Detection Is Easy, Classification Is Hard: The Role of Dataset Design in Machine-Learning Malware Analysis

Evidence from Five Real-World Malware Datasets

Jesse Sabatini
College of Natural Sciences
Master of Science in Computer Science
University of Texas at Austin
jes7575@my.utexas.edu

***Abstract*—This case study examines how malware dataset design governs the performance of machine-learning detection and family classification. A single CatBoost pipeline is applied to five widely used datasets (EMBER2017_v2, EMBER2018, BIG2015, and BODMAS) and to the MalAPI2019 API-call collection. This workflow ensures that performance differences can be attributed to data characteristics rather than model changes. On the merged EMBER2017_v2 + EMBER2018 static PE dataset, binary detection achieves AUROC 0.998 and AUPRC 0.998, indicating that benign and malicious binaries are cleanly separable under engineered static features. Family-level results are far more heterogeneous. BIG2015, which provides a curated taxonomy and relatively balanced class distribution, reaches a macro F_1 of 0.98. MalAPI2019, based on TF–IDF representations of API-call sequences, attains a macro F_1 of 0.38, reflecting behavioral overlap and sparse traces. BODMAS performs worst (macro F_1 0.07) under severe class imbalance and coarse, heterogeneous family labels. Taken together, the results show that modern tree-based models can reliably solve static detection when features and labels are well structured, while family classification is fundamentally constrained by class balance, taxonomy coherence, and feature-to-label alignment in the underlying datasets.**

***Index Terms*—malware detection, malware classification, machine learning, static analysis, dataset quality, feature engineering**

I. Introduction

Traditional signature-based malware defenses cannot keep pace with the scale and diversity of modern threats. Recent surveys describe how rapid family evolution, widespread packing, and increasingly complex obfuscation techniques reduce the effectiveness of manual triage and signature matching [1]–[3]. As a result, machine-learning methods have become central to automated malware analysis, where models learn structural or behavioral patterns associated with malicious activity. Static approaches rely on properties such as section entropy, opcode distributions, and import tables, while dynamic analysis captures runtime behavior through system calls, API sequences, and execution traces [2], [21], [26]. Hybrid systems combine both perspectives and can improve robustness against evasive malware [1], [5].

Across these techniques, prior work consistently shows that dataset design is a primary factor governing model reliability. Challenges documented in recent studies include inconsistent family taxonomies, heterogeneous labeling practices, class imbalance, and feature sets that do not align well with family definitions [5], [13], [21]. Benchmark datasets such as EMBER were created to provide large-scale and standardized static features [10], while other collections demonstrate how noise and structural inconsistencies can distort classification performance even when modern models are used [12], [13]. Comparative evaluations further indicate that variability in dataset structure often contributes more to performance differences than the choice of learning algorithm [25], [26].

This case study investigates how dataset characteristics influence the outcomes of machine-learning malware detection and malware family classification. Five widely used datasets are examined under a single CatBoost workflow to ensure that differences in performance arise from the data rather than from model configuration. The analysis focuses on class balance, taxonomy quality, and feature modality, and how these factors affect both binary detection and family-level prediction.

The goal of this study is to provide a clear, data-driven comparison of how real-world malware datasets behave under a consistent modeling pipeline. By evaluating detection and classification across static and behavioral feature modalities, the results highlight the dataset conditions that support reliable learning and those that limit the feasibility of accurate family classification.

II. Background and Dataset Overview

A. *Factors Affecting Dataset Reliability for Malware ML*

Machine-learning models for malware detection and classification depend on datasets that provide meaningful features, accurate labels, and class distributions that support generalizable decision boundaries. Prior work shows that these properties are not consistently satisfied in widely used malware datasets, which leads to performance variation caused by dataset construction rather than inherent model capability [4], [5], [7].

Static datasets rely on consistent extraction of PE metadata, byte frequency distributions, strings, and import patterns. When preprocessing is uniform, these features support stable binary detection. However, packing, obfuscation, or inconsistent extraction procedures can suppress informative data features and hinder family-level separation [2], [6]. Behavioral datasets based on API-call traces introduce their own difficulties. Execution traces may be incomplete, environment dependent, or highly sparse, which limits the reliability of behavioral indicators for distinguishing malware families [18], [19], [21]. Hybrid static and dynamic approaches alleviate some of these weaknesses but still depend on coherent data collection and high-quality labels [6], [22].

Label reliability is an additional challenge. Malware taxonomies differ between vendors, and naming conventions often overlap or conflict with one another. These inconsistencies blur family boundaries and complicate supervised learning [4], [5]. Severe class imbalance, which is common in family datasets, further biases models toward majority families and produces weak recall for minority classes [7], [18]. These structural factors help explain why machine-learning models often achieve strong binary detection performance but struggle with fine-grained family classification.

The datasets evaluated in this study exhibit these issues to varying degrees. They differ in feature modality, labeling practices, taxonomy granularity, and class distribution. These disparities allow for a systematic evaluation of how dataset structure affects model behavior under a consolidated training pipeline.

B. *Overview of Datasets Used in This Study*

The five datasets used in this work (EMBER2017_v2, EMBER2018, BIG2015, BODMAS, and MalAPI2019) were chosen because they represent distinct structural conditions relevant to malware ML research. They differ in feature representation, labeling quality, and sample balance, which enables controlled comparison of detection and classification performance. Table I summarizes their primary characteristics.

TABLE I
SUMMARY OF DATASETS EVALUATED IN THIS STUDY

Dataset	**Feature Type**	**Primary Task**	**Families**	**Notes**
EMBER2017_v2	Static (PE)	Detection	No	Engineered features; balanced benign and malicious data
EMBER2018	Static (PE)	Detection	No	Updated schema; high-quality static features
BIG2015	Static (PE)	Classification	Yes	Curated taxonomy; moderate class balance
BODMAS	Static (PE)	Classification	Yes	Severe imbalance; heterogeneous and coarse family labels
MalAPI2019	Behavioral (API calls)	Classification	Yes	Dynamic traces; sensitive to sparsity and noise

1) EMBER2017_v2 & EMBER2018: Static PE datasets such as EMBER2017_v2 and EMBER2018 contain engineered feature vectors extracted through a standardized pipeline that summarizes header layout, section statistics, entropy measurements, strings, and byte-level distributions [9]–[11]. These datasets do not include family labels and therefore serve as stable baselines for evaluating binary detection under high-quality static features.

2) BIG2015: BIG2015 provides a curated static dataset specifically designed for malware family classification. Its structured taxonomy, unpacked binaries, and moderate class balance contribute to lower label noise and more reliable family-level structure compared to large open-source collections [5], [7]. These properties make BIG2015 suitable for evaluating classification performance under favorable static conditions.

3) BODMAS: BODMAS differs substantially. It aggregates samples from multiple sources, mixes broad behavioral categories with campaign-level identifiers, and exhibits severe class imbalance. Packing and obfuscation are common and often reduce the information available in static PE features [4], [5]. These characteristics make BODMAS an appropriate dataset for studying the limitations of static family classification.

