Large-Scale Static Malware Detection Using Classical Machine Learning Models: An Evaluation on the EMBER Dataset

Achmad Fauzan[1], Tito Pinandita[2], Aulia Desy Nur Utomo[3]
Department of Informatics Engineering-Faculty of Engineering and Science, Universitas Muhammadiyah Purwokerto, Indonesia[1]
Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka, Malaysia[2]
Informatics Engineering Study Program, Telkom University, Purwokerto Campus, Indonesia[3]

Abstract*—Malware detection is a major difficulty in cybersecurity as malicious software continues to evolve in scale, diversity, and sophistication. While deep learning and highly complex architectures are becoming increasingly important in recent work, the practical efficiency of conventional machine learning methods for large-scale static malware detection remains underexplored. We perform a comparative evaluation of four machine learning models (Random Forest, XGBoost, Logistic Regression, and Decision Tree) on approximately 600,000 Portable Executable (PE) samples from the EMBER dataset. To enable a fair comparison of the models, we created a common experiment setup including standardised preprocessing, repeated evaluation with numerous random seeds, selective hyperparameter optimisation, feature importance analysis, and confusion matrix-based error analysis. The experimental results show a strong benefit of ensemble-based approaches for the structured feature representation provided by the EMBER dataset. Random Forest showed the best overall performance, with 96.74 % accuracy, 96.71 % F1-score, and a ROC-AUC of 0.9953, retaining a very steady behavior in repeated runs. XGBoost likewise demonstrated good predictive capacity with less training time but did not outperform Random Forest even with careful hyperparameter adjustment. On the other hand, Logistic Regression performed significantly worse, suggesting that linear decision boundaries were insufficient to capture the deep structural relationships encoded in static malware traits. Further study of the confusion matrix shows a balanced classification behavior with relatively low false negative rates, which is significant for operational malware detection situations. The feature importance analysis suggested that entropy-based features, PE structure metadata, and import-based features played an important role in the malware classification judgments. In conclusion, the results suggest that well-designed classical ensemble approaches are still quite competitive for scalable and interpretable static malware detection even with the rising usage of more and more powerful machine learning architectures.

Keywords*—Malware detection; static analysis; EMBER benchmark; ensemble learning; PE file analysis; random forest; XGBoost

I. Introduction

Malware detection has become increasingly difficult as modern cyberattacks continue to evolve in both scale and technical sophistication. In operational environments, security systems must process enormous volumes of executable files while simultaneously dealing with malware that frequently changes its structure, behavior, or observable characteristics to evade detection. Traditional signature-based approaches are still widely used because of their speed and practicality, yet their effectiveness declines substantially when facing polymorphic, packed, or obfuscated malware capable of modifying identifiable patterns with minimal effort. As a result, maintaining manually engineered signatures alone is no longer sufficient for addressing rapidly evolving threat landscapes. These limitations have contributed to the growing adoption of machine learning (ML) techniques that can identify broader detection patterns beyond previously observed malware variants [1], [2].

Among the various malware analysis strategies, static analysis remains one of the most practical solutions for large-scale detection pipelines. Unlike dynamic analysis, which requires executable files to run inside monitored environments, static analysis examines structural properties, metadata, and statistical characteristics without interacting with the runtime system. This approach significantly reduces computational overhead and enables efficient large-scale inspection of binary collections. However, static methods are not without limitations. Sophisticated malware can still employ obfuscation, encryption, or structural manipulation techniques specifically designed to conceal malicious indicators and reduce detection effectiveness [3], [4].

The development of machine learning-based malware detection has also been accelerated by the availability of large public benchmark datasets. One of the most widely adopted resources is the EMBER dataset, which provides structured feature representations extracted from Portable Executable (PE) files for reproducible large-scale experimentation [5], [6]. EMBER combines entropy-based measurements, PE metadata, import information, and hashed feature representations, allowing researchers to examine both statistical and structural characteristics of executable files within a consistent evaluation framework.

Recent studies increasingly emphasize deep learning architectures, including convolutional, recurrent, and transformer-based models capable of learning complex feature representations directly from raw or semi-structured malware data [7], [8], [9], [10], [11]. In parallel, growing attention has been directed toward adversarial robustness, transfer learning, and adaptive malware detection strategies designed to address

evolving threat landscapes [12], [13]. These developments demonstrate substantial progress in predictive capability, particularly in highly complex classification scenarios. Nevertheless, practical deployment considerations—including scalability, interpretability, computational feasibility, and reproducibility—remain important concerns in operational cybersecurity environments.

Classical ensemble learning methods continue to attract attention because of their ability to capture non-linear relationships while remaining comparatively stable in structured feature representations. Random Forest, in particular, has demonstrated strong robustness across a range of malware detection studies involving PE-based static features [14]. However, despite the growing preference for increasingly sophisticated architectures, the extent to which classical ensemble models remain competitive under large-scale static analysis conditions is still insufficiently understood.

Several limitations can also be identified within the current literature. Many existing studies rely on relatively small or imbalanced datasets, which may reduce the generalizability of reported findings. Comparative evaluations are often conducted under inconsistent preprocessing pipelines or heterogeneous experimental settings, making cross-study interpretation difficult. In addition, predictive performance is frequently discussed independently from computational behavior, feature contribution patterns, or classification error characteristics, even though these aspects are highly relevant for practical deployment scenarios [6], [7], [15].

This study addresses these gaps through a large-scale comparative evaluation of four machine learning models—Random Forest, XGBoost, Logistic Regression, and Decision Tree—using approximately 600,000 samples derived from the EMBER dataset. The study investigates whether classical ensemble methods remain competitive for large-scale static malware detection under highly structured feature conditions.

To investigate this premise, a consistent experimental framework was developed incorporating standardized preprocessing, repeated evaluation across multiple random seeds, selective hyperparameter optimization, feature importance analysis, and confusion-matrix-based error characterization. The study aims to examine how model complexity, feature structure, and computational behavior interact within large-scale static malware classification tasks. More broadly, the findings are intended to provide insight into the continuing practical relevance of classical machine learning approaches for scalable, interpretable, and operationally feasible cybersecurity systems.

II. Related Work

Machine learning has become one of the dominant approaches in modern malware detection, allowing security systems to identify malicious patterns that go beyond manually crafted signatures. Early work showed that classical algorithms such as Decision Trees, Random Forest, and Support Vector Machines can reliably separate malicious from benign binaries using structured feature representations [16], [17]. These studies, however, were typically conducted on dataset sizes several orders of magnitude smaller than the EMBER-scale collection used here, which limits how confidently their reported accuracy figures generalize to large, production-scale detection pipelines. As the field matured, attention shifted from raw classification accuracy toward feature quality, model adaptability, and long-term robustness across heterogeneous operating environments [18], [19] — questions that are addressed only partially in the present study but that motivate its emphasis on stability across repeated runs.

