaviors (e.g., API call sequences, system interactions). These features are easy to process with traditional ML models such as decision trees and random forests. In contrast, graph-based representations capture relationships between entities, such as function calls or blocks of binary code. They can either be extracted using static analysis (Control flow Graph, Function Call Graph) or dynamic analysis (System Call Dependency Graph, API Call Graph). While tabular features offer simplicity and efficiency, graph-based representations capture richer structural relationships, albeit with a higher computational cost. Graphs are particularly well-suited for deep learning models, such as Graph Neural Networks (GNNs) [3]. However, it remains unclear whether the higher extraction cost of graph-based features is outweighed by a clear gain in model performance and robustness when compared to tabular features, or if the effectiveness of either representation is impacted by the use of either static or dynamic analysis.

Despite extensive research using tabular [4]–[7] and graph-based [8]–[12] features, a systematic comparison of feature representations on a unified dataset is lacking. A fair assessment requires both feature types to be extracted from the same samples under identical conditions to mitigate dataset bias. However, this approach might reduce the dataset size, potentially introducing other biases. Additionally, packing, a common obfuscation technique, can impact ML-based malware detection performance [5]. It is, therefore, necessary to evaluate its impact.

In this study, we aim to answer the following research questions in the context of ML-based malware detection:

- RQ1: How do graph-based features and tabular features, extracted through static or dynamic analysis, compare in performance?
- RQ2: Does the addition of packing impact the different feature representations?
- RQ3: How do adversarial attacks influence the performance of feature representations extracted with static analysis, before and after retraining is applied?

Our results show that static features, particularly tabular ones, perform well in malware detection, even on out-of-distribution or packed datasets. Unfortunately, these features are prone to simple static adversarial attacks and the retraining can worsen the scores if the attack does not cover enough domain space. Our code [1] and datasets [2] of features are open-source.

*These authors contributed equally to this work.
*AI was **only** used to correct grammar and style of the text.*

[1]https://doi.org/10.5281/zenodo.15237014
[2]https://www.kaggle.com/sigehmireglbjevf/datasets (Version #1)

© 2025, Samy Bettaieb. Under license to IEEE.

DOI 10.1109/EuroSPW67616.2025.00012
Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

This paper is structured as follows. Section 2 provides background on ML-based malware detection, and a review of related work on feature representations. Section 3 details the methodology, including the feature representations and models considered. Section 4 presents the experimental setup and results, followed by a discussion in Section 5, where we address the research questions. Section 6 covers limitations and potential perspectives. Finally, we conclude this paper in Section 7.

2. Background & Related Work

This section outlines key concepts for this study. We first discuss methods for extracting meaningful features from malware samples, followed by an overview of various malware representations, including tabular, graph-based, and hybrid approaches. Finally, we cover adversarial attacks against ML-based malware detection systems.

2.1. Feature Extraction

Extracting features from binaries is a major step in an ML-based malware detection pipeline. This process can be performed through static or dynamic analysis, each offering distinct advantages and limitations.

Static analysis examines a binary without executing it. This method is the most efficient but struggles with static obfuscation techniques, such as packing [5], [13]. The simplest approach is to extract features directly from raw bytes in a binary [14]. For more detailed features, PE files can be parsed using libraries such as LIEF [15] or pefile [16] to extract structural features and metadata. More advanced tools, such as IDA Pro [17], can perform disassembly and decompilation to extract Control Flow Graphs (CFGs) and Function Call Graphs (FCGs, a directed graph that represents the calling relationships between functions in a program).

Dynamic analysis executes a binary in a controlled environment to observe its behavior, revealing runtime activities like API calls, file modifications, and network usage. It can be performed via different methods, including sandboxing (e.g., Cuckoo [18] or CAPEv2 [19]), emulation (e.g., QEMU [20] or Speakeasy [21]), and instrumentation (e.g., DynamoRIO [22] or Frida [23]). While dynamic analysis is more effective against obfuscated malware than static analysis, it is resource-intensive and vulnerable to evasion techniques that detect analysis environments.

2.2. Malware Representation & Detection

Feature representations significantly impact malware detection performance and robustness. Common approaches include tabular features, graphs, images, and raw bytes [14], each capturing different structural or behavioral malware characteristics.

Tabular representations remain popular due to their simplicity and effectiveness. The EMBER features dataset [4] has become a benchmark for static malware detection. It uses 2,381 static features. Dambra *et al.* [7] compare and combine static and dynamic features, using the static features of Aghakhani *et al.* [5] and selecting dynamic features based on a literature review. Their results reveal that static features outperform dynamic ones in malware detection, with only marginal gains from combining both, likely due to the high sparsity of dynamic features. Rabadi *et al.* [6] use dynamic execution to extract features from API calls and their arguments, transforming them into a tabular format for classification using XGBoost [24].

Graph-based representations capture structural and behavioral information. Recent work includes DawnGNN [12], which builds API call graphs using dynamic analysis with the Cuckoo Sandbox [18]. They use a Graph Attention Network (GAT) [25] model for classification. In these API call graphs, nodes are API calls, and the directed edges represent sequential connections. Li *et al.* [9] extend API call graphs by incorporating arguments as edge features. They build a graph neural network (GNN) combining a modified graph isomorphism network (GINE) [26] and a GAT [25]. Their results show that using a GINE alone performs best for the malware detection task. System Call Dependency Graphs (SCDGs) [8], [27] extend API call graphs by including different dependencies and edge types in addition to sequential dependencies. Bertrand van Ouytsel *et al.* [8] create SCDGs and evaluate different graph-building strategies. For static analysis, Malgraph [10], a hierarchical GNN, combines function call graphs and control flow graphs to capture both inter-function and intra-function semantics, to improve detection accuracy and robustness against adversarial attacks.

Hybrid approaches combine feature representations to improve performance and robustness. Quo-vadis [28] proposes a hybrid architecture that combines static (EMBER) and dynamic (API sequences) features using kernel emulation for efficient behavioral analysis. It shows that dynamic analysis benefits from the EMBER features.

SLIFER [29] introduces a cascading framework that starts with lightweight static analysis methods and escalates to dynamic analysis only when needed, balancing accuracy with resource efficiency.

Our study presents a systematic comparison of feature representations on a unified dataset in order to identify the most efficient and robust feature representations.