4) MalAPI2019: MalAPI2019 provides dynamic behavioral representations through API-call sequences derived from execution logs. Term frequency-inverse document frequency (TF-IDF) encoding captures high-level patterns such as registry access and network activity. However, sparsity, incomplete execution traces, and environment-dependent variability introduce noise that affects family-level discrimination [18], [19], [21]. Therefore, this dataset occupies a middle ground between the curated static features of BIG2015 and the heterogeneous static features of BODMAS.

Taken together, these datasets span a diverse set of structural conditions that influence model behavior in both binary detection and family classification. EMBER2017_v2 and EMBER2018 establish a high-quality baseline for static detection, while BIG2015, BODMAS, and MalAPI2019 represent varying degrees of classification difficulty that are directly examined in Section V.

III. Experimental Design and Methods

A unified experimental workflow was used for all datasets to isolate the effects of dataset structure from model behavior. The pipeline covers data preparation, feature construction, model training, evaluation, and explainability, and is applied identically across EMBER, BIG2015, BODMAS, and MalAPI2019. Dataset-specific preprocessing decisions, including the merging of EMBER2017_v2 with EMBER2018, the preservation of long-tail family distributions in BODMAS, and the sanitization and minimum-frequency filtering applied to MalAPI2019 API-call data, are detailed in the subsections that follow.

A. Overall Experimental Design

For each collection, the workflow consists of dataset loading, preprocessing, feature construction, model training and validation, final evaluation on a held-out test split, and post hoc explainability analysis. The same high-level sequence of operations is used for static PE feature vectors, byte-level histograms, and API-call representations, which supports controlled comparisons across heterogeneous feature spaces [1], [2], [4], [5], [7]. This standardized structure ensures that any performance differences observed across datasets arise from the structure of the data themselves rather than from variations in preprocessing, feature engineering, or model configuration.

A high-level overview of this end-to-end workflow is shown in Fig. 1. Raw data are first transformed into task-specific feature matrices, using the EMBER static PE feature schema for both EMBER and BODMAS samples [8], [10], engineered histogram features for BIG2015, and TF–IDF representations of API-call sequences for MalAPI2019 [15]–[18], [29]. These feature matrices are then passed to a common preprocessing stage that performs stratified train, validation, and test splits with fixed random seeds, and stores the resulting partitions in a standardized format used by all subsequent steps. The training stage fits supervised models on the training data with early stopping on the validation set, and the evaluation stage computes a shared set of metrics for each task, including area under the ROC curve, area under the precision-recall curve, and F_1 score for binary detection, and macro and weighted F_1 scores for multi-class family prediction.

Two prediction tasks are addressed in this study. The first task is binary malware detection, where the goal is to distinguish malicious from benign samples in the EMBER-derived dataset. The second task is malware family classification, where the goal is to assign each malicious sample to a specific family or class in BODMAS, BIG2015, and MalAPI2019. CatBoost serves as the primary model for both tasks, as gradient-boosted decision trees have demonstrated strong performance on tabular security data [1], [4], [19], [20], while logistic regression and support vector machines are included as classical baselines. This modeling framework, combined with dataset-specific decisions such as EMBER year merging, BODMAS family preservation, and MalAPI token sanitization, provides a controlled basis for analyzing how dataset design governs both detection performance and the feasibility of accurate family classification.

B. Dataset Construction and Feature Preparation

All experiments operate on standardized feature bundles that are derived from the raw datasets and stored in a common format. For each data collection, the original archives (PE files, byte streams, or API-call logs) are transformed into a matrix of numerical features X, an integer label vector y, and accompanying metadata that records feature names and class names. These bundles are written as compressed `.npz` files

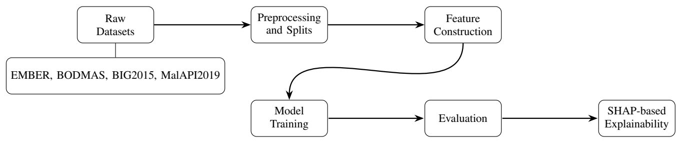

Fig. 1. Unified pipeline for dataset loading, preprocessing, feature construction, model training, evaluation, and SHAP-based explainability. The same workflow is applied to all datasets; only the underlying data and feature modalities differ.

and serve as the entry point for the shared preprocessing and training pipeline described in Sections III-A and III. Dataset-specific steps, such as the static PE feature extraction used for EMBER and BODMAS [8], [10], [21], [26], the byte-histogram construction applied to BIG2015, and the TF–IDF representation of MalAPI2019 API-call sequences [15]–[18], are detailed in Sections III-B1–III-B4.

1) EMBER-based Static PE Datasets: EMBER2017_v2 and EMBER2018 provide large-scale static PE datasets with engineered feature vectors designed for supervised malware detection [8], [10]. In this work, both years are converted into a consolidated feature representation using the official EMBER feature extractor. The extractor produces fixed-length vectors that summarize header metadata, section statistics, byte- and string-level histograms, and entropy measurements for each executable. The resulting train and test partitions from each year are concatenated, and samples with unlabeled or ambiguous targets (original label -1) are removed before further processing.

To form a single EMBER-derived compilation, the feature matrices and labels from EMBER2017_v2 and EMBER2018 are merged after verifying that their feature schemas and dimensionalities are identical. A preprocessing script optionally performs benign downsampling through a `max_benign_ratio` parameter that limits the ratio of benign to malicious samples in the merged dataset. This step mitigates extreme class imbalance when necessary but does not alter the feature space or the definition of the binary labels. After merging and any benign sample reduction, labels are mapped to $\{0, 1\}$ with class names *benign* and *malicious*, and the final feature bundle is stored as a compressed `.npz` file that is consumed by the downstream splitting, training, and evaluation pipeline.

2) BODMAS Static PE Dataset: The BODMAS collection is processed using the same static PE feature extraction pipeline applied to EMBER, but without modifying or collapsing the original family taxonomy. Each PE file is converted into a fixed-length feature vector that captures header metadata, section characteristics, entropy statistics, strings features, and byte histogram summaries, following conventions established in prior static-analysis work [6], [21], [26]. After extraction, all samples with missing or invalid family assignments are removed.

Unlike EMBER, BODMAS exhibits substantial long-tail class behavior, with many families represented by only a small number of samples. These families are deliberately preserved rather than filtered out, since one goal of this study is to evaluate how class imbalance and coarse taxonomy boundaries affect downstream classification performance. Family names are mapped to integer labels, and the complete feature matrix, label vector, and class-name map are stored as a final `.npz` feature bundle used by the training and evaluation pipeline.

3) BIG2015 Byte-Level Dataset: The BIG2015 dataset is processed by converting each binary sample into a normalized byte histogram representation. This representation, which counts the frequency of each byte value $\{0, \ldots, 255\}$, provides a lightweight approximation of structural characteristics without requiring full PE parsing. A custom extraction module reads each file, computes the histogram, scales it to unit length, and assigns the provided family label. Samples that cannot be decoded due to corruption or truncated file contents are discarded.