Ensemble-based methods have repeatedly outperformed simpler classifiers in malware classification, particularly on structured feature representations where the relationships between variables are strongly non-linear [14], [20]. Random Forest and related strategies tend to integrate heterogeneous features effectively while staying robust to noise, which makes them attractive for the scalable, computationally efficient pipelines that operational cybersecurity systems require [21], [22]. What is less consistent across this body of work is the preprocessing pipeline behind these results: some studies apply aggressive feature selection before training, while others, like the present one, retain the full processed feature set to preserve interpretability. This inconsistency makes raw performance numbers difficult to compare directly across papers, even when the underlying algorithm is the same — a point the present study tries to mitigate by holding preprocessing constant across all four models. Maintaining stable performance over time also remains an open problem, since malware continuously evolves through obfuscation, packing, and adaptive evasion [23], a limitation this study does not resolve but explicitly acknowledges in Section V.

Feature representation is central to static PE-based detection. Prior work has shown that entropy measurements, structural PE metadata, imported API functions, and string-based statistics carry a strong discriminative signal [4], [24], and the feature importance results reported here (Section IV-C) largely reinforce this pattern, with entropy- and import-based attributes again emerging as top contributors. EMBER extends these engineered features with large-scale hashed representations of imports and strings [5], [6]. While hashed features expand coverage, they come at the cost of interpretability, since individual hashed attributes cannot be inspected in isolation. The present study deliberately avoids this trade-off by retaining the complete processed feature representation for model training while reporting only a subset of interpretable features during feature importance analysis, which sacrifices some representational capacity in exchange for transparent feature attribution — a methodological choice that distinguishes it from studies that exploit the full hashed feature space for marginal accuracy gains.

A related line of work focuses on feature selection and dimensionality reduction to improve computational efficiency and reduce redundancy in large feature spaces [25], [26]. These approaches can shorten training time and, in some cases, improve stability, but identifying a universally optimal feature subset remains difficult because malware characteristics differ substantially across families and contexts [27], [28]. This is precisely why the present study takes the opposite approach: rather than searching for a reduced feature subset, it retains the complete interpretable feature set and instead controls variability through repeated evaluation across five random seeds

(Section III-E). The comparison suggests that feature-reduction strategies and seed-repetition strategies are addressing the same underlying problem — performance instability — from different angles, and a direct empirical comparison between the two remains an open question for future work.

Deep learning approaches occupy a different position in this landscape. Convolutional networks, hybrid architectures, and transformer-based systems have demonstrated strong predictive capability by learning hierarchical representations directly from raw or semi-structured malware data [7], [8], [10], and subsequent work has pushed further into transfer learning, representation learning, and adversarial robustness [9], [11], [12], [13], [29], [30]. These studies generally report higher peak accuracy than classical models, but rarely under conditions directly comparable to this study's evaluation protocol: many deep learning studies evaluate on smaller curated subsets, use GPU-dependent training pipelines, and do not report training time or statistical significance testing across repeated runs. This makes it difficult to determine whether reported accuracy gains stem from the architecture itself or from differences in dataset scale and preprocessing — a gap the present comparative framework is designed to close, at least within the classical-model space.

Model configuration also matters considerably for deep learning-based detectors. Training algorithm selection [31] and hidden-layer architecture design [32] can shift predictive performance substantially, but these gains typically come with added computational cost and optimization effort, raising practical concerns around deployment cost, scalability, and interpretability [21], [22], [33]. By contrast, the four classical models evaluated here were deliberately kept close to default configurations (with selective tuning applied only to XGBoost, as detailed in Section III-F), trading a degree of potential peak performance for a more transparent and reproducible comparison baseline.

Comparative evaluations across model families confirm that behavior depends heavily on both feature representation and dataset characteristics. Random Forest and other tree-based ensembles tend to capture non-linear relationships effectively while remaining stable under noise [26], [34], whereas linear models such as Logistic Regression often struggle in highly structured feature spaces, and single Decision Trees are prone to instability and overfitting [16], [17] — a pattern this study's own results (Section IV-A) reproduce almost exactly. Dataset composition compounds these effects, particularly when class imbalance, heterogeneous malware families, or shifting feature distributions are involved [15]. Because preprocessing pipelines and evaluation conditions vary so much from one study to the next, direct comparison across the literature is rarely straightforward; this is one reason the present study fixes a single preprocessing pipeline across all four models rather than optimizing each independently.

Within this landscape, EMBER has become one of the most widely used benchmarks for static malware detection [5], [6]. Existing EMBER-based studies have examined feature effectiveness, classification accuracy, temporal degradation, and concept drift under evolving malware conditions [6], [7], [35], [36]. A consistent finding across this work is that detection performance deteriorates once a model is deployed outside the temporal window of its training data [37], [38] — a concern this study does not directly test, since its train–test split is randomized rather than temporally ordered (a limitation discussed further in Section V). Other EMBER-based work has explored simplified, more interpretable feature representations and shown that reduced feature spaces can still achieve competitive detection accuracy [39], a finding broadly consistent with this study's own decision to retain the complete EMBER feature representation for model training while highlighting a subset of interpretable features during feature importance analysis.

Taken together, these studies report strong malware detection performance using both classical and deep learning approaches, but cross-study comparison remains genuinely difficult: dataset size, feature engineering choices, class balance, preprocessing steps, and evaluation protocols all vary simultaneously, making it hard to attribute reported performance differences to the learning algorithm alone. Few studies hold these factors constant while comparing multiple classical models side by side, and fewer still combine predictive accuracy with computational efficiency, feature importance analysis, and confusion-matrix-based error characterization within a single, unified experimental setup.

This study addresses that gap through a large-scale comparative evaluation of four classical machine learning models — Random Forest, XGBoost, Logistic Regression, and Decision Tree — on approximately 600,000 labeled EMBER samples, all processed through an identical preprocessing pipeline. Beyond classification accuracy, the evaluation framework incorporates repeated experiments across multiple random seeds, statistical significance testing, computational efficiency measurement, feature importance analysis, and detailed confusion-matrix characterization. This integrated design is intended to isolate algorithmic differences from the confounding methodological variation that complicates comparison across the broader literature reviewed above.

III. Materials and Methods

A. Dataset

The EMBER dataset was used as the primary data source for this study. EMBER provides pre-extracted static features derived from Portable Executable (PE) files and has become one of the most widely adopted benchmarks for machine learning-based malware detection research [5], [6]. The dataset contains both labeled and unlabeled samples, enabling large-scale evaluation under realistic malware analysis settings.

The EMBER 2018 training dataset contains 900,000 Portable Executable (PE) files, consisting of 300,000 benign samples, 300,000 malicious samples, and 300,000 unlabeled instances. Following the standard supervised learning setting adopted in the EMBER benchmark, unlabeled samples (label = −1) were excluded from the analysis. Consequently, the final dataset comprised 600,000 labeled PE files, including 300,000 benign and 300,000 malicious samples. This balanced class distribution reduces potential bias toward majority classes and enables fair comparison of machine learning models. The

resulting dataset was partitioned using a stratified 80:20 train–test split. Detailed dataset statistics are provided in Table I.