2.3. Adversarial Attacks & Model Hardening

In ML-based malware detection, adversarial attacks aim to create malware samples that evade detection from a targeted model. These attacks can be categorized into feature space attacks, which modify the model's input representation, and problem space attacks, which alter the malware binary. Demetrio *et al.* [30] introduce `secml-malware`, an open-source Python library for adversarial attacks on Windows malware detectors. It attacks models in a white-box or black-box fashion, by applying functionality-preserving manipulations on Windows malware. Recent work also relies on reinforcement learning (RL) to perform adversarial attacks. `Mab-Malware` [31] models the attack as a series of independent actions, using a probabilistic approach to balance exploration and exploitation when selecting modifications for malware samples. The framework aims to minimize changes in adversarial examples (AEs) to ensure accurate reward assignment in RL. MTMG [32] uses RL to dynamically

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

select obfuscation techniques and parameters. It generates AEs that can deceive multiple ML-based malware detectors while preserving malicious functionality. Another aspect associated to adversarial attacks is model hardening, which aims at improving robustness. Several approaches allow hardening a model. Among these, adversarial training [33] gradually introduces AEs into the training set. A simpler approach, referred to as adversarial retraining, generates AEs from a trained model and incorporates them into the dataset before retraining from scratch. AutoRobust [34] applies this concept by generating adversarial samples through the problem space for a feature representation based on dynamic analysis (CAPEv2 sandbox reports). These samples are used for retraining the targeted model, eventually achieving a 0% attack success rate.

3. Methodology

The goal of our study is to compare the performance of ML-based malware detection models using tabular and graph-based feature representations, extracted through static or dynamic analysis. This section introduces the different components of our approach, which are summarized in Figure 1. We first describe how we collect our datasets. Next, we present the static features used for our comparison and the tools involved in their extraction. We then describe the dynamic analysis setup , followed by the definition of the dynamic features. Afterward, we explain the models that we use for each feature representation, give details on the more complex architectures, and justify our model choices. Finally, we present the methodology used to evaluate the robustness of each approach, namely packing obfuscation and adversarial attacks performed by MAB-Malware.

3.1. Datasets

To the best of our knowledge, there is no reference dataset of executable files for our use case. Specifically, there are no large collection of cleanware binaries openly available online. Hence, we decided to use an approach similar to state-of-the-art work [7], [35], [36]. In order to reduce bias between the feature representations, we limit ourselves to the analysis of 32-bit portable executable as the vast majority of malware is 32-bit for compatibility with older systems. This restriction to 32-bit binaries has already been done in the literature [7]. Our main cleanware dataset consists of PE32 files originating from a clean Windows 10 virtual machine on which we installed packages from the Chocolatey community package manager [37]. We collected 8,421 PE32 files, of which 6,792 passed all of the feature extraction processes. To limit feature extraction time, we consider a subset of the BODMAS [38] dataset for our main malware dataset. We randomly sample 500 files from each of the 20 most common PE32 families in the dataset, resulting in a collection of 10,000 PE32 files. 7,676 samples passed the complete feature extraction process. We randomly select 6,792 of these samples to work with a balanced dataset. The initial and final distributions of the malware families are available in Appendix B. We split both datasets into 70% for training and 30% for testing. We perform a temporal split (older samples for training and more recent samples for testing) based on the provided timestamp for the BODMAS dataset (i.e., the first-seen time of a sample based on VirusTotal reports [39]). We do not have reliable timestamps for the cleanware installed with Chocolatey so this dataset is split randomly. We refer to this dataset as the **IID** dataset.

To better comprehend the robustness of each feature, we add two more out-of-distribution (OOD) datasets from different sources. For malware, we collected 1,000 samples from MalwareBazaar [40] in November 2024, of which 855 were 32-bit portable executable. The family distribution for the 855 samples can be found in Appendix B. 804 samples produced valid results for all feature representations. For cleanware, we use the Portable Freeware Dataset [41], which contains 879 PE32 files. 817 samples got through the entire feature extraction process but only 804 were kept to maintain a balanced dataset. Both of these datasets were used solely for testing purposes. We refer to the combination of these two datasets as the **OOD** dataset. A summary of the datasets that we use for our experiments can be found in Table 1.

TABLE 1. DATASET SUMMARY

Dataset	Origin	samples
train **IID**	BODMAS	4,754
	Chocolatey	4,754
test **IID**	BODMAS	2,038
	Chocolatey	2,038
test **OOD**	MalwareBazaar	804
	Portable Freeware	804

3.2. Static Features Extraction

The static tabular features used in this study are derived from the EMBER dataset [4], a well-established benchmark for ML-based malware detection. These features include byte histogram, byte entropy histogram, strings, general file information, section characteristics, header information, imported libraries and functions, and exported functions, providing a comprehensive representation of malware samples based on their static properties. Static graph-based features are represented using a hierarchical graph structure that integrates Function Call Graphs (FCGs) and Control Flow Graphs (CFGs), following the approach of MalGraph [10]. This combined representation captures both inter-function relationships and intra-function control flow details. The graphs are extracted using IDA Pro 8.6 for accurate binary code analysis.

3.3. Dynamic Analysis: Cuckoo Sandbox

We use Cuckoo Sandbox [18] to analyze the behavior of benign and malicious samples in a controlled virtual environment. Cuckoo is widely used in the literature for dynamic malware analysis [6], [9], [12], [42], [43], for its flexibility and support for behavioral logging. Our setup consists of a host machine running Ubuntu 18.04.6 LTS and a guest machine with Windows 7 64-bit. We execute each sample in the guest system for two minutes while Cuckoo monitors and logs its behavior. Our use of a two-minute execution time is supported by previous work [44] identifying that most of a sample's behavior is observed

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

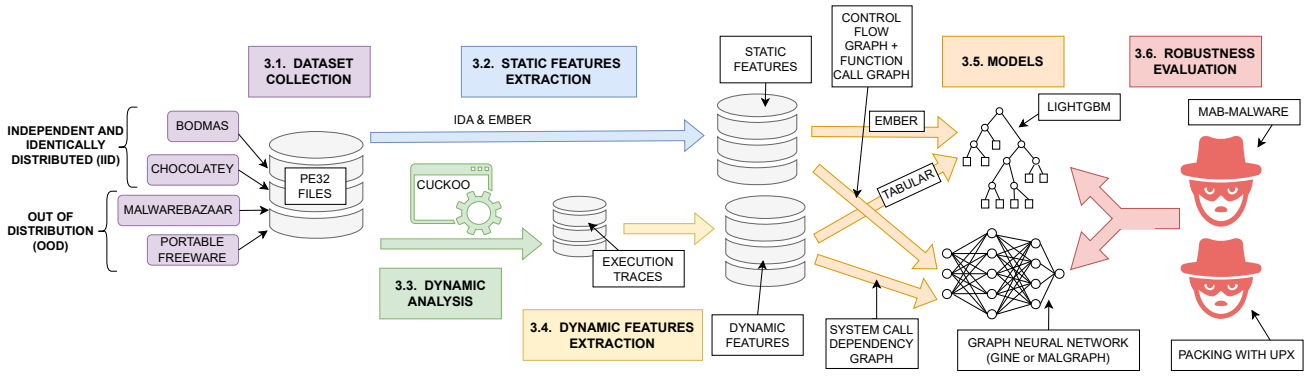

Figure 1. Strategy overview

within the first two minutes. The generated logs, containing API call sequences and their arguments, are compiled into a behavioral analysis report and stored on the host for further processing. We extract both graphs and tabular features based on dynamic analysis from these reports. We use Cuckoo 2.0.7 instead of the latest Cuckoo 3 [45], as the latter does not provide behavioral reports at the time of writing this paper.