BIG2015 contains a relatively balanced distribution of families compared to BODMAS, and its taxonomy is known to be cleaner and less noisy. This structure provides a contrast case for evaluating family classification under more favorable dataset conditions. The resulting histogram feature matrix and label vector are stored in a `.npz` bundle consumed by the pipeline described in Section III-A.

4) MalAPI2019 API-Call Dataset: MalAPI2019 contains dynamic API-call sequences derived from execution logs. In this work, each sequence is transformed into a sparse TF–IDF feature vector to represent behavioral frequency patterns, following established approaches for API-call–based malware classification [15]–[18]. The preprocessing script performs several dataset-specific cleaning steps to ensure consistent vectorization. Notably, any tokens equal to the placeholder string `__exception__` are removed before vectorization. These tokens arise when a log entry is malformed or empty during parsing; retaining them would create meaningless vocabulary entries and degrade TF–IDF stability.

After sanitization, the remaining API tokens are tokenized, assigned integer indices, and transformed into a TF–IDF matrix using the full dataset vocabulary. Family names are mapped to contiguous integer labels. The final feature matrix, label vector, and supporting metadata are stored as a compressed `.npz` file that integrates with the shared preprocessing, training, and evaluation framework.

C. Preprocessing and Data Splits

All datasets are converted into compressed `.npz` bundles that serve as the common data interface for downstream training and evaluation. Each bundle contains arrays for X_{train}, X_{val}, and X_{test}, label vectors y_{train}, y_{val}, and y_{test}, along with feature-name lists, class-name mappings, and a metadata JSON record. This format enables consistent loading logic across heterogeneous feature modalities and ensures that all models receive standardized inputs.

Stratified train/validation/test splits are produced using fixed random seed values to maintain determinism across experiments. The default partition allocates 20% of each dataset to the test set, and 20% of the remaining data to the validation set, ensuring class-proportion stability for both binary and multiclass tasks. Feature matrices are cast to `float32` to reduce memory usage, and label arrays to `int64` for compatibility with CatBoost and classical baselines. For datasets with large TF–IDF representations, such as MalAPI2019, sparse CSR matrices are retained unless densification is required for a specific model. These preprocessing steps are implemented directly in the pipeline's parsing and preconditioning modules and applied uniformly to all experiments.

TABLE II
MODEL SUPPORT IN THE EXPERIMENTAL FRAMEWORK

Model	**Key Hyperparameters**	**Role in Framework**
CatBoost	depth, learning_rate, iterations, loss_function (Logloss for detection)	Executed for all detection and classification experiments
Logistic Regression	penalty=L2, C, max_iter	Supported but not executed in this study
SVM	kernel (linear or RBF), C, gamma	Supported but not executed in this study

D. Modeling Framework

1) CatBoost Gradient-Boosted Trees: CatBoost serves as the primary model for all detection and family-classification tasks. Prior work highlights the robustness of tree boosting on heterogeneous static and behavioral features [1], [4], [5], [7], [19], [20]. For binary detection, the model is configured with `loss_function="Logloss"` and `eval_metric="AUC"`, with early stopping based on validation AUC. Multiclass classification uses `loss_function="MultiClass"` and `eval_metric="TotalF1"`. Key hyperparameters include tree depth, learning rate, maximum iterations, and random seed. Class weights may be applied for binary detection when benign–malicious ratios exceed balanced thresholds, though no explicit weighting is used for family datasets.

2) Optional Baseline Models Supported by the Framework: The experimental framework is structured to support additional baseline classifiers, including logistic regression and support vector machines. These models provide lower-complexity alternatives for future comparative studies and can be applied to both static and behavioral feature sets. Logistic regression supports dense and sparse inputs with L2 regularization, and SVMs can be configured with linear or RBF kernels depending on feature modality. Although the pipeline accommodates these baseline models, only CatBoost was executed during the experiments reported in this case study.

E. Training and Evaluation Procedure

All models are trained using the established `.npz` bundles described in Section III-C. CatBoost is trained with early stopping on the validation split, and the best iteration is automatically selected based on the configured evaluation metric. Baseline models are fit once using the full training data.

Evaluation uses a consistent metric suite for each task. For binary detection, metrics include AUROC, AUPRC, F_1,

Algorithm 1 Unified Training Procedure

```
1: Load feature bundle $(X_{\text{train}}, X_{\text{val}}, X_{\text{test}}, y_{\text{train}}, y_{\text{val}}, y_{\text{test}})$
2: Select model type based on task (binary or multiclass)
3: Initialize model with fixed hyperparameters and random seed
4: Train model on $X_{\text{train}}$; evaluate on $X_{\text{val}}$
5: Apply early stopping (CatBoost) or full optimization (baselines)
6: Compute predictions on $X_{\text{test}}$
7: Compute task-specific metrics (AUC, $F_1$, etc.)
8: Save model, metrics, and plots to output directory
```

and confusion matrices, along with ROC and precision–recall curves stored as separate figures. For multiclass classification, macro F_1 and weighted F_1 are used to capture both per-class balance and dataset skew, supplemented by full confusion matrices and per-family F_1 reports. These metrics are computed by the dataset-specific training and evaluation modules before being written to JSON and CSV outputs for reproducibility.

F. Explainability Workflow

SHAP-based feature attribution is used to examine how each model utilizes static or behavioral features [27]–[30]. For all CatBoost experiments, `TreeExplainer` is applied with a background sample of up to 200 training instances and probability outputs to produce consistent per-feature contributions. For each dataset, global explanations are generated using SHAP summary plots and mean absolute importance plots, which reveal the dominant static features in EMBER and BIG2015 and the primary API-call indicators in MalAPI2019. Local explanations are supported by the framework but omitted for space. Hooks for `LinearExplainer` and `KernelExplainer` enable future SHAP analysis of additional baseline models.

Listing 1. SHAP computation for CatBoost models

```
# X_train, X_test, and model are loaded from the NPZ
    bundle and
# CatBoost checkpoint, respectively.

background, X_test_sub = sample_background_and_test(
    X_train,
    X_test,
    background_size=args.background_size,
    test_size=args.test_size,
)

explainer = shap.TreeExplainer(
    model,
    data=background,
    model_output="probability",
)

shap_values = explainer.shap_values(X_test_sub)

plt.figure(figsize=(8, 4))
shap.summary_plot(shap_values, X_test_sub, show=
    False)
plt.title(f"{args.fig_prefix}␣CatBoost␣SHAP␣summary"
    )
plt.tight_layout()
summary_path = os.path.join(
    args.out_dir,
    f"{args.fig_prefix}_shap_summary.png",
)
plt.savefig(summary_path, dpi=300)
plt.close()
```

TABLE III
DETECTION PERFORMANCE ON EMBER2017_V2 + EMBER2018

Metric	Value
AUROC	0.998
AUPRC	0.998
F_1 Score	0.9801
Accuracy	0.9832

G. Implementation Details and Reproducibility

All experiments are implemented in Python 3.10 using CatBoost, scikit-learn, NumPy, SciPy, and PyArrow. Training is conducted on a CPU-based environment, though CatBoost GPU acceleration can be used without modification. Deterministic seeds are applied throughout preprocessing (seed 42), model initialization, and splitting logic to ensure reproducibility.