TABLE I. STATISTICAL SUMMARY OF THE DATASET

Property	Value
Total Samples	600000
Total Features	2381
Malicious Samples	300000
Benign Samples	300000
Train-Test Ratio	80:20
Number of Experimental Runs	5
Random Seeds	[42, 52, 62, 72, 82]
Chunk Size	5000

B. Data Preprocessing

Data preprocessing was performed to ensure consistency across all experimental stages and to improve compatibility with the evaluated machine learning models. Duplicate entries were removed to reduce redundancy, while samples labeled as −1, representing uncertain or unlabeled instances, were excluded from the analysis. Only samples labeled as 0 (benign) and 1 (malicious) were retained, following common practices in EMBER-based malware classification studies [36].

Feature scaling was subsequently applied using the StandardScaler implementation provided in the scikit-learn library. Numerical attributes were transformed into zero-mean and unit-variance distributions to reduce disparities in feature magnitude. This preprocessing step is particularly important for models such as Logistic Regression, where optimization behavior is sensitive to feature scale.

Standardization transforms each feature according to

$$z = (x - \mu) / \sigma \quad (1)$$

where, x denotes the original feature value, μ represents the feature mean, and σ denotes the standard deviation. This transformation improves numerical stability and reduces disparities in feature magnitude across variables.

Tree-based methods, including Random Forest and Decision Tree, are generally less affected by feature normalization. Nevertheless, all models were trained using the same processed dataset in order to preserve consistency throughout the comparative evaluation. Although scaling does not substantially influence tree-based learning behavior, a unified preprocessing pipeline was maintained to ensure experimental comparability across all classifiers.

C. Feature Processing

The EMBER dataset contains multiple categories of static Portable Executable (PE) features, including entropy measurements, PE header metadata, imported functions, string-based characteristics, and hashed representations extracted from executable binaries [5], [6]. In the present study, all numerical features generated during preprocessing were retained to preserve the full representational capacity of the processed dataset. Explicit dimensionality reduction or feature elimination techniques were not applied.

Missing values were handled during preprocessing to maintain dataset consistency before model training. The retained features included entropy-related measurements, PE header metadata, string-based representations, import-related characteristics, byte histograms, byte-entropy histograms, and other static Portable Executable attributes, all of which have previously been identified as informative indicators for malware classification tasks [4], [24].

After preprocessing, the resulting dataset contained 2,381 numerical features derived from the original EMBER feature representation. No feature selection, dimensionality reduction, or feature elimination procedures were applied. Consequently, all available processed features were retained and used during model training and evaluation.

For interpretability purposes, feature importance analysis focused on a subset of representative human-readable features originating from entropy-related attributes, PE metadata, imported functions, and string-based characteristics. These features were highlighted in the visual analysis because they provide intuitive explanations of model behavior, whereas many EMBER features are represented through hashed or aggregated feature encodings that are less directly interpretable.

D. Model Selection

Four machine learning models were selected to represent different learning paradigms and classification behaviors: Random Forest, Logistic Regression, Decision Tree, and Extreme Gradient Boosting (XGBoost). Random Forest, originally proposed by Breiman [40], was selected because of its strong performance on structured feature representations and its ability to capture complex nonlinear relationships through ensemble aggregation. Logistic Regression was included as a linear baseline to evaluate feature separability within the EMBER representation space. Decision Tree provides an interpretable non-linear alternative, while Extreme Gradient Boosting (XGBoost), introduced by Chen and Guestrin [41], was incorporated as a boosting-based ensemble framework capable of modeling complex feature interactions and serving as a strong contemporary benchmark.

More computationally intensive approaches, such as Support Vector Machines, were not included due to scalability limitations when applied to datasets of EMBER-scale dimensionality and size. The selected models, therefore, provide a balanced comparison across predictive performance, interpretability, computational behavior, and structural complexity [16], [20].

E. Training and Evaluation

The dataset was divided into training and testing subsets using a stratified 80:20 split configuration to preserve class balance across both partitions. This study adopted a randomized evaluation protocol because its primary objective was to perform a controlled comparative assessment of classical machine learning models under identical experimental conditions. Temporal validation based on malware collection chronology was not considered in the present work and remains an important direction for future research.

To reduce sampling bias and improve experimental robustness, all experiments were repeated across multiple random seeds. Performance results are therefore reported as mean values accompanied by standard deviations.

Classification effectiveness was evaluated using five commonly adopted metrics in malware detection research: accuracy, precision, recall, F1-score, and ROC-AUC [1], [15]. ROC-AUC was included to assess each model's ability to distinguish malicious and benign samples across varying decision thresholds, following the evaluation framework proposed by Fawcett [42].

The evaluation metrics were calculated based on the confusion matrix consisting of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Accuracy was computed as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (2)$$

Precision as

$$Precision = \frac{TP}{TP+FP}, \quad (3)$$

Recall as

$$Recall = \frac{TP}{TP+FN}, \quad (4)$$

and F1-score as

$$F1 = \frac{2TP}{2TP+FP+FN} \quad (5)$$

Performance variability across repeated runs was additionally analyzed to assess experimental stability and reproducibility.

To further investigate model behavior, confusion matrix analysis was conducted to examine false positive and false negative distributions. Feature importance analysis was also performed for tree-based models in order to identify the variables contributing most strongly to classification decisions.

The overall analytical workflow of the proposed methodology is illustrated in Fig. 1.

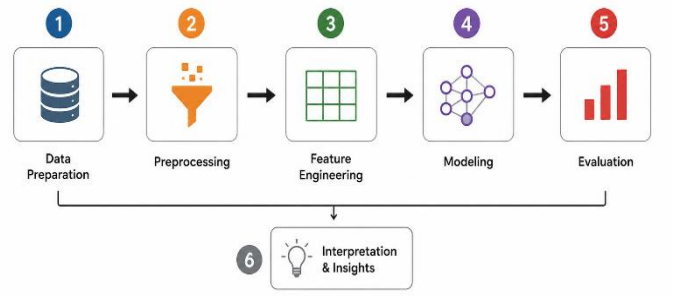

Fig. 1. Overall workflow of the proposed static malware detection framework.

F. Implementation Details

All experiments were conducted using Python 3.10. Machine learning models were implemented using the scikit-learn library (version 1.3) [43], while the XGBoost classifier was implemented using the XGBoost library (version 1.7).