3.4. Dynamic Features Extraction

We use the same dynamic tabular features as Dambra *et al.* [7]. These features capture runtime behaviors such as file activity, use of mutexes, registry operations, services, processes, and threads. Although the original work also included a network activity category, we did not use it as our dynamic environment did not include any network simulation. We extract the named features (file names, registry names, mutex names, and process names) from the train set only and we consider them if they appeared in at least five samples. We extracted all of the features from cuckoo reports.

For the dynamic graph-based features, our work focuses on System Call Dependency Graphs (SCDGs) as initially presented by Christodorescu et al. [27]. More specifically, we parse each execution trace so that a node is created for each API call. We merge nodes if they have the same function name and the same PID. Then, we create four types of edges between the nodes, inspired by several research works on graph-based representations [8], [9], [27]. A type-1 edge links two nodes that are consecutive in the execution trace. A type-2 edge links nodes that share an argument (i.e., have a common substring of at least five characters in their arguments). A type-3 edge is similar to a type-2 but concerns the first node's return value and the other node's arguments. Finally, a type-4 edge represents that a node's function name is in another node's arguments.

3.5. Models used for detection

All the models use the same subset of the dataset for training and testing to ensure a fair comparison.

Gradient Boosted Decision Tree (GBDT) models are used to classify the tabular features, both static and dynamic. We use the implementation from LightGBM [46]. This type of models are commonly used in malware classification research using tabular features [4], [7]. Notably, a GBDT-based model using LightGBM, achieves state-of-the-art performance on EMBER features when used with its default parameters [4].

Graph Neural Networks (GNNs) are a popular choice for performing classification on graph-based features, as supported by existing literature [3], [11]. They work by iteratively updating node embeddings through neighborhood aggregation, refining each node's representation at every step. After several iterations, a readout function compiles these embeddings into a graph-level representation for classification. Various GNN architectures exist, each differing in the aggregation step.

The model used to classify our static graph-based features is proposed by MalGraph [10], a recently introduced model to classify hierarchical graphs composed of FCGs and CFGs. This hierarchical GNN architecture comprises three layers: an intra-function graph encoding layer, an inter-function graph encoding layer, and a prediction layer. Both graph encoding layers consist of multiple layers of GraphSAGE [47] and a max-pooling function. The prediction layer comprises three multilayer perceptrons and a sigmoid function as the last activation layer.

For the dynamic graph-based representation, SCDGs, we adopt the GINE model, following the approach of Hu *et al.* [26]. This extension of the GIN architecture [48] gives meaning to graphs with different edge types by incorporating edge features into the message aggregation process. Prior studies show the effectiveness of GIN-based architectures in ML-based malware detection [3], [49]. In particular, the GINE variant has shown strong performance on this task [9].

3.6. Robustness Evaluation

For our first robustness analysis, we evaluate the impact of packing on the four feature representations considered in our work. We use the `UPX` packer [50] version 4.0.2 with default parameters to pack all of the samples from the test set issued from the **IID** dataset. We then evaluate the ML models on the features extracted from this dataset of packed samples. The `UPX` packer is an executable file compressor. The program's executable code is compressed using UCL [51]. This packer significantly alters a PE file by compressing its code and data sections, modifying the section headers, redirecting the entry point

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

to a decompression stub, and minimizing the import table to only essential loader functions. These changes increase the file's entropy and obscure its original structure, making static analysis harder.

Afterward, we perform an adversarial attack using MAB-malware [31], an open-source reinforcement learning framework. We use this tool in its default configuration, except for the targeted model, which we define as the model we trained on the EMBER features extracted from our **IID** training set. We only apply this attack to the static feature representations (tabular and graph-based) as it only affects the static properties of the executable and not its behavior. Moreover, we only apply the attack to malware samples that have been correctly detected as malicious by the unhardened model.

4. Experiments & Results

This section presents the results of our evaluation of tabular and graph-based feature representations for malware detection. We first discuss the feature extraction process in Section 4.1. We compare the success rate and the computational cost for the extraction of tabular and graph-based representations. In Section 4.2, we present the configuration of the models used in the experiments. Then, we compare the detection scores of ML models trained on the different feature representations in Section 4.3. Finally, we assess robustness against packing and adversarial attacks in Sections 4.4 and 4.5, respectively.

Figure 2 offers a summary of which datasets are used in which experiments. Specifically, the first three experiments (detection, robustness to packing, and robustness to adversarial examples) are performed on a model trained on the original **IID** training set.

Testing for the detection experiment is performed on the original **IID** and **OOD** test sets. For the packing robustness experiment, the model is tested on the packed version of the **IID** test set. For the adversarial study, the same model is tested against the adversarial malware created from the **IID** and **OOD** test sets.

Finally, we evaluate the impact of adversarial retraining using a model trained on the original **IID** training set augmented with the adversarial malware created from this training set. The testing is done in two phases: First, we test the hardened model on the original **IID** and **OOD** test sets. Then, we test it against the adversarial malware originating from the **IID** and **OOD** test sets, but only those that initially evaded the unhardened model.

4.1. Feature Extraction

The extraction of the features has been performed on a machine with a 12-core 12th Gen Intel Core i7-1255U and 16 GB of RAM, running Ubuntu 24.04.

Table 2 shows the extraction success rates for each feature representation across the four data sources. Static features are extracted with more success than dynamic ones, and failures are more common in malware.