Each dataset is stored in its own standardized `.npz` bundle, and each experiment produces model artifacts, metrics, and plots in dedicated subdirectories. This structure supports full reproducibility of detection and classification results across EMBER, BODMAS, BIG2015, and MalAPI2019.

IV. RESULTS

Results are reported separately for the binary malware detection task (EMBER2017_v2 + EMBER2018) and the multi-class family classification tasks (BIG2015, BODMAS, and MalAPI2019). All metrics are computed on the held-out test sets described in Section III-C. Detection results include AUROC, AUPRC, F_1, and accuracy. Classification results include macro and weighted F_1, as well as confusion matrices that illustrate class-level behavior across datasets.

A. Detection Performance on EMBER2017_v2 + EMBER2018

Detection is evaluated on the merged EMBER2017_v2 and EMBER2018 static PE compilation using CatBoost configured with a binary loss function. Table III summarizes the primary detection metrics. The ROC and precision–recall curves are shown in Fig. 2 and Fig. 3, and the corresponding confusion matrix is provided in Fig. 4. Together, these artifacts reflect the strong separability of benign and malicious samples under static-feature analysis.

B. Classification Performance on BIG2015

Family classification performance on the BIG2015 dataset is shown in Table IV. CatBoost is trained in multiclass mode using the byte-histogram features described in Section III-B3, and the resulting confusion matrix is shown in Fig. 5. Macro and weighted F_1 are both high and closely aligned, indicating that performance is consistent across major families with limited degradation in minority classes. This behavior reflects

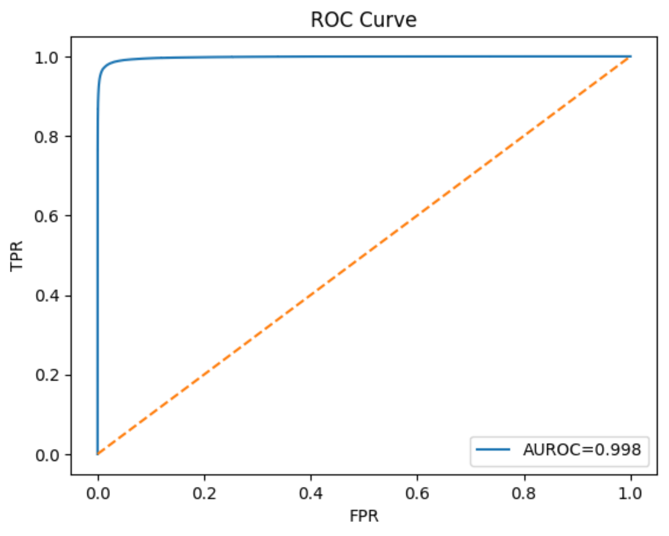

Fig. 2. ROC curve for the merged EMBER2017_v2 + EMBER2018 detection test set.

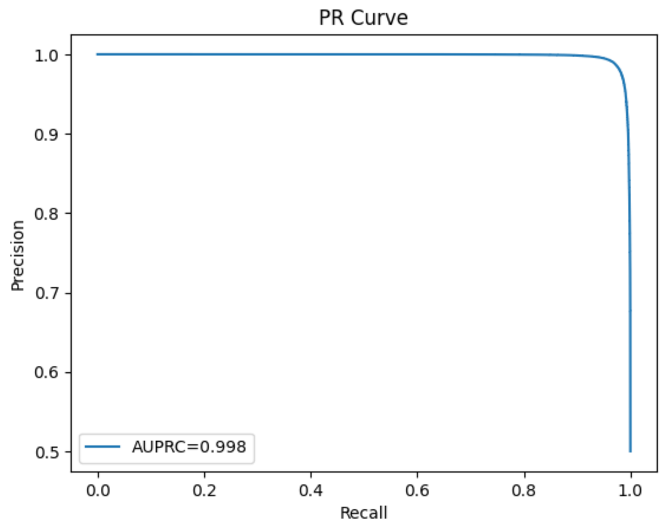

Fig. 3. Precision–recall curve for the merged EMBER detection task.

TABLE IV
CLASSIFICATION PERFORMANCE ON BIG2015

Metric	Value
Macro F_1	0.9770
Weighted F_1	0.9876

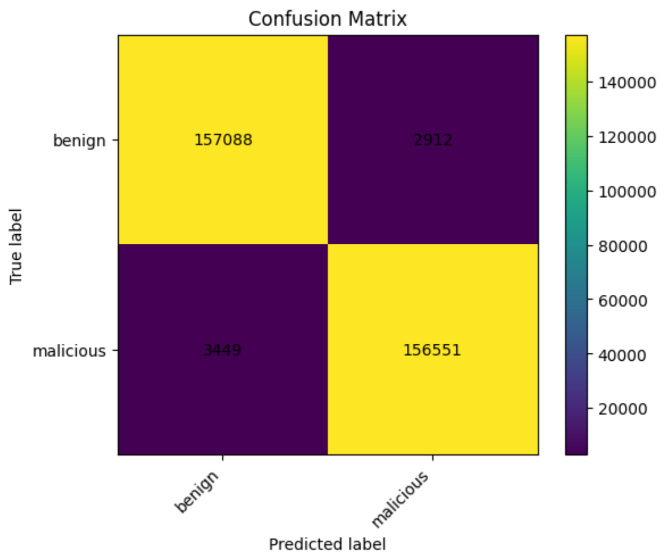

Fig. 4. Confusion matrix for the merged EMBER test set.

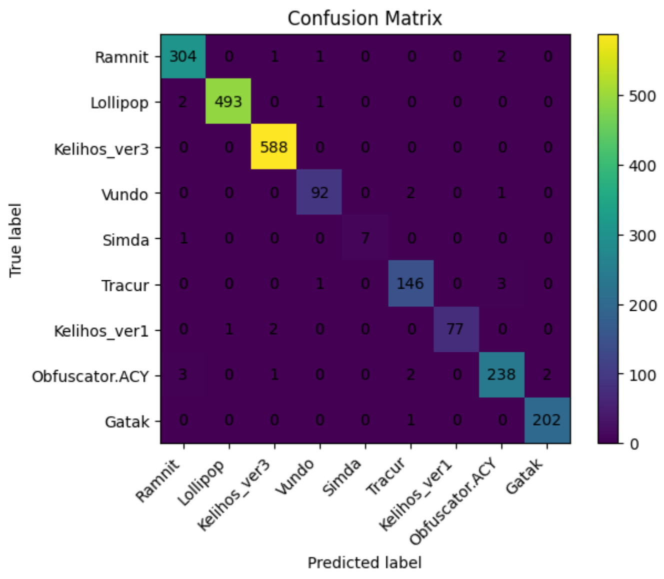

Fig. 5. Confusion matrix for BIG2015 family classification.