The primary hyperparameter configurations used in the experiments are summarized in Table II. Most models were evaluated using default or near-default parameter settings in order to maintain consistency and reduce the risk of excessive configuration bias during comparison. Hyperparameter optimization was selectively applied to XGBoost using a randomized search strategy. The explored parameters included the number of estimators, learning rate, maximum tree depth, subsampling ratio, and regularization terms. Selective tuning was applied primarily to XGBoost because boosting-based ensemble methods are generally more sensitive to parameter configuration than bagging-based or linear approaches.

TABLE II. MAIN HYPERPARAMETERS OF THE EVALUATED MODELS

Model	Parameter	Setting
Random Forest	n_estimators	100
	max_depth	None
	min_samples_split	2
	random_state	Multiple seeds (42, 52, 62, 72, 82)
XGBoost	n_estimators	500
	max_depth	8
	learning_rate	0.05
	subsample	0.8
	colsample_bytree	0.8
	reg_lambda	1
	gamma	0
	random_state	Multiple seeds (42, 52, 62, 72, 82)
Logistic Regression	solver	lbfgs
	max_iter	1000
	random_state	Multiple seeds (42, 52, 62, 72, 82)
Decision Tree	max_depth	None
	min_samples_split	2
	random_state	Multiple seeds (42, 52, 62, 72, 82)

Models with lower sensitivity to hyperparameter variation were evaluated primarily under standardized configurations to preserve comparability and reduce unnecessary optimization bias across the experimental framework. Hyperparameter optimization was selectively applied to XGBoost using RandomizedSearchCV, as the predictive performance of gradient-boosting models is known to be highly sensitive to parameter configurations. The search process explored combinations of key parameters, including the number of estimators, maximum tree depth, learning rate, subsampling ratio, and column sampling ratio. The final configuration was selected based on cross-validation performance and subsequently used in all experiments. In contrast, Random Forest, Decision Tree, and Logistic Regression were evaluated using widely adopted baseline configurations to maintain methodological consistency and facilitate fair comparative analysis.

To reduce potential comparison bias, the objective of this study was not to identify the globally optimal configuration for

each classifier, but rather to evaluate representative implementations of commonly used machine learning algorithms under a consistent experimental framework. Random Forest, Decision Tree, and Logistic Regression were therefore evaluated using widely adopted baseline configurations, whereas XGBoost received limited hyperparameter optimization because its performance is known to be particularly sensitive to parameter settings. Consequently, the reported results should be interpreted as a comparison between a tuned boosting-based ensemble and standard implementations of alternative classifiers rather than as an exhaustive hyperparameter optimization study.

IV. Results

A. Performance Comparison of Machine Learning Models

The predictive performance of the evaluated machine learning models—Random Forest (RF), XGBoost, Logistic Regression (LR), and Decision Tree (DT)—was analyzed using five standard classification metrics: accuracy, precision, recall, F1-score, and ROC-AUC. The corresponding results are presented in Table III.

Random Forest produced the strongest classification performance across nearly all evaluation metrics. The model achieved an accuracy of 0.9674 ± 0.0005, accompanied by a precision of 0.9769 ± 0.0007 and an F1-score of 0.9671 ± 0.0005. Its ROC-AUC value reached 0.9953 ± 0.0001, indicating an exceptionally strong capability to distinguish between benign and malicious samples across varying decision thresholds. The relatively small standard deviations observed across repeated runs further suggest stable behavior under different random initializations.

XGBoost (Tuned) also demonstrated high predictive capability, achieving an accuracy of 0.9493 ± 0.0007 and a ROC-AUC of 0.9906 ± 0.0002. Although the difference compared with Random Forest appears numerically modest, the separation remained consistent across all evaluation metrics. This indicates that both ensemble-based approaches adapted effectively to the engineered static PE feature representation, yet Random Forest maintained a slight but persistent advantage in classification consistency.

TABLE III. Performance Comparison of Machine Learning Models on the EMBER Dataset

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC	Training Time (s)
Random Forest	0.9674 ± 0.0005	0.9769 ± 0.0007	0.9574 ± 0.0007	0.9671 ± 0.0005	0.9953 ± 0.0001	48.5112 ± 0.5054
XGBoost (Tuned)	0.9493 ± 0.0007	0.9511 ± 0.0009	0.9474 ± 0.0006	0.9492 ± 0.0007	0.9906 ± 0.0002	20.3657 ± 0.3057
Logistic Regression	0.7402 ± 0.0009	0.7355 ± 0.0012	0.7502 ± 0.0012	0.7428 ± 0.0008	0.8047 ± 0.0010	4.3501 ± 1.2262
Decision Tree	0.9417 ± 0.0008	0.9385 ± 0.0015	0.9453 ± 0.0010	0.9419 ± 0.0008	0.9417 ± 0.0008	9.0032 ± 0.2118

To further evaluate whether the observed performance differences between Random Forest and XGBoost were statistically significant, paired t-test and McNemar test analyses were conducted. As shown in Table IV, both tests produced extremely small p-values, indicating that the observed differences are unlikely to be attributable to random variation alone. These findings provide additional evidence supporting the robustness of the comparative results obtained in this study.

TABLE IV. Statistical Significance Analysis Between Random Forest and XGBoost

Test	Comparison	P-value
Paired t-test	RF vs XGBoost	4.26×10^{-8}
McNemar Test	RF vs XGBoost	1.04×10^{-245}

Decision Tree achieved competitive results, with an accuracy of 0.9417 ± 0.0008 and a recall of 0.9453 ± 0.0010. The model showed reasonable discriminative capability despite its substantially simpler structure. However, its ROC-AUC value remained noticeably lower than that of the ensemble methods, suggesting reduced robustness when handling more ambiguous decision boundaries within the dataset.

A markedly different pattern emerged for Logistic Regression. The model obtained an accuracy of 0.7402 ± 0.0009 and ROC-AUC of 0.8047 ± 0.0010, substantially below the other evaluated approaches. Precision and recall also remained comparatively limited, indicating that linear decision boundaries were insufficient to fully capture the complex structural relationships embedded in the static PE features. The performance gap becomes particularly visible in Fig. 2, where the separation between linear and ensemble-based methods is clearly illustrated.

In terms of computational cost, Logistic Regression required the shortest training time, averaging 4.35 ± 1.23 s. Decision Tree and XGBoost followed with training times of 9.00 ± 0.21 s and 20.37 ± 0.31 s, respectively. Random Forest required the longest training duration at 48.51 ± 0.51 s. Even so, the increase in computational overhead was accompanied by consistently superior predictive performance, especially in ROC-AUC and precision, which are highly relevant in malware detection scenarios where false classifications may carry significant operational consequences.

A visual comparison of model performance is presented in Fig. 2.

The comparative results reveal a distinct separation between ensemble-based models and simpler linear approaches. Random Forest and XGBoost maintained consistently high scores across all metrics, suggesting strong compatibility with the statistical and structural characteristics of the EMBER feature representation. The stability observed across repeated experiments also indicates that the extracted features provide highly separable decision regions for tree-based learning methods.