The static tabular features (EMBER) show the highest extraction success, with a rate close to 100%. The static graph-based features extraction with IDA Pro yields an acceptable success rate, reaching above 90% for all four dataset sources. The reasons for the failed extractions are

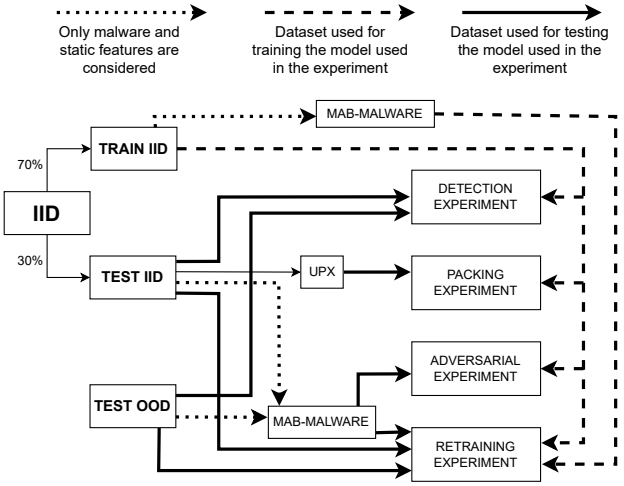

Figure 2. Summary of the datasets used in each experiment

TABLE 2. SUCCESS RATE OF THE FEATURE EXTRACTION PROCESS ON ALL FOUR DATASETS

Dataset	Static		Dynamic		Combined
	tabular	graph	tabular	graph	
BODMAS	100	91.3	83.5	84.3	76.8
Chocolatey	99.8	99.2	81.9	83.0	80.7
MalwareBazaar	100	98.1	95.1	95.3	94.0
Portable Freeware	100	96.5	97.0	97.7	93.6

either a graph that was too small (graphs with only one function of one block were considered unsuccessful) or an analysis that took more than 900 seconds, as would happen for samples with large code sections.

Regarding the features extracted with dynamic analysis, the BODMAS and Chocolatey datasets were the least successful with rates between 80% and 85%. The other two dataset sources have success rate above 95%. The tabular feature extraction was slightly less successful than for the graph-based features. The main cause of failure comes from the dynamic execution in the cuckoo sandbox. A significant portion of the samples do not detonate in the virtual environment. It can be due to several issues such as binary incompatibility, anti-VM techniques, missing dependencies for the binary, missing command line argument, etc. The lower success rate for the tabular features originates from execution reports that did not contain an execution summary. This summary was used for the tabular features but not for the graph-based features. Causes of failures for the graph-based dynamic features (other than a failed execution) include graphs that did not have at least one edge and graph extractions that lasted for more than 900 seconds.

For BODMAS malware families, CFG and FCG extraction with IDA Pro was least successful for autoit, sfone, sillyp2p, and wacatac (around 75% success). Dynamic analysis failed entirely on sillyp2p and small, and was only partially effective on musecador and wacatac (approximately 50%). Detailed success rates per family are provided in Appendix B.

Table 3 lists the extraction time distribution quartiles for the four feature representations. We can observe that the graph-based features take more time (approximately 10 times slower) to be extracted than the tabular features. This is to be expected as the graph-based features are more

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

TABLE 3. TIME REQUIRED IN SECONDS, PER SAMPLE, TO EXTRACT EACH TYPE OF FEATURE REPRESENTATION

	Static		Dynamic	
	Tabular	Graph	Tabular	Graph
First quartile	0.012	0.122	0.003	0.011
Median	0.027	0.224	0.031	0.141
Third quartile	0.128	1.070	0.082	1.900

complex than the tabular features. We do not consider the time needed to create the execution reports with the cuckoo sandbox. It means that an additional 3 to 5 minutes must be added for the processing of each sample with dynamic analysis. This aligns with past observations [29], which suggest that dynamic features should ideally not be used to classify every sample but only those for which the results with the static features are not conclusive.

4.2. Model configurations

This subsection describes the machine learning models and configurations used in our experiments. We specify whether hyper-parameters were adopted from the original implementations or tuned specifically for this study.

LightGBM is used for both static and dynamic tabular features. All hyper-parameters are set to their default values as defined by the LightGBM version used [52].

Malgraph is used for static graph-based features, following the configuration and hyper-parameters reported by the original authors [10], [53]. It is based on a GraphSAGE GNN architecture and uses a max pooling function. We report the parameter values in Appendix C.1.

GINE for SCDGs is the only model for which we tune the hyper-parameters. In this paper, we tune the number of layers, batch size, and hidden dimensions of the model to optimize its performance. We use the Adam optimizer, the NLLLoss criterion, and a CosineAnnealingLR scheduler for the learning rate [54]. We tune the hyper-parameter via grid search, using validation balanced accuracy and loss as selection criteria.

As already mentioned, 70% of the dataset is used for training, and the remaining 30% is used for testing. The training set is further split 80-20% to separate training and validation subsets. We evaluate hyper-parameter combinations via 5-fold cross-validation, with early stopping. After selecting the optimal combination, the model is retrained on the full training set for 60 epochs and then evaluated on the test set. The tested hyper-parameter values are detailed in Appendix C.2.

Regarding feature encoding, SCDGs are converted into PyTorch Geometric Data objects [55]. The node features are represented as 1-dimensional tensors encoding function names. Given the four possible edge types, the edge features are one-hot encoded as binary vectors, where each vector component indicates the presence (1) or absence (0) of the respective edge type between two nodes.

4.3. Malware Detection

In this subsection we present the results of the detection experiment for the four feature representations on the **IID** and **OOD** test sets.

TABLE 4. DETECTION PERFORMANCE FOR ALL FEATURE REPRESENTATIONS, ON THE **IID** AND **OOD** TEST SETS

		Static		Dynamic	
		Tabular	Graph	Tabular	Graph
IID Test set	**Accuracy**	**0.997***	0.987	0.995*	0.989
	Recall	**0.994***	0.993	0.993	**0.994***
	Precision	**0.999***	0.980	0.997*	0.984
	F1 score	**0.997***	0.987	0.995*	0.989
OOD Test set	**Accuracy**	**0.764***	0.762	0.705*	0.645
	Recall	0.537	**0.539***	0.414*	0.311
	Precision	0.984*	0.973	**0.991***	0.936
	F1 score	**0.695***	0.693	0.584*	0.467

*: best between tabular and graph; **Bold**: best overall.

Table 4 reports the malware detection performance across different feature representations on both the **IID** and **OOD** test sets. Static tabular features achieve the highest scores across all metrics on the **IID** test set, while dynamic graph features reach the best recall. On the **OOD** test set, static tabular features also lead in most metrics, though static graph features achieve higher recall. Overall, static representations outperform dynamic ones, and tabular features tend to outperform graph-based ones.

Static vs Dynamic feature representations. This experiment shows that models with static features outperform those with dynamic ones. This complements Dambra *et al.* [7], who highlight that the sparsity of dynamic features hinders their effectiveness.

We also observe that static features (tabular and graph) generally outperform dynamic features on the **OOD** test set. This may be due to the nature of the information they capture. Dynamic features reflect one execution's behavior, which can vary significantly across malware families, while static features provide a broader view of the binary, not at a specific execution. Additionally, dynamic tabular features tend to be sparse, further impacting generalization. Another possible explanation is that cleanware exhibits more diverse behaviors than malware, which are often clustered into a limited number of families. As a result, dynamic models may misclassify novel behaviors as cleanware due to their more significant variability, leading to lower recall on the dynamic side.