TABLE V
CLASSIFICATION PERFORMANCE ON BODMAS

Metric	Value
Macro F_1	0.0685
Weighted F_1	0.4311

the relatively balanced family distribution and lower label noise characteristic of BIG2015.

C. Classification Performance on BODMAS

Table V summarizes the performance of CatBoost on the BODMAS static PE dataset. These results reflect the heterogeneous labeling practices and severe class imbalance discussed in Section III-B2. The confusion matrix in Fig. 6 illustrates how predictions concentrate in a subset of dominant families.

D. Classification Performance on MalAPI2019

Table VI reports the classification metrics for MalAPI2019 using the TF–IDF API-call representations described in Section III-B4. The corresponding confusion matrix appears in Fig. 7, displaying variability across families that reflects differences in behavioral similarity and sparsity of API-call sequences.

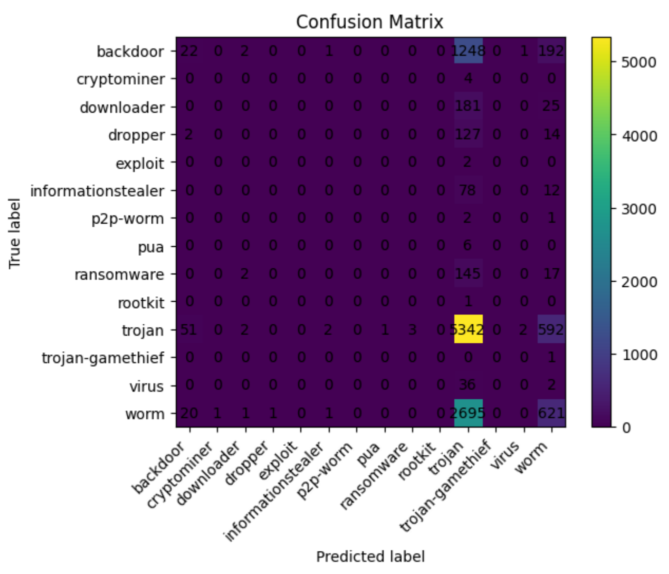

Fig. 6. Confusion matrix for BODMAS family classification.

TABLE VI
CLASSIFICATION PERFORMANCE ON MALAPI2019

Metric	Value
Macro F_1	0.3765
Weighted F_1	0.3496

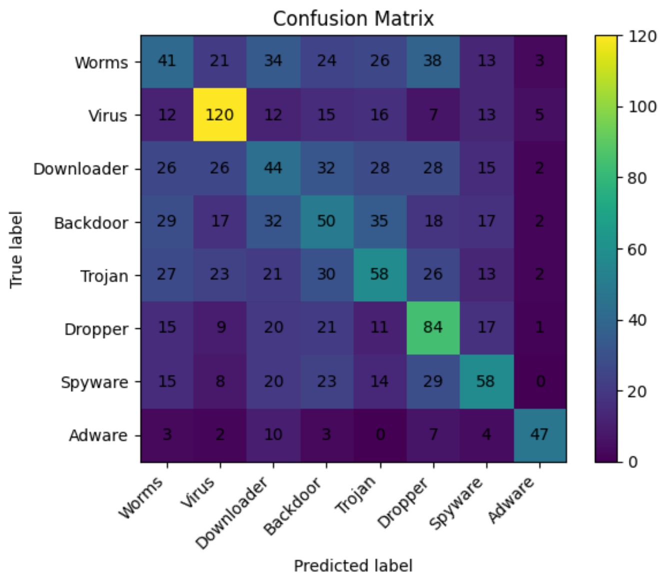

Fig. 7. Confusion matrix for MalAPI2019 family classification.

E. Cross-Dataset Summary

Table VII provides a consolidated view of results across all datasets. Detection on the merged EMBER dataset exhibits strong separability under static-feature analysis, while family classification outcomes vary substantially according to dataset characteristics, feature modality, and labeling quality. These aggregated results serve as the basis for the dataset-driven interpretation in Section V.

TABLE VII
SUMMARY OF RESULTS ACROSS ALL DATASETS

Dataset	Task	AUROC	Macro F_1
EMBER (merged)	Detection	0.998	N/A
BIG2015	Classification	N/A	0.98
BODMAS	Classification	N/A	0.07
MalAPI2019	Classification	N/A	0.38

V. DATASET-DRIVEN INTERPRETATION AND ANALYSIS

This section examines how dataset characteristics shape the performance outcomes reported in Section IV. Although a common CatBoost pipeline is used across all experiments, the datasets yield sharply different results. These differences arise primarily from structural properties such as taxonomy coherence, class imbalance, feature modality, sample diversity, and labeling noise. By comparing detection and classification across heterogeneous malware datasets, this section clarifies the conditions under which static-feature machine learning is reliable and where inherent limitations emerge.

A. Static Detection: Why EMBER Separates Cleanly

The merged EMBER2017_v2 + EMBER2018 dataset achieves near-perfect performance (AUROC=0.998, AUPRC=0.998). Three characteristics drive this separation. First, the engineered static feature set captures strong malicious-versus-benign distinctions through section entropy, import statistics, byte histograms, and header properties. The confusion matrix (Fig. 4) shows minimal overlap between classes. Second, EMBER provides hundreds of thousands of labeled samples, allowing CatBoost to model nonlinear relationships without overfitting. Third, binary detection is inherently easier than family classification because malicious binaries share broad structural traits regardless of family membership. These properties explain why static detection remains comparatively easy: the classes are statistically well separated, the feature set is rich, and the dataset is sufficiently large to support stable learning.

B. SHAP Analysis for EMBER Detection

The EMBER SHAP summary and importance plots (Fig. 8, Fig. 9) show that a small group of high-impact features (e.g., Features 637, 2359, 691, 2355) accounts for most predictive influence. The remaining features form a long low-impact tail, consistent with the dataset's strong global class separation.