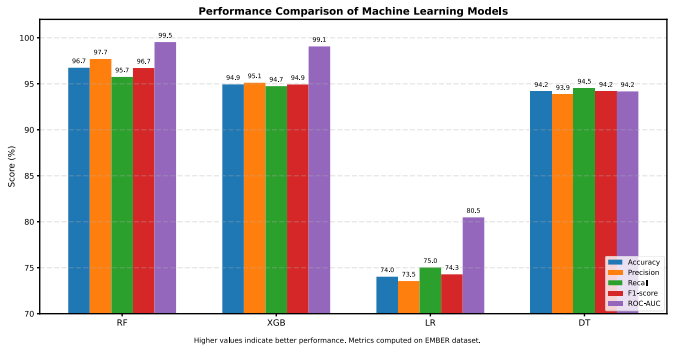

Fig. 2. Comparison of classification performance across evaluated machine learning models.

Another notable observation concerns the relationship between predictive performance and computational efficiency. XGBoost achieved substantially lower training time than Random Forest while still maintaining strong classification capability. Nevertheless, Random Forest consistently preserved higher accuracy, precision, and ROC-AUC values across all experimental runs. This behavior suggests that bagging-based ensemble strategies may provide stronger robustness than boosting approaches under highly structured static malware feature conditions.

To further evaluate whether the performance difference between Random Forest and XGBoost was statistically meaningful, paired statistical analysis was conducted. Both paired t-test and McNemar analysis indicated statistically significant differences between Random Forest and XGBoost ($p < 0.001$), confirming that the observed performance advantage of Random Forest was statistically reliable rather than attributable to random variation.

B. Confusion Matrix Analysis

To examine classification behavior in greater detail, confusion matrix analysis was performed for the Random Forest model. The corresponding results are presented in Fig. 3.

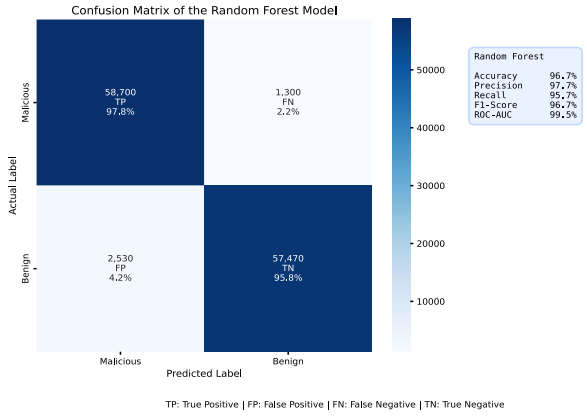

Fig. 3. Confusion matrix of the random forest model for malware detection.

The confusion matrix demonstrates that the model maintained a strong detection capability for both malware and benign samples.

A total of 58,700 malicious files were correctly identified as malware, representing 97.8% of the malicious class. At the same time, 57,470 benign samples were correctly classified, corresponding to 95.8% true negative recognition. These results are consistent with the high accuracy and ROC-AUC values reported previously in Table II, indicating that the model preserved stable discrimination across both classes rather than favoring a single category.

Misclassification rates remained comparatively limited, although several important patterns can still be observed. The model produced 1,300 false negatives, equivalent to 2.2% of malicious samples incorrectly predicted as benign. From a cybersecurity perspective, this category of error is particularly critical because undetected malware may continue to execute within operational environments without triggering defensive mechanisms.

The number of false positives was slightly higher, with 2,530 benign samples incorrectly classified as malicious, corresponding to 4.2% of the benign class. While false positives may increase operational overhead by generating unnecessary alerts or manual inspections, they are generally considered less severe than false negatives in malware detection systems, where missed threats can lead to direct security compromise.

Another notable characteristic of the confusion matrix is the relatively balanced classification behavior across both classes. The model did not exhibit substantial bias toward either malicious or benign predictions, suggesting that the ensemble structure of Random Forest was able to capture stable decision boundaries within the EMBER feature space. This balance is further reflected in the close alignment between precision (97.7%) and recall (95.7%), as illustrated in Fig. 3.

The distribution of errors also provides indirect insight into the nature of the EMBER dataset itself. Samples located near the decision boundary may contain overlapping structural or statistical characteristics, particularly in cases involving obfuscated binaries, packed executables, or benign software with behavior patterns resembling malicious activity. Despite these challenges, the relatively small proportion of misclassified samples indicates that the extracted static features remain highly informative for large-scale malware discrimination tasks.

C. Feature Importance Analysis

Feature importance analysis was conducted using the Random Forest model to identify the variables contributing most strongly to malware classification decisions. The ten highest-ranked features are presented in Fig. 4.

Among the evaluated attributes, strings.entropy emerged as the most influential feature, with an importance score of 0.1088. This result suggests that statistical irregularities within extracted string content provide highly discriminative signals for distinguishing malicious and benign executables. Elevated entropy values are commonly associated with packing, encryption, or obfuscation techniques frequently employed by malware to conceal internal behavior and evade static inspection.

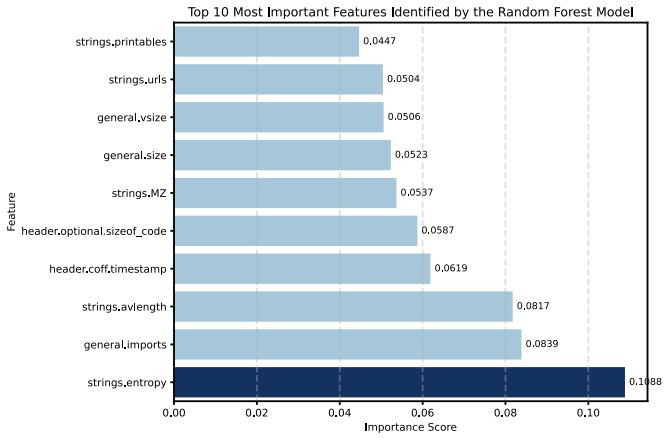

Fig. 4. Top 10 most important features identified by the random forest model.

The second and third most influential features were general.imports (0.0839) and strings.avlength (0.0817). The prominence of general.imports indicate that imported API patterns remain highly informative for malware identification, reflecting differences in functionality and execution behavior between benign and malicious software. Meanwhile, the importance of average string length suggests that textual artifacts embedded within executable files carry measurable structural information relevant to classification.

Several PE header and structural attributes also contributed substantially to model decisions. Features such as header.coff.timestamp (0.0619) and header.optional.sizeof_code (0.0587) ranked among the most relevant variables, indicating that metadata and binary organization continue to provide meaningful indicators of malicious activity. These attributes may capture indirect signatures related to compilation patterns, code organization, or manipulation of executable structures.