4.4. Impact of Packing

We evaluate the models trained previously (Table 4) on the packed versions of the samples in the **IID** test set. We pack the malware and cleanware binaries using `UPX` [50]. Not all samples could be packed which lead us to use a subset of the original **IID** test set. We end up using 32% of the original test set from BODMAS and 22% of the test set from Chocolatey.

TABLE 5. IMPACT OF PACKING, USING THE PACKED **IID** TEST SET

	Static		Dynamic	
	Tabular	Graph	Tabular	Graph
Balanced Acc.	0.947	0.528	0.927	0.732
Recall	0.963	1.000	0.861	0.866
Precision	0.953	0.607	0.995	0.759
F1 score	0.958	0.755	0.923	0.809

Table 5 presents the performance of the models on the packed **IID** test set. It shows that tabular features are less impacted by packing compared to graph-based representations. The static graph-based representation reaches

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

a recall of 100%, but with a notable drop in precision, indicating a high number of false positives. In contrast, graph-based representations are more sensitive to structural changes introduced by unpacking routines, which alter the graph's topology, making the presence of the additional packer more noticeable than in tabular features.

As for dynamic features, graphs again show lower robustness than tabular features, though with a smaller accuracy drop compared to static graphs. While dynamic analysis is generally expected to handle packing better, the fixed two-minute analysis timeout likely impacts the observed behavior and the information captured, due to the additional unpacking routine.

4.5. Adversarial Attacks & Retraining

Table 6 shows the results of the attack performed with `Mab-Malware` on the two static feature representations using the **IID** and **OOD** test sets. We consider three evaluation metrics. The metrics considered are the Attack Success Rate (ASR), which is the percentage of adversarial samples that evade detection from the unhardened model, along with the retrained model's accuracy on the original dataset (clean accuracy, **C. Acc**) and the retrained model's accuracy on adversarial samples (robust accuracy, **R. Acc**). Before retraining, we observe that `Mab-Malware` is highly effective on EMBER features (ASR close to 100%), but ineffective on the static graph-based representation (ASR close to 0%) across both datasets. On the **OOD** test set, only six samples evaded the MalGraph model, all using a "Section Add" attack, where benign functions were inserted, altering the graph structure and causing misclassification. This is expected, as the attack targets the EMBER model, and most modifications do not affect graph-based representations.

To perform retraining, we augment the original training set with adversarial malware samples created from the training set with `Mab-Malware`, resulting in an imbalanced training set with approximately twice as many malware than cleanware. The hardened model is trained from scratch on this new dataset. After adversarial retraining, the clean accuracy for tabular features remains high on the **IID** test set but drops significantly on the **OOD** test set, even compared to pre-retraining results (cf. Table 4). In contrast, graph-based features maintained high clean accuracy on the **IID** test set and achieved 0.795 clean accuracy on the original **OOD** test set, suggesting that retraining helped the model generalize better. The robust accuracy of the tabular features model remains high on the adv. **IID** test set but is very low on the adv. **OOD** test set. Since the attack with the adv. **IID** test set on the MalGraph model was ineffective, we cannot measure the robust accuracy of the retraining with this dataset. However, for the graph-based representations of the adv. **OOD** test set, we see that, after retraining, only one out of the six adversarial samples remains misclassified.

TABLE 6. ADVERSARIAL ATTACK AND RETRAINING RESULTS

	IID test set		**OOD test set**	
	Tabular	Graph	Tabular	Graph
ASR	0.991	0.000	0.984	0.014
C. Acc	0.996	0.990	0.532	0.795
R. Acc	0.999	/	0.106	0.833

5. Discussion

In this section we discuss our results to answer the different research questions (RQ), and add more insights.

For RQ1, the results in Table 4 indicate that tabular features slightly outperform graph-based representations, using both static and dynamic analysis. We also see that tabular features tend to generalize better to an out-of-distribution dataset. In Appendix A, we evaluate the static and dynamic representations separately and include in each datasets the samples that failed in the dynamic or static analysis, respectively. We find that the scores on the static features drop significantly after the additions of the samples that failed the dynamic analysis. It is likely due to the addition of two new malware families that were not present in the training set. The scores of detection for the dynamic features are mostly unaffected. They even improve a little on the **OOD** test set for the tabular dynamic features.

Table 5 shows that models using tabular features perform better against packing than the ones using graph-based representations, as the code of the packer mostly impacts the structure of graph-based features, answering RQ2. The observed 100% recall and drop in precision for the static graph-based representation suggests that packed samples are systematically classified as malware. This aligns with Aghakhani *et al.* [5], who highlight that some models tends to classify packed samples as malware due to the packer. We extend this by showing that the impact also depends on the feature representation. Tabular features show greater robustness despite a slight drop in accuracy and precision. As noted by Dambra *et al.* [7], packing mainly affects other static analyses like reverse engineering, but it has less impact on ML classifiers using tabular static features. A comparison of the datasets before and after packing shows that most EMBER features are affected, with only 3 unchanged features actually used by the LightGBM model. This suggests that packing alters the features without preventing the model from distinguishing cleanware and malware.

Both of these results support the use of tabular features over graph-based representations in malware detection. However these results also depend on the chosen tabular features and models. We use benchmark tabular features, well grounded in the literature [4], [7], but the results might vary with different features. This highlights the importance of using benchmark datasets and features for evaluating emerging approaches. In order to answer these RQs more comprehensively, we would have to add more representations in the study. For tabular representations, we could investigate the static tabular features introduced by Aghakhani *et al.* [5] which comprise nine families of features extracted from the PE structure, the program's assembly, and the raw bytes of the binary. Other static feature representations that would be worth exploring are the simple yet effective Control Flow Graph [56], [57], raw bytes (e.g. as used by MalConv [14]) or image-based representations [58], [59]. In the case of dynamic features, we can explore API call sequences [42], API Call Graphs [12] or even bypassing the whole feature engineering process by directly feeding the sandbox execution report to the ML model [34]. Moreover, we could combine these representations and use them with ensemble learning

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

or in a multi-modal setting [60], [61].

According to our adversarial setting, the answer to RQ3 is that static tabular features are more vulnerable than static graph-based representations to the considered adversarial attack, with and without retraining, as shown in Table 6. The scores on the original **OOD** dataset after retraining show a severe drop for the tabular features. It can be explained by a well-known phenomenon [62] which shows that, when hardening is performed on a certain kind of distributional shift (the adversarialized **IID** dataset in our case), the hardened model can become less effective on other kinds of distributions shifts (the **OOD** dataset here). The graph-based model was not affected by this phenomenon, as the features were not affected by the attack. It implies that the **IID** training set was distributed similarly to its adversarialized version. For a more definitive answer to this last RQ, more attacks should be explored, targeting more diverse properties of the binaries to impact each feature representation.