C. Balanced Taxonomies and Consistent Semantics in BIG2015

BIG2015 yields the strongest family-level performance among the classification datasets (Macro F_1=0.9770). Its stability stems from four structural properties. First, the nine families follow a consistent, curated taxonomy drawn from an established benchmark, producing semantically distinct classes (Fig. 5). Second, the dataset exhibits moderate class balance, preventing dominance effects and keeping macro and

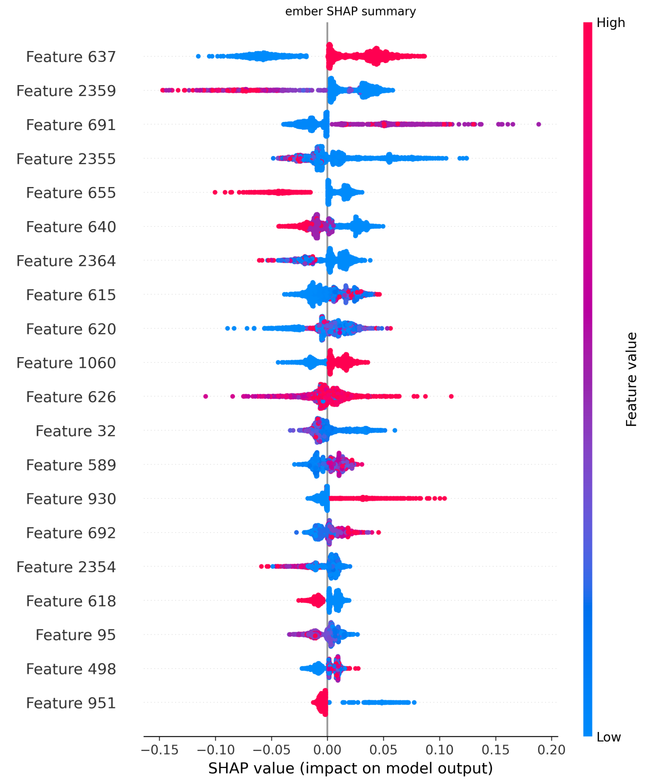

Fig. 8. SHAP summary plot for the EMBER detection model.

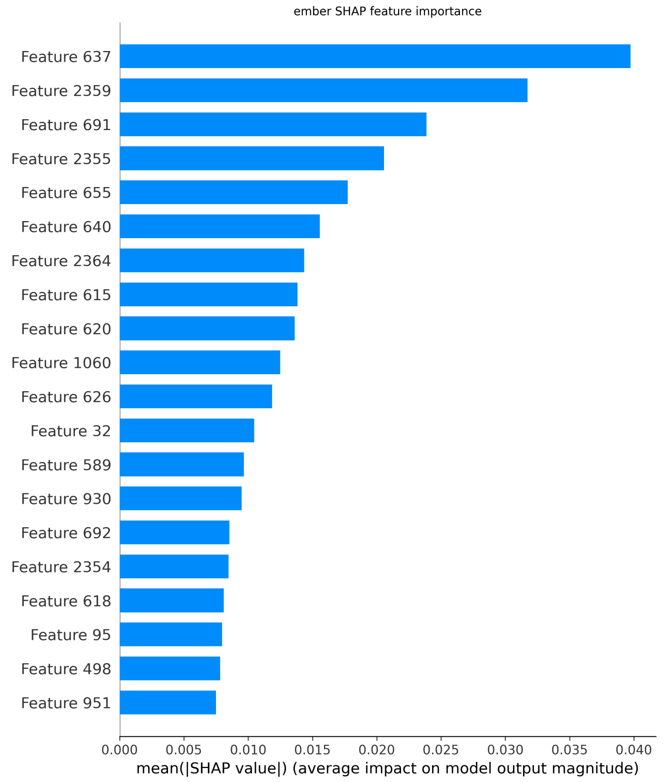

Fig. 9. Mean absolute SHAP feature importance for EMBER.

weighted scores aligned. Third, samples are largely unpacked and structurally homogeneous, which preserves discriminative static patterns. Finally, the TF–IDF and byte-histogram features capture strong lexical and statistical indicators of family identity. This combination makes BIG2015 a clean, well-behaved benchmark where static family classification is feasible and interpretable.

D. SHAP Analysis for BIG2015 Family Classification

The BIG2015 SHAP plots (Fig. 10, Fig. 11) show that a small set of features (notably Features 256, 0, 106, 257) dominates family-level decisions. Compared to EMBER, importance is distributed across a broader feature subset, reflecting the increased difficulty of distinguishing semantically related families in a multiclass setting.

E. Why BODMAS Collapses: Noise, Imbalance, and Taxonomy Drift

BODMAS produces extremely low macro performance (Macro F_1=0.0685). This outcome reflects four dataset-level issues. Severe class imbalance causes majority classes ("trojan," "worm") to dominate predictions (Fig. 6). The taxonomy aggregates heterogeneous labels from multiple vendors, mixing behavioral categories with campaign-level identifiers and creating semantic inconsistencies. Extensive packing and obfuscation flatten key static signals, limiting feature discriminability. Finally, several families share overlapping behaviors, further reducing separability under static analysis. Together,

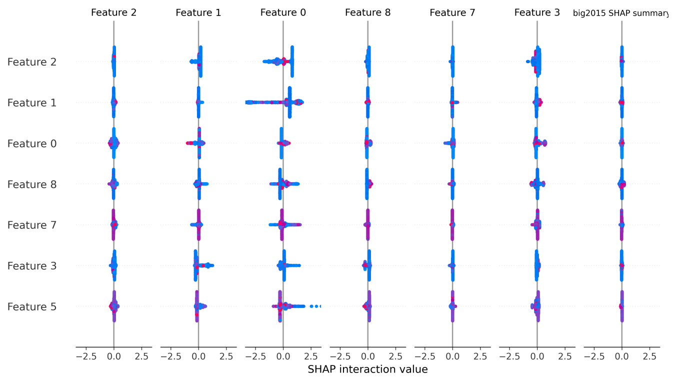

Fig. 10. SHAP summary plot for BIG2015 family classification.

these properties make fine-grained static classification effectively unattainable for BODMAS.

F. Behavioral Similarity and API-Level Ambiguity in MalAPI2019

MalAPI2019 achieves intermediate performance (Macro F_1=0.3765), reflecting both the strengths and limits of TF–IDF behavioral encoding. API-call sequences capture high-level intent (e.g., network operations, registry modification), allowing families with distinct workflows to separate clearly (Fig. 7). However, families with similar goals often share API usage patterns, causing confusion. Sequence sparsity, truncated traces, and instrumentation noise reduce

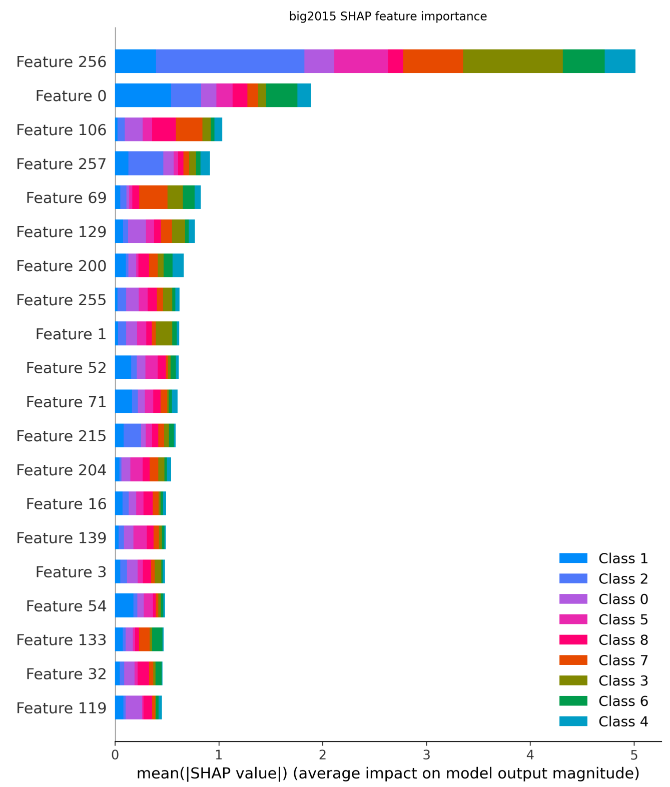

Fig. 11. Mean absolute SHAP feature importance for BIG2015.