Additional features associated with binary size and embedded strings—including strings.MZ, general.size, general.vsize, strings.urls, and strings.printables—also demonstrated notable importance scores ranging from 0.0447 to 0.0537. Although individually less dominant than entropy-related attributes, their collective contribution highlights the multidimensional nature of the EMBER feature space, where both statistical and structural characteristics jointly influence classification behavior.

Another notable observation is the diversity of feature categories represented within the top-ranked variables. Rather than relying on a single dominant indicator, the Random Forest model leveraged complementary information derived from entropy measures, PE structural metadata, import-related characteristics, and string-based representations. This behavior likely contributed to the strong predictive stability observed in the previous evaluation results, particularly under repeated experiments across multiple random seeds.

The feature distribution shown in Fig. 4 also reinforces the suitability of tree-based ensemble methods for static malware analysis. Because Random Forest can capture complex non-linear interactions among heterogeneous attributes, the model is able to integrate multiple weakly correlated indicators into a robust classification framework. This characteristic is particularly advantageous in malware detection environments, where malicious samples often exhibit subtle variations designed to evade signature-based identification.

V. Discussion

The comparison between Random Forest and XGBoost is particularly noteworthy from a methodological perspective. Although XGBoost underwent selective hyperparameter optimization and achieved lower training time, it did not surpass the predictive performance of the bagging-based ensemble model. Furthermore, the superiority of Random Forest remained consistent across repeated experimental runs, suggesting that the observed differences cannot be attributed solely to parameter tuning. Instead, the findings indicate that the interaction between the learning mechanism and the structural characteristics of the EMBER feature representation may play a more influential role than algorithmic complexity alone. Similar observations regarding the effectiveness of ensemble tree methods in malware classification have been reported in previous studies involving structured PE-based feature representations [14], [26].

The results further suggest that feature structure may play a more decisive role than model complexity itself. The EMBER dataset contains a mixture of entropy-related attributes, import-based information, PE metadata, and hashed representations that collectively form highly informative decision regions. It should be noted that the reported feature importance rankings are limited to interpretable features presented for explanatory purposes. The predictive models themselves were trained using the complete set of 2,381 processed EMBER features. Ensemble methods appear particularly well-suited for integrating these heterogeneous feature categories while maintaining robust classification boundaries.

This behavior becomes more evident in the feature importance analysis. Entropy-based variables, especially strings.entropy, emerged as a dominant contributor to the classification process, indicating that statistical irregularities associated with packing, encryption, or obfuscation remain highly informative indicators of malicious activity. Import-related and structural PE features also contributed substantially, reinforcing the idea that executable organization and API dependency patterns preserve detectable traces even under code variation. These observations are consistent with prior work emphasizing the discriminative value of PE structural features in static malware analysis [4], [6].

A different pattern emerged for Logistic Regression. The model demonstrated substantially lower performance across all evaluation metrics, suggesting that linear decision boundaries are insufficient for capturing the complex relationships embedded within the EMBER feature space. Malware classification in large-scale static datasets appears to involve non-linear interactions among structural and statistical attributes, limiting the effectiveness of simpler linear models under realistic conditions.

The confusion matrix analysis provides additional practical insight into classification behavior. Random Forest maintained high true positive and true negative rates while preserving a relatively balanced error distribution across both classes. From

an operational cybersecurity perspective, the low false negative rate is particularly important because undetected malware may continue executing without triggering defensive mechanisms. Although false positives were still present, their proportion remained comparatively limited relative to the scale of the dataset.

The reported results should be interpreted within the context of a controlled benchmark evaluation. Because the objective of this study was comparative model assessment rather than deployment-oriented forecasting, model performance was evaluated using randomized train–test partitions rather than temporally separated malware collections.

Modern malware increasingly employs packing, encryption, and obfuscation techniques that can reduce the effectiveness of purely static feature representations. These conditions may partially explain the false negative cases observed in the present evaluation, where approximately 2.2% of malicious samples were incorrectly classified as benign.

From a computational standpoint, the findings also reveal an important trade-off between efficiency and predictive strength. Random Forest required longer training time than Logistic Regression and Decision Tree, yet the additional computational overhead remained reasonable considering the substantial improvement in detection capability. In practical deployment scenarios, such trade-offs are often acceptable when stronger malware discrimination significantly reduces the risk of missed threats.

Several limitations should nevertheless be acknowledged. First, the study relies exclusively on static analysis features extracted from Portable Executable files. While static analysis offers major advantages in scalability and processing speed, it may be less effective against sophisticated malware designed specifically to evade structural inspection through obfuscation or polymorphic transformation techniques [2].

Second, the evaluation was conducted using a temporally bounded snapshot of the EMBER dataset. Malware ecosystems evolve continuously, and feature distributions may shift over time as attackers adopt new evasion strategies, modify compilation behavior, or alter execution patterns across environments. Prior studies have shown that such concept drift conditions can significantly degrade malware detection performance when models are deployed beyond the temporal scope of their training data [37], [38]. As a result, continuous adaptation and periodic retraining mechanisms may become necessary to preserve long-term detection effectiveness.

Another limitation concerns the scope of the evaluated models. Deep learning architectures were intentionally excluded in order to focus on the balance between predictive performance, interpretability, and computational efficiency. However, representation-learning approaches may capture latent behavioral characteristics that are not fully represented through manually engineered static features. Similarly, the present evaluation did not explicitly measure operational deployment factors such as inference latency, memory consumption, or robustness against adversarial manipulation, all of which are becoming increasingly relevant in modern cybersecurity systems.

The evaluation was conducted on labeled samples only. Although this setting follows the standard EMBER benchmark protocol for supervised learning, future studies may investigate semi-supervised approaches that leverage the available unlabeled instances.

Furthermore, the evaluation employed a randomized train–test split rather than temporal validation. Consequently, the reported results reflect comparative model performance under controlled benchmark conditions and may not fully capture concept drift or evolving malware characteristics encountered in real-world deployment scenarios.

Another limitation concerns the potential overlap of malware families between the training and testing partitions. Although stratified random splitting is commonly adopted in EMBER-based studies, the present work did not explicitly enforce family-aware partitioning. Consequently, related malware families may appear in both subsets, potentially leading to optimistic performance estimates. Future research should investigate family-aware evaluation protocols to better assess model generalization to previously unseen malware families.

More broadly, the findings suggest that model selection in malware detection should be guided by the characteristics of the available feature representation rather than assumptions regarding algorithmic sophistication alone. Within the experimental setting considered in this study, classical ensemble methods provided a favorable balance between predictive performance, robustness, scalability, and interpretability.

VI. Conclusion

This study conducted a large-scale comparative evaluation of machine learning models for static malware detection using the EMBER dataset, with particular emphasis on predictive performance, computational efficiency, and classification behavior under consistent experimental conditions. The results demonstrate that strong malware detection capability can still be achieved through classical machine learning approaches when supported by informative and well-structured feature representations.