Attacks that should be explored include the SecML Malware framework proposed by Demetrio *et al.* [30], which implements several attacks targeting PE files. This framework was tested against MalConv [14] and for some attacks, against EMBER. For the graph-based static features, recent work [63], [64] propose attacks against Control Flow Graphs. It would be worth studying the impact of such attacks on other features such as the graphs proposed by MalGraph [10] and the various tabular features.

Regarding the attacks based on behavioral features, several directions have been studied in the literature. For the SCDGs, Ming *et al.* [65] implemented a replacement attack to harm the effectiveness of malware clustering systems. Their modifications, although realistic, can only be applied to source code. In the absence of source code, modifying malware to create variants capable of evading behavior-based detection systems becomes significantly more challenging. Digregorio *et al.* [43] made a step in that direction by proposing Tarallo, a framework that modifies binary files to incorporate benign API calls in the API sequence. Tsingenopoulos *et al.* [34] propose modifying features instead of the binary, while preserving functionality. They focus on CAPEv2 execution report summaries, altering them by adding or replacing elements. Their approach could extend to similar representations.

6. Limitations and Perspectives

This section reports limitations present in our methodology in order to guide future research.

The GINE model using SCDGs shows the largest accuracy drop among the four models on the **OOD** dataset. This may be due to our use of a more straightforward approach compared to more advanced GNN architectures and graph embeddings in the literature [9], [12]. For example, DawnGNN [12] uses BERT-based [66] API embeddings, while DMalNet [9] uses Word2Vec [67] embeddings for API calls and combines GINE and GAT. In contrast, our approach uses one-hot encoding for edge features and numerical IDs for node labels. This reduces the computational cost of graph extraction. While more complex embeddings could slightly improve generalization or robustness, they would add computational overhead, potentially outweighing their benefits, especially since tabular features are faster to extract and yield better ML performance.

Our datasets, like most in this research area, are not free from bias. The malware and cleanware distributions do not necessarily reflect real-world spatial distributions, and our benign dataset lacks reliable "first seen" timestamps. We partially mitigate the temporal bias by performing the temporal split on the malware samples. We also evaluate generalization using an out-of-distribution dataset, as an example of how different distributions can impact performance. Future work should explore more representative, time-aware cleanware benchmark datasets to better align with real-world deployment scenarios and allow better concept drift analysis.

Including alternative representations such as images, sequence-based, or text-based formats like raw bytes and behavioral reports, could offer further insights into the trade-offs to consider when choosing a representation.

Additionally, we could expand the adversarial study to target different aspects of each feature representation and thus, cover a wider part of the domain space when retraining. Indeed, the adversarial attack used in this work was designed to target models using tabular features, making it ineffective against static graph-based representations. As a result, the relative robustness of graph-based models remains uncertain. Evaluating attacks for static graph-based models would offer more insights into the impact of attacks and their transferability across representations.

Besides, we did not explore adversarial attacks on dynamic feature representations, as they present unique challenges due to their reliance on runtime behavior. The main challenge is to design a realistic attack without corrupting or altering the malicious behavior of the sample. Developing a problem-space attack targeting Cuckoo reports or SCDGs would be a significant undertaking, which is beyond the scope of the present work. Nevertheless, it remains a promising direction for future research.

Lastly, we did not conduct statistical significance testing (e.g. confidence intervals or variance estimation via resampling), as the use of a temporal split limits our ability to produce representative resampled subsets. While our results are consistent, the addition of statistical significance testing could strengthen the reliability of our results.

7. Conclusion

Our work studies the effectiveness of feature representations for malware detection across different settings. Surprisingly, despite the common belief that more complex features give better results, tabular features perform remarkably well, even on out-of-distribution data. Packing impacts all feature types, with graph-based features being most affected and static tabular features the most resilient. In adversarial settings, the studied attack significantly degraded static tabular features but had no effect on static graph-based features. Adversarial retraining led to a performance drop on out-of-distribution data for the static tabular features, likely due to the attack's limited diversity.

Acknowledgments

This research is supported by the Walloon region's CyberExcellence program (grant #2110186).

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

References

[1] ENISA, "ENISA threat landscape report," 2024, accessed: Avr. 14, 2025. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

[2] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, 2019.

[3] V. Malhotra, K. Potika, and M. Stamp, "A comparison of graph neural networks for malware classification," *Journal of Computer Virology and Hacking Techniques*, 2024.

[4] H. S. Anderson and P. Roth, "EMBER: an open dataset for training static PE malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[5] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[6] D. Rabadi and S. G. Teo, "Advanced windows methods on malware detection and classification," in *36th Annual Computer Security Applications Conference (ACSAC)*, 2020.

[7] S. Dambra, Y. Han, S. Aonzo, P. Kotzias, A. Vitale, J. Caballero, D. Balzarotti, and L. Bilge, "Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[8] C.-H. Bertrand Van Ouytsel and A. Legay, "Malware analysis with symbolic execution and graph kernel," in *Nordic Conference on Secure IT Systems (NordSec)*, 2022.

[9] C. Li, Z. Cheng, H. Zhu, L. Wang, Q. Lv, Y. Wang, N. Li, and D. Sun, "DMalNet: Dynamic malware analysis based on API feature engineering and graph learning," *Computers & Security*, 2022.

[10] X. Ling, L. Wu, W. Deng, Z. Qu, J. Zhang, S. Zhang, T. Ma, B. Wang, C. Wu, and S. Ji, "Malgraph: Hierarchical graph neural networks for robust windows malware detection," in *IEEE INFOCOM Conference on Computer Communications*, 2022.

[11] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, "A survey on malware detection with graph representation learning," *ACM Computing Surveys*, 2024.

[12] P. Feng, L. Gai, L. Yang, Q. Wang, T. Li, N. Xi, and J. Ma, "DawnGNN: Documentation augmented windows malware detection using graph neural network," *Computers & Security*, 2024.

[13] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-third Annual Computer Security Applications Conference (ACSAC)*, 2007.

[14] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.

[15] R. Thomas, "LIEF - Library to Instrument Executable Formats," accessed: Avr. 14, 2025. [Online]. Available: https://lief.quarkslab.com/

[16] E. Carrera, "pefile," accessed: Avr. 14, 2025. [Online]. Available: https://github.com/erocarrera/pefile

[17] Hex-Rays, "IDA Pro: Interactive Disassembler," accessed: Feb. 07, 2025. [Online]. Available: https://hex-rays.com/ida-pro

[18] "Cuckoo sandbox," accessed: Avr. 14, 2025. [Online]. Available: https://github.com/cuckoosandbox/cuckoo

[19] K. O'Reilly and A. Brukhovetskyy, "CAPE: Malware Configuration And Payload Extraction," accessed: Avr. 14, 2025. [Online]. Available: https://github.com/kevoreilly/CAPEv2

[20] "QEMU, A generic and open source machine emulator and virtualizer," accessed: Feb. 11, 2025. [Online]. Available: https://www.qemu.org/

[21] V. Vouvoutsis, F. Casino, and C. Patsakis, "On the effectiveness of binary emulation in malware classification," *Journal of Information Security and Applications (JISA)*, 2022.