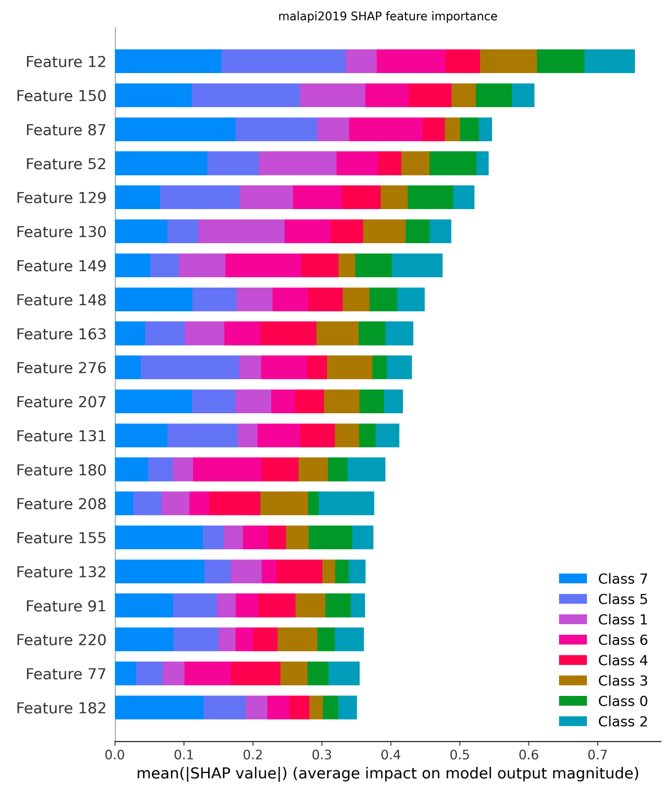

Fig. 13. Mean absolute SHAP feature importance for MalAPI2019.

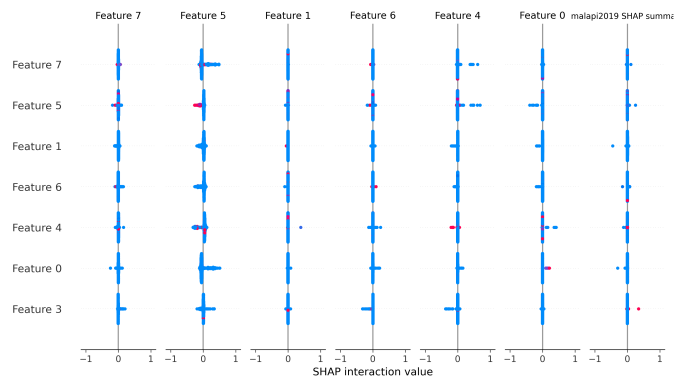

Fig. 12. SHAP summary plot for MalAPI2019 API-call classification.

robustness, particularly for minority classes. The taxonomy is less coherent than BIG2015 but more organized than BODMAS, producing intermediate difficulty.

G. SHAP Analysis for MalAPI2019 Behavioral Classification

The MalAPI2019 SHAP plots (Fig. 12, Fig. 13) show that the classifier relies primarily on a small set of API-related features (e.g., Features 12, 150, 87, 52). A wider distribution of intermediate-impact features reflects behavioral overlap among families, matching the moderate performance observed in Section IV.

H. Cross-Dataset Themes and Implications

Four themes emerge across datasets. First, detection is intrinsically easier than family classification because binary separation relies on broad structural differences, while family distinctions hinge on subtle, often unstable features or predictors. Second, dataset quality determines achievable performance more than model choice; results range from near-perfect (EMBER) to collapsed (BODMAS) under the same classifier. Third, feature modality governs learnability: static PE features excel at benign/malicious separation, whereas behavioral TF–IDF supports only coarse family grouping. Fourth, taxonomy coherence is essential; curated definitions (BIG2015) yield stable predictions, whereas mixed vendor taxonomies (BODMAS) lead to label collapse.

I. Limitations and Considerations for Future Work

This study focuses on static features for EMBER, BIG2015, and BODMAS; incorporating unpacking or hybrid representations may improve robustness for obfuscated samples. MalAPI2019 uses coarse TF–IDF behavioral encoding; sequence models or graph-based dynamic analysis could reduce ambiguity. Adversarial and evasive samples are not modeled here but represent an important area for future evaluation. Overall, the viability of ML-based malware classification depends primarily on dataset construction and labeling quality, which impose fundamental limits that model sophistication cannot fully overcome.

VI. Conclusion

This case study examined how dataset construction governs the feasibility of machine-learning malware detection and family classification. By applying a unified CatBoost workflow across five real-world datasets, the analysis isolated performance differences that arise from structural properties of the data rather than from modeling decisions. The results show a clear divide: static-feature detection remains highly reliable when datasets are balanced, consistently labeled, and engineered for benign–malicious separation, as demonstrated by the near-perfect performance on the merged EMBER2017_v2 and EMBER2018 collections.

Family classification, however, proved far more sensitive to dataset design. BIG2015 produced stable, high-accuracy predictions due to its curated taxonomy and relatively even family distribution, whereas BODMAS collapsed under severe imbalance, heterogeneous labeling, and widespread packing. MalAPI2019 fell between these extremes, reflecting both the utility and the limits of sparse behavioral representations. These outcomes emphasize that classification difficulty is shaped predominantly by label quality, feature modality, and class structure rather than by model sophistication.

Although the study focused on static features for three of the datasets and a TF–IDF abstraction for API-call logs, these constraints were necessary to maintain a controlled comparison. They also reflect current practice in large-scale malware ML, where unpacked binaries and complete behavioral traces are not always available. Future work that integrates hybrid static–dynamic features or execution-level patterns may clarify distinctions in families that appear inseparable under static analysis alone.

Overall, the findings reinforce that meaningful progress in malware classification depends as much on improving dataset design as on advancing modeling techniques. Balanced taxonomies, coherent family definitions, and representative feature modalities are prerequisites for reliable classification. Without these foundations, even state-of-the-art models cannot overcome the structural limitations embedded in the data itself.