Among the evaluated models, Random Forest consistently produced the strongest predictive results across accuracy, precision, F1-score, and ROC-AUC metrics while maintaining stable behavior across multiple random seeds. The model also demonstrated balanced classification characteristics in the confusion matrix analysis, achieving high detection rates for both malicious and benign samples with relatively limited misclassification. Feature importance analysis further showed that entropy-based attributes, import-related patterns, and structural PE characteristics contributed substantially to classification decisions, indicating that static executable features continue to provide highly discriminative signals for large-scale malware analysis.

XGBoost also achieved strong classification performance, although its predictive capability remained slightly below that of Random Forest despite selective hyperparameter optimization. Within the scope of the two ensemble strategies compared here, this finding suggests that increasing boosting-model sophistication through tuning does not necessarily translate into measurable improvement once the feature space already exhibits

strong separability. In contrast, Logistic Regression showed substantially lower effectiveness, reinforcing the non-linear nature of malware classification within the EMBER representation space.

The findings also highlight an important practical implication for cybersecurity system design. Model selection should not be driven exclusively by trends toward increasingly complex architectures, but rather by the compatibility between learning mechanisms, feature structure, and operational requirements. In the present study, the bagging-based ensemble strategy provided a favorable balance between predictive robustness and computational feasibility without requiring excessive optimization procedures.

From an applied perspective, Random Forest remains a highly practical solution for scalable static malware detection environments. Its combination of strong predictive stability, interpretable feature behavior, and moderate computational requirements makes it suitable for deployment scenarios where reliability and efficiency are both critical considerations.

At the same time, several challenges remain open for future research. The current framework relies exclusively on static PE-based features and does not incorporate runtime behavioral analysis, temporal adaptation, or adversarial robustness evaluation. The evaluation is also limited to classical machine learning models; deep learning and transformer-based architectures, which have shown strong results in related EMBER studies, were not included in the present comparison and therefore cannot be directly assessed against the models evaluated here. Future studies may therefore benefit from integrating static and dynamic representations, benchmarking against deep learning-based detectors under matched experimental conditions, exploring adaptive learning strategies for evolving malware ecosystems, and evaluating resilience against adversarial manipulation in real-world operational settings.

Taken together, the results indicate that under highly structured feature conditions such as those provided by EMBER, a well-designed classical ensemble method like Random Forest achieved higher predictive performance than the tuned XGBoost configuration evaluated in this study. Whether this advantage extends to deep learning or other more complex architectures was outside the scope of this study and remains an open question for future comparative work.

ACKNOWLEDGMENT

The authors would like to thank the contributors of the EMBER dataset for making the dataset publicly available for malware detection research. During the preparation of this manuscript, the authors used ChatGPT (OpenAI) and QuillBot for language refinement and grammatical improvement. The authors reviewed and edited the generated content and take full responsibility for the content of this publication.

REFERENCES

[1] V. Rathod, C. Parekh, and D. Dholariya, “AI & ML-Based Anomaly Detection and Response Using Ember Dataset,” in 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), IEEE, Sep. 2021, pp. 1–5. doi: 10.1109/ICRITO51393.2021.9596451.

[2] K. M. Balasubramanian et al., “Obfuscated Malware detection using Machine Learning models,” in 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2023, pp. 1–8. doi: 10.1109/ICCCNT56998.2023.10307598.

[3] A. Pathak, Th. S. Kumar, and U. Barman, “Static analysis framework for permission-based dataset generation and android malware detection using machine learning,” EURASIP J. Inf. Secur., vol. 2024, no. 1, p. 33, 2024, doi: 10.1186/s13635-024-00182-3.

[4] L. El Neel, A. Copiaco, W. Obaid, and H. Mukhtar, “Comparison of Feature Extraction and Classification Techniques of PE Malware,” in 2022 5th International Conference on Signal Processing and Information Security (ICSPIS), 2022, pp. 26–31. doi: 10.1109/ICSPIS57063.2022.10002693.

[5] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.04637

[6] Y. Oyama, T. Miyashita, and H. Kokubo, “Identifying Useful Features for Malware Detection in the Ember Dataset,” in 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), IEEE, Nov. 2019, pp. 360–366. doi: 10.1109/CANDARW.2019.00069.

[7] S. Pramanik and H. Teja, “EMBER - Analysis of Malware Dataset Using Convolutional Neural Networks,” in 2019 Third International Conference on Inventive Systems and Control (ICISC), IEEE, Jan. 2019, pp. 286–291. doi: 10.1109/ICISC44355.2019.9036424.

[8] G. Karat, J. Kannimoola, N. Nair, A. Vazhayil, S. V G, and P. Poornachandran, “CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection,” Procedia Comput. Sci., vol. 233, pp. 492–503, May 2024, doi: 10.1016/j.procs.2024.03.239.

[9] A. Al-ojaimi, “Advanced Framework for Detecting Malware in Portable Executable (PE) Files Using a Multi-Model,” Journal of Information Systems Engineering and Management, vol. 10, pp. 769–781, May 2025, doi: 10.52783/jisem.v10i36s.6562.

[10] T. Kacem and S. Tossou, “Trandroid: An Android Mobile Threat Detection System Using Transformer Neural Networks,” Electronics (Basel)., vol. 14, no. 6, 2025, doi: 10.3390/electronics14061230.

[11] A. Tyagi, “Scaling deep learning models: Challenges and solutions for large-scale deployments,” World Journal of Advanced Engineering Technology and Sciences, vol. 16, pp. 10–20, May 2025, doi: 10.30574/wjaets.2025.16.2.1252.

[12] M. Eslamnejad, R. Taheri, M. Shojafar, and M. Bader-El-Den, “Federated learning-based robust android malware detection: label-flipping attacks and defenses,” Neural Comput. Appl., vol. 37, no. 32, pp. 27057–27082, 2025, doi: 10.1007/s00521-025-11656-x.

[13] D. Gibert, G. Zizzo, Q. Le, and J. Planes, “Adversarial Robustness of Deep Learning-Based Malware Detectors via (De)Randomized Smoothing,” IEEE Access, vol. 12, pp. 61152–61162, 2024, doi: 10.1109/ACCESS.2024.3392391.

[14] I. Darmawan, H. Setiaji, L. Sutriani, A. Supriyadi, R. Rizal, and A. Rahmatulloh, “Ensemble Learning for Malware Classification: A Performance Evaluation Using Random Forest and Hist Gradient Boosting,” May 2025, pp. 1–6. doi: 10.1109/ICoICT66265.2025.11193116.

[15] P. Mittal, H. S. Lallie, and E. Titis, “Impact of imbalanced datasets on ML algorithms for malware classification,” Information Security Journal: A Global Perspective, vol. 34, no. 3, pp. 251–264, 2025, doi: 10.1080/19393555.2025.2459736.

[16] R. Hasan et al., “Enhancing malware detection with feature selection and scaling techniques using machine learning models,” Sci. Rep., vol. 15, no. 1, p. 9122, 2025, doi: 10.1038/s41598-025-93447-x.