[22] "DynamoRIO: Dynamic Instrumentation Tool Platform," accessed: Feb. 11, 2025. [Online]. Available: https://dynamorio.org/

[23] O. A. V. Ravnås, "Frida: Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers," accessed: Feb. 11, 2025. [Online]. Available: https://frida.re/

[24] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[25] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, 2017.

[26] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations (ICLR)*, 2020.

[27] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *The 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007.

[28] D. Trizna, "Quo Vadis: hybrid machine learning meta-model based on contextual and behavioral malware representations," in *15th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2022.

[29] A. Ponte, D. Trizna, L. Demetrio, B. Biggio, I. T. Ogbu, and F. Roli, "SLIFER: Investigating performance and robustness of malware detection pipelines," *Computers & Security*, 2025.

[30] L. Demetrio and B. Biggio, "Secml-malware: Pentesting windows malware classifiers with adversarial exemples in python," *arXiv preprint arXiv:2104.12848*, 2021.

[31] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, 2022.

[32] L. Jia, Y. Yang, J. Li, H. Ding, J. Li, T. Yuan, L. Liu, and Z. Jiang, "MTMG: A framework for generating adversarial examples targeting multiple learning-based malware detection systems," in *Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2023.

[33] H. Bostani, J. Cortellazzi, D. Arp, F. Pierazzi, V. Moonsamy, and L. Cavallaro, "On the effectiveness of adversarial training on malware classifiers," *arXiv preprint arXiv:2412.18218*, 2024.

[34] I. Tsingenopoulos, J. Cortellazzi, B. Bošanskỳ, S. Aonzo, D. Preuveneers, W. Joosen, F. Pierazzi, and L. Cavallaro, "How to train your antivirus: RL-based hardening through the problem space," in *27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2024.

[35] S. Aonzo, Y. Han, A. Mantovani, and D. Balzarotti, "Humans vs. machines in malware classification," in *32nd USENIX Security Symposium*, 2023.

[36] L. Maffia, D. Nisi, P. Kotzias, G. Lagorio, S. Aonzo, and D. Balzarotti, "Longitudinal study of the prevalence of malware evasive techniques," *arXiv preprint arXiv:2112.11289*, 2021.

[37] *Chocolatey: The Package Manager for Windows*, accessed: Feb. 11, 2025. [Online]. Available: https://chocolatey.org/

[38] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An open dataset for learning based temporal analysis of PE malware," in *4th Deep Learning and Security Workshop (DLS)*, 2021.

[39] "VirusTotal," accessed: Mar. 03, 2025. [Online]. Available: https://www.virustotal.com

[40] abuse.ch, *MalwareBazaar*, accessed: Feb. 11, 2025. [Online]. Available: https://bazaar.abuse.ch/

[41] C. Sébastien, "Portable freeware dataset," accessed: Avr. 14, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.2528209

[42] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, and Y. Qiao, "A novel deep framework for dynamic malware detection based on API sequence intrinsic features," *Computers & Security*, 2022.

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

[43] G. Digregorio, S. Maccarrone, M. D'Onghia, L. Gallo, M. Carminati, M. Polino, and S. Zanero, "Tarallo: Evading behavioral malware detectors in the problem space," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2024.

[44] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does every second count? time-based evolution of malware behavior in sandboxes," in *Network and Distributed Systems Security Symposium (NDSS)*, 2021.

[45] CERT-EE, "Cuckoo3 - malware analysis tool," accessed: Avr. 14, 2025. [Online]. Available: https://github.com/cert-ee/cuckoo3

[46] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems (NeurIPS)*, 2017.

[47] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems (NeurIPS)*, 2017.

[48] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[49] F. Z. Bolun Wu, Yuanhang Xu, "Malware classification by learning semantic and structural features of control flow graphs," in *IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2021.

[50] *UPX, the Ultimate Packer for eXecutables*, accessed: Feb. 11, 2025. [Online]. Available: https://upx.github.io/

[51] "The UCL Compression Library," accessed: Apr. 04, 2025. [Online]. Available: https://www.oberhumer.com/opensource/ucl/

[52] "LightGBM, version 4.5.0," accessed: Apr. 07, 2025. [Online]. Available: https://lightgbm.readthedocs.io/en/v4.5.0/

[53] "Malgraph," accessed: Apr. 07, 2025. [Online]. Available: https://github.com/ryderling/MalGraph

[54] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations (ICLR)*, 2017.

[55] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds (RLGM)*, 2019.

[56] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *12th USENIX Security Symposium*, 2003.

[57] M. Someya, Y. Otsubo, and A. Otsuka, "FCGAT: Interpretable malware classification method using function call graph and attention mechanism," in *Network and Distributed Systems Security Symposium (NDSS)*, 2023.

[58] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Bao, "Byte-level malware classification based on markov images and deep learning," *Computers & Security*, 2020.

[59] X. Huang, L. Ma, W. Yang, and Y. Zhong, "A method for windows malware detection based on deep learning," *Journal of Signal Processing Systems*, 2021.

[60] D. Gibert, C. Mateu, and J. Planes, "HYDRA: A multimodal deep learning framework for malware classification," *Computers & Security*, 2020.

[61] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel android malware detection system using ensemble learning methods," *Computers & Security*, 2017.

[62] J. Gilmer and D. Hendrycks, "A discussion of'adversarial examples are not bugs, they are features': Adversarial example researchers need to expand what is meant by'robustness'," *Distill*, 2019.

[63] X. Ling, Z. Wu, B. Wang, W. Deng, J. Wu, S. Ji, T. Luo, and Y. Wu, "A wolf in sheep's clothing: practical black-box adversarial attacks for evading learning-based windows malware detection in the wild," in *33rd USENIX Security Symposium*, 2024.

[64] H. Peng, Z. Yu, D. Zhao, Z. Ding, J. Yang, B. Zhang, J. Han, X. Zhang, S. Ji, and M. Zhong, "Evading control flow graph based gnn malware detectors via active opcode insertion method with maliciousness preserving," *Scientific Reports*, 2025.

[65] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao, "Impeding behavior-based malware analysis via replacement attacks to malware specifications," *Journal of Computer Virology and Hacking Techniques*, 2017.

[66] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Conference of the North American chapter of the association for computational linguistics (NAACL)*, 2019.

[67] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[68] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations (ICLR)*, 2019.

Appendix A.