Appendix A
Use of AI-Based Tools

AI-based tools (ChatGPT) were used only for improving clarity, refining wording, and assisting with organization and grammar. All experimental design, dataset processing, feature engineering, model development, evaluation, interpretation of results, and scientific reasoning were carried out entirely by the author. No AI system generated technical content, analysis, or conclusions presented in this study. All decisions regarding methodology, implementation, and the framing of findings reflect the author's independent work.

References

[1] A. Bensaoud *et al.*, "A survey of malware detection using deep learning," *Journal of Information Security and Applications*, 2024. [Online]. Available: https://doi.org/10.1016/j.mlwa.2024.100546

[2] A. Shalaginov *et al.*, "Machine Learning Aided Static Malware Analysis: A Survey and Tutorial," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1808.01201

[3] R. Sihwail, M. Omar, and K. A. Z. Ariffin, "A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid, and Memory Analysis," CORE, 2019. [Online]. Available: https://core.ac.uk/download/pdf/325990564.pdf

[4] F. A. Aboaoja *et al.*, "Malware Detection Issues, Challenges, and Future Directions," *Applied Sciences*, vol. 12, no. 17, p. 8482, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/17/8482

[5] A. Abusitta, B. C. M. Fung, and M. Debbabi, "Malware classification and composition analysis: A survey of recent developments," *Journal of Information Security and Applications*, vol. 58, p. 102750, 2021. [Online]. Available: https://doi.org/10.1016/j.jisa.2021.102828

[6] I. Santos *et al.*, "Static and Dynamic Malware Analysis Using Machine Learning," 2019. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.utexas.edu/stamp/stamp.jsp?tp=&arnumber=8667136

[7] Y. Wu, H. Zhuang, Y. Jia, Y. Zhang, "A Survey of Machine Learning Approaches for Malware," *ACM*, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3732365.3732410

[8] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1804.04637

[9] R. J. Joyce *et al.*, "EMBER2024: A Benchmark Dataset for Holistic Evaluation of Malware Classifiers," *arXiv*, 2025. [Online]. Available: https://arxiv.org/abs/2506.05074

[10] Elastic, "EMBER Dataset GitHub Repository," 2018. [Online]. Available: https://github.com/elastic/ember

[11] CrowdStrike, "EMBER2024: Advancing Cybersecurity ML Training on Evasive Malware," 2025. [Online]. Available: https://www.crowdstrike.com/en-us/blog/ember-2024-advancing-cybersecurity-ml-training-on-evasive-malware/

[12] D. G. Corlatescu *et al.*, "A Large-Scale Databank for Boosting Similarity Search in Malware Analysis," *NeurIPS Datasets and Benchmarks*, 2023. [Online]. Available: https://www.semanticscholar.org/reader/6348f8b62bf55eab9cf7ac5bd1cb9faee87ee089

[13] D. Bálik *et al.*, "RawMal-TF: Raw Malware Dataset Labeled by Type and Family," *arXiv*, 2025. [Online]. Available: https://arxiv.org/pdf/2506.23909

[14] J. Meoak, "Static Malware Analysis for Incident Response," Kennesaw State Univ., 2025. [Online]. Available: https://digitalcommons.kennesaw.edu/jcerp/vol2025/iss1/16/

[15] F. O. Catak and A. F. Yazi, "A Benchmark API Call Dataset for Windows PE Malware Classification," *arXiv e-prints*, 2021, [Online]. Available: https://arxiv.org/abs/1905.01999

[16] F. O. Catak, A. F. Yazı, O. Elezaj, and J. Ahmed, "Deep Learning Based Sequential Model for Malware Analysis Using Windows exe API Calls," *PeerJ Computer Science*, vol. 6, e285, 2020. [Online]. Available: https://doi.org/10.7717/peerj-cs.285

[17] Z. Li *et al.*, "Comprehensive evaluation of Mal-API-2019 dataset by multiple classical machine learning algorithms," *IJCSIT*, 2024. [Online]. Available: https://wepub.org/index.php/IJCSIT/article/view/655

[18] B. Panda, S. S. Bisoyi, S. Panigrahy, "Behavioural Analysis of Malware by Selecting Influential API Calls," *IJACSA*, 2025. [Online]. Available: https://thesai.org/Downloads/Volume16No5/Paper_75-Behavioural_Analysis_of_Malware_by_Selecting_Influential_API.pdf

[19] R. Pascanu, J. W. Stokes, H. Sanossian, M. G. Mozer, and M. Marino, "Malware Classification with Deep Learning," in *Proc. IEEE Security and Privacy Workshops*, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7163050

[20] G. Karat *et al.*, "CNN–LSTM Hybrid Model for Enhanced Malware Analysis and Detection," *Procedia Computer Science*, 2024. [Online]. Available: https://doi.org/10.1016/j.procs.2024.03.239

[21] P. V. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015. [Online]. Available: https://doi.org/10.1016/j.procs.2015.02.149

[22] I. Santos and J. Devesa, "OPEM: A Static-Dynamic Approach for Malware Detection," 2018. [Online]. Available: https://santosgrueiro.com/papers/2012/2012-Santos-OPEM.pdf

[23] P. Somasekhar, N. L. V. Venu Gopal, "Dynamic and Static Malware Detection," *IJRPR*, 2025. [Online]. Available: https://ijrpr.com/uploads/V5ISSUE11/IJRPR35575.pdf

[24] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: A Deep Learning Framework for Intelligent Malware Detection," 2016. [Online]. Available: https://www.covert.io/research-papers/deep-learning-security/DL4MD-%20A%20Deep%20Learning%20Framework%20for%20Intelligent%20Malware%20Detection.pdf

[25] N. Á. Romero, "Artificial Intelligence–Based Malware Analysis for Family Classification," UPCommons, 2021. [Online]. Available: https://upcommons.upc.edu/bitstreams/f0280300-9d61-49d2-b928-bfe1fb1ddcc7/download

[26] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017. [Online]. Available: https://doi.org/10.1007/s11416-015-0261-z

[27] H. Manthena, M. Abdelsalam, S. Shajarian, J. Kimmel, S. Khorsandroo, M. Gupta, "Explainable Artificial Intelligence (XAI) for Malware Analysis," *arXiv*, 2024. [Online]. Available: https://arxiv.org/html/2409.13723v3

[28] S. Nazim *et al.*, "Advancing malware imagery classification with explainable deep learning: SHAP, LIME and Grad-CAM," *PLOS ONE*, 2025. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0318542

[29] N. V. Sarı *et al.*, "A hybrid CNN–GRU model with XAI-driven interpretability using LIME and SHAP," *PeerJ Computer Science*, 2025. [Online]. Available: https://peerj.com/articles/cs-3258/

[30] P. Hermosilla *et al.*, "Explainable AI for Forensic Analysis: A Comparative Study of SHAP and LIME," *Applied Sciences*, vol. 15, no. 13, p. 7329, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/13/7329