[17] A. M. Hilal et al., “Malware Detection Using Decision Tree Based SVM Classifier for IoT,” Computers, Materials and Continua, vol. 72, no. 1, pp. 713–726, 2022, doi: 10.32604/cmc.2022.024501.

[18] D. Dunsin, M. C. Ghanem, K. Ouazzane, and V. Vassilev, “Reinforcement learning for an efficient and effective malware investigation during cyber incident response,” High-Confidence Computing, vol. 5, no. 3, p. 100299, 2025, doi: 10.1016/j.hcc.2025.100299.

[19] D. Tomar, A. K. Verma, and K. Chhillar, “A Hybrid Static-Dynamic Malware Analysis Framework Using Interpretable Neural Network,”

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT, vol. 09, May 2025, doi: 10.55041/IJSREM52505.

[20] Md. F. Bin Hafiz, N. A. Khan, Z. Kamal, S. Hossain, and S. Barman, "A Robust Malware Classification Approach Leveraging Explainable AI," in 2024 International Conference on Intelligent Systems for Cybersecurity (ISCS), 2024, pp. 1–6. doi: 10.1109/ISCS61804.2024.10581382.

[21] Y. Hou, T. Truong-Huu, Z. Chen, C.-K. Kwoh, and S. G. Teo, "PROTEUS: Domain Adaptation for Dynamic Features in AI-assisted Windows Malware Detection," in 2023 IEEE International Conference on Data Mining Workshops (ICDMW), 2023, pp. 1322–1331. doi: 10.1109/ICDMW60847.2023.00170.

[22] Z. Zhang, H. A. Hamadi, E. Damiani, C. Y. Yeun, and F. Taher, "Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research," IEEE Access, vol. 10, pp. 93104–93139, 2022, doi: 10.1109/ACCESS.2022.3204051.

[23] A. Sabbah, R. Jarrar, S. Zein, and D. Mohaisen, "Empirical Evaluation of Concept Drift in ML-Based Android Malware Detection," Jul. 2025. [Online]. Available: http://arxiv.org/abs/2507.22772

[24] M. Yousuf, I. Anwer, T. Shakir, M. Siddiqui, and M. Shahid, "A Multi-Feature Dataset for Windows PE Malware Classification," SSRN Electronic Journal, May 2023, doi: 10.2139/ssrn.4359519.

[25] T.-H. Lai, Y.-J. Tsai, and C.-L. Liu, "Improving the Performance of Static Malware Classification Using Deep Learning Models and Feature Reduction Strategies," Mathematics, vol. 13, no. 23, 2025, doi: 10.3390/math13233753.

[26] P. S. Nguyen, T. N. Huy, T. A. Tuan, P. D. Trung, and H. V. Long, "Hybrid feature extraction and integrated deep learning for cloud-based malware detection," Comput. Secur., vol. 150, p. 104233, 2025, doi: 10.1016/j.cose.2024.104233.

[27] H. Darabian et al., "A multiview learning method for malware threat hunting: windows, IoT and android as case studies," World Wide Web, vol. 23, no. 2, pp. 1241–1260, Mar. 2020, doi: 10.1007/s11280-019-00755-0.

[28] H. Mohammadian, G. Higgins, S. Ansong, R. Razavi-Far, and A. A. Ghorbani, "Explainable Malware Detection through Integrated Graph Reduction and Learning Techniques," Dec. 2024. [Online]. Available: http://arxiv.org/abs/2412.03634

[29] A. Bensaoud and J. Kalita, "CNN-LSTM and transfer learning models for malware classification based on opcodes and API calls," Knowl. Based. Syst., vol. 290, p. 111543, 2024, doi: 10.1016/j.knosys.2024.111543.

[30] Z. Zhao, S. Yang, and D. Zhao, "A New Framework for Visual Classification of Multi-Channel Malware Based on Transfer Learning," Applied Sciences, vol. 13, no. 4, 2023, doi: 10.3390/app13042484.

[31] H. Mustafidah, U. B. Pambudi, and S. Suwarsito, "Selection of the most accurate training algorithm in the backpropagation network based on the accuracy of data pattern matching," 2022, p. 060012. doi: 10.1063/5.0111243.

[32] H. Mustafidah, S. Geography, and S. N. C. Permatasari, "Accuracy of the Neurons Number in the Hidden Layer of the Levenberg-Marquardt Algorithm," International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 4, pp. 2349–2353, Nov. 2019, doi: 10.35940/ijrte.D8259.118419.

[33] A. Agarwal, S. B. Verma, and S. Singh, "Real-Time Malware Prevention and Detection (MP&D) Framework in Cloud Computing Environments," in 2024 International Conference on Cybernation and Computation (CYBERCOM), IEEE, Nov. 2024, pp. 273–278. doi: 10.1109/CYBERCOM63683.2024.10803195.

[34] A. Alhogail and R. A. Alharbi, "Effective ML-Based Android Malware Detection and Categorization," Electronics (Basel)., vol. 14, no. 8, p. 1486, Apr. 2025, doi: 10.3390/electronics14081486.

[35] C. Galen and R. Steele, "Evaluating Performance Maintenance and Deterioration Over Time of Machine Learning-based Malware Detection Models on the EMBER PE Dataset," in 2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS), IEEE, Dec. 2020, pp. 1–7. doi: 10.1109/SNAMS52053.2020.9336538.

[36] M. Şandor, R. M. Portase, and A. Coleşa, "Ember Feature Dataset Analysis For Malware Detection," in 2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP), IEEE, Oct. 2023, pp. 203–210. doi: 10.1109/ICCP60212.2023.10398693.

[37] F. Ceschin, M. Botacin, H. M. Gomes, F. Pinagé, L. S. Oliveira, and A. Grégio, "Fast & Furious: On the modelling of malware detection as an evolving data stream," Expert Syst. Appl., vol. 212, p. 118590, 2023, doi: 10.1016/j.eswa.2022.118590.

[38] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, "Android malware concept drift using system calls: Detection, characterization and challenges," Expert Syst. Appl., vol. 206, p. 117200, 2022, doi: 10.1016/j.eswa.2022.117200.

[39] J. Mojžiš and M. Kenyeres, "Interpretable Rules with a Simplified Data Representation - a Case Study with the EMBER Dataset," in Data Analytics in System Engineering, R. Silhavy and P. Silhavy, Eds., Cham: Springer International Publishing, 2024, pp. 1–10. doi: 10.1007/978-3-031-53552-9_1.

[40] L. Breiman, "Random Forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[41] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[42] T. Fawcett, "An introduction to ROC analysis," Pattern Recognit. Lett., vol. 27, no. 8, pp. 861–874, Jun. 2006, doi: 10.1016/j.patrec.2005.10.010.

[43] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, Jun. 2012.