The results of the detection experiment using a separate dataset for static and dynamic based features can be found in Table 7. The main difference with the experiment in Section 4.3 is that the test sets used for the experiment on the static features also contain samples that did not succeed in the dynamic analysis and conversely, the test sets used for the experiment on the dynamic features contain samples that did not pass the static features extraction. The models used are the same as the ones for the original experiment

The results for the **IID** test set show that the score for the static features are much lower compared to the original experiment, especially the recall. This is likely due to the fact that two malware families (small and sillyp2p) were discarded from the original dataset because of their failure to pass the dynamic analysis. These families were not in the training set and are then logically wrongly detected by the models. The score for the dynamic features are essentially unaffected by the addition of the samples that failed in the static analysis. This can be explained by the fact that there aren't many samples added to the test set and that the family of these samples was already represented in the original test set and in the training set.

The results on the **OOD** test set are mostly unaffected when compared to the original experiment. Once again likely due to the small amount of samples added to the test set. Nonetheless, we observe that the graph-based static features are even less affected than the tabular static features and thus surpass the later in most scores.

TABLE 7. DETECTION PERFORMANCE FOR ALL FEATURE REPRESENTATIONS, ON THE **IID** AND **OOD** TEST SETS WHEN SELECTING A COMMON DATASET FOR STATIC AND DYNAMIC BASED FEATURES SEPARATELY

		Static Only		**Dynamic Only**	
		Tabular	Graph	Tabular	Graph
IID Dataset	**Balanced Acc.**	**0.927**	0.895	**0.994**	0.984
	Recall	**0.859**	0.832	**0.992**	**0.992**
	Precision	**0.996**	0.960	**0.998**	0.984
	F1 score	**0.923**	0.891	**0.995**	0.988
OOD Dataset	**Balanced Acc.**	0.756	**0.760**	**0.706**	0.638
	Recall	0.521	**0.537**	**0.415**	0.317
	Precision	**0.980**	0.970	**0.991**	0.884
	F1 score	0.680	**0.692**	**0.585**	0.466

Bold: best between tabular and graph.

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.

TABLE 8. MALWARE FAMILY DISTRIBUTION OF THE SAMPLES EXTRACTED FROM MALWAREBAZAAR FOR THE **OOD** DATASET

	AZORult	Adware.Generic	AgentTesla	AilurophileStealer	Amadey	AsyncRAT	AveMariaRAT	BazaLoader	Berbew	Bitter	BlackMatter	Blackmoon	BlihanStealer	Chaos	ConnectWise	Cosmu	CredentialFlusher	CryptBot	DCRat	DarkCloud	DarkTortilla	DiscordTokenStealer
Initial #	1	1	26	1	13	17	1	1	8	1	3	4	1	3	4	1	50	17	8	3	2	1
Final #	1	1	24	1	13	12	1	1	8	1	3	1	1	2	4	1	49	16	8	3	1	1

	DragonForce	Formbook	GCleaner	Gh0stRAT	GuLoader	HawkEye	Healer	Heodo	HexonStealer	INC	KoiLoader	LockBit	LodaRAT	Loki	Lu	LummaStealer	MassLogger	MeduzaStealer	Mimic	NanoCore	Neshta	NetSupport
Initial #	3	81	1	2	32	1	95	1	1	1	1	1	4	1	143	14	1	2	1	4	1	2
Final #	3	79	1	2	32	1	94	0	1	1	1	1	4	1	132	14	1	2	1	4	1	2

	Njrat	OrcusRAT	Phobos	PureCrypter	PureLogStealer	QuasarRAT	RedLineStealer	RemcosRAT	STRRAT	SakulaRAT	ShipUp	SilverFox	SnakeKeylogger	Stealc	StormKitty	VIPKeylogger	Venik	Vidar	Wiper	Worm.Ramnit	Worm.m0yv	XWorm	YoungLotus	No family detected	total
Initial #	1	1	4	2	4	33	39	1	3	1	1	18	99	1	11	5	3	1	1	1	1	1	62	855	
Final #	1	1	4	2	4	31	39	1	2	1	1	18	97	1	11	0	2	1	1	0	1	1	52	804	

Appendix B.

Tables 9 and 8 present the family distribution for the subset of BODMAS that we considered as well as the samples that we downloaded from MalwareBazaar. We give the initial number of samples as well as the number of samples for which we could extract the four feature representations (in the ”final” column).

For our subset of the BODMAS dataset, we also provide, for each family, the number of samples that passed the analysis with IDA Pro and that detonated during the dynamic execution in the cuckoo sandbox. We did not include a column for the extraction of the EMBER features as it reached a 100% success rate.

TABLE 9. MALWARE FAMILY DISTRIBUTION OF THE SUBSET OF BODMAS CONSIDERED IN OUR EXPERIMENTS

family	initial	IDA Pro analysis	sandbox execution	final
autoit	500	377	485	307
benjamin	500	500	500	500
berbew	500	500	500	499
ceeinject	500	424	500	413
delf	500	499	499	497
dinwod	500	500	499	499
drolnux	500	500	500	500
gandcrab	500	500	497	488
ganelp	500	500	500	500
gepys	500	493	500	492
mira	500	496	500	473
musecador	500	500	252	248
qukart	500	500	466	461
sfone	500	94	499	91
sillyp2p	500	376	0	0
small	500	497	0	0
unruy	500	500	498	498
upatre	500	489	500	489
wabot	500	500	500	494
wacatac	500	390	289	227
total	10,000	9,135	8,484	7,676

Appendix C.

C.1. Malgraph parameters

Table 10 reports the parameter values used for the Malgraph model in this study. For additional details, please refer to the original paper and github repository [10], [53].

TABLE 10. MALGRAPH TRAINING AND CONFIGURATION PARAMETERS

	Parameter
Training	
Max Epochs	10
Train Batch Size	16
Model	
GNN Type	GraphSAGE
Pool Type	Global Max Pool
Dropout Rate	0.2
Optimizer	
Name	AdamW [68]
Learning Rate	1×10^{-3}
Weight Decay	1×10^{-5}
Learning Anneal	1.1

C.2. GINE hyper-parameters tuning

Table 11 presents the detailed specifications of the tested hyper-parameters for the GINE model on SCDGs, along with the best combination (in bold) across the 5 folds.

TABLE 11. HYPERPARAMETERS TESTED FOR THE GINE MODEL.

Hyperparameter	Grid Values
Number of layers	[3, **4**, 5, 6]
Hidden dimensions	[32, 64, **128**, 256]
Batch size	[8, 16, 32, **64**, 128]

Authorized licensed use limited to: Univ Catholique de Louvain/UCL. Downloaded on September 30,2025 at 08:40:09 UTC from IEEE Xplore. Restrictions apply.