Enhancing Malware Detection Using Machine Learning Models on Static Features

Ashwag Alotaibi, Mounir Frikha
Department of Computer Networks and Communications-College of Computer Sciences and Information Technology,
King Faisal University, Al-Ahsa, 31982, Saudi Arabia

***Abstract*—This research introduces a CPU-optimized static malware-detection framework for resource-constrained environments, such as endpoints and IoT devices. We address the significant challenge of high memory and computational demands by proposing a robust, memory-safe data ingestion pipeline. This pipeline exclusively extracts histogram-based static features, employs type compression, and utilizes batch-wise loading with global sample limits to prevent memory overflows on systems with only 16 GB of RAM and no GPU support. Our core contribution is a compact stacking ensemble composed of three high-efficiency gradient-boosting models: LightGBM, CatBoost, and XGBoost, with a LightGBM meta-learner. This novel ensemble structure enables efficient, CPU-only training and inference while ensuring strong detection performance. Evaluated on the EMBER 2024 dataset, the framework achieves 86.99% accuracy, 0.87 F1-score, and 0.9473 AUC. This work fills a critical gap by demonstrating that carefully optimized gradient-boosting ensembles can serve as a highly deployable alternative to resource-intensive Deep Learning methods in limited security situations.**

***Keywords*—Malware detection; machine learning (ML); static features; stacking ensemble; CPU optimization; resource constraints; memory efficiency; computational efficiency**

I. Introduction

A. Background

Malware detection has become a crucial area of research and development in cybersecurity due to the growing complexity and frequency of cyberattacks. Malware, including viruses, worms, trojans, and ransomware, is a serious threat to individuals, enterprises, and critical infrastructure, causing data breaches, financial losses, and service disruptions [1][2]. As dependence on interconnected systems increases, cloud services and large enterprise networks grow, the attack surface that malicious actors exploit also expands [3][4]. These trends impose significant demands on endpoint security solutions to provide accurate and timely detection while operating on hardware with limited computational resources [5].

Traditional signature-based and heuristic detection methods rely on predefined rules or known patterns and remain effective against previously observed threats [6]. They are inherently limited when it comes to zero-day exploits and rapidly evolving polymorphic and metamorphic malware that alter their code to bypass static signatures [4]. These methods rely on regularly updated signature databases, which might leave systems vulnerable when updates fail to keep up with new campaigns and don't reveal much about previously undiscovered behaviors. As a result, ML methods have become an acceptable alternative for malware detection since they can learn from labeled data and generalize to threats by modeling complex relationships between features and malicious behavior [3].

The ability of ML to identify complex patterns and adjust to changing threat environments is what makes it scientifically relevant for malware detection [3][7]. ML models can increase the overall speed and accuracy of malware identification, automate the detection process, and reduce the need for manual analysis [7]. Static analysis, which evaluates file characteristics without executing them, offers a secure and effective method for extracting distinguishing features such as Portable Executable (PE) headers, API calls, and entropy values [8][7]. However, malware detection based on MLis not free from difficulties, including high false-positive rates, challenges in adapting to new malware families, and vulnerability to attacks designed to avoid detection [4].

Deep Learning (DL) and hybrid static-dynamic approaches can improve detection quality by learning more comprehensive representations or integrating various feature sources, but require GPU-accelerated training and inference, significant memory resources, and more intricate deployment processes [5]. These resource demands make many high-capacity models difficult to deploy on endpoints, laptops, and IoT devices, where CPU-only execution, constrained RAM, and strict latency requirements are common. Furthermore, in large-scale systems, AutoML and optimization-focused frameworks frequently place a higher priority on accuracy than on managing resource footprint, inference time, or operational false positives.

These limitations highlight the importance of malware detection frameworks that explicitly integrate operational constraints such as CPU-only execution, limited memory usage, and regulated false-positive rates, while maintaining effective detection performance. The EMBER dataset family has become a widely used benchmark for static malware research [9][10]. The original EMBER dataset provides standardized feature vectors for 1.1 million Windows PE files, while EMBER2024 extends this concept to more than 3.2 million files across several formats, enabling holistic evaluation of static malware classifiers at scale [10]. Against this backdrop, focusing on a constraint-aware gradient-boosting ensemble over EMBER static features offers a practical path toward high-quality malware detection that remains deployable on CPU-only, memory-limited systems.

B. Problem Statement and Motivation

The evolution of malware has exposed the vulnerabilities of traditional signature-based detection, necessitating a shift

toward ML techniques that can identify sophisticated threats. While current research often relies on resource-heavy, GPU-accelerated DL, practical antivirus solutions must remain functional on constrained hardware like laptops and IoT devices. This research introduces a pipeline-based framework utilizing EMBER 2024 static features and optimized clusters to bridge this gap. By prioritizing memory-safe data ingestion and consistent performance within hardware limitations, the approach enables high-performance malware detection in real-world, resource-restricted environments.

A primary motivation of this study is to establish the upper bounds of detection capability in a "zero-trust" environment, where structural and string-based features are intentionally withheld. By restricting the model to 256-byte histogram features, a 90% reduction in feature density compared to standard sets, the research aims for a benchmark accuracy exceeding 85% and an AUC of 0.9473. This focus on "Green AI" for cybersecurity demonstrates that robust, high-accuracy detection is achievable even with minimal feature complexity, proving that efficient statistical analysis can substitute for computationally expensive feature engineering.

C. Objectives

The primary objective of this research is to design and evaluate a high-accuracy and memory-optimized malware detection framework using the EMBER 2024 dataset and advanced ensemble ML techniques on CPU-based systems. The following are the specific research objectives:

- Develop a memory-safe data ingestion pipeline to handle massive sample files, use batch-based loading, and avoid memory overflow.
- Build a three-model stacking ensemble using LightGBM, CatBoost, and XGBoost, with a final LightGBM meta-learner.
- Train the model on large-scale data without a GPU.

D. Contributions

The research's contributions show how a constraint-driven approach can result in improvements in ML-based malware detection. The key contributions are:

1) Constraint-aware feature synthesis: Unlike traditional models that use high-dimensional feature sets, we propose a methodology centered on minimalist feature extraction. By utilizing only 256-byte histogram features, we investigate the upper limits of detection performance when structural and string-based data are intentionally withheld to simulate zero-trust, low-resource environments.

2) Compensatory stacking architecture: We introduce a multi-stage Gradient Boosting Stacking Ensemble specifically designed to compensate for reduced feature density. This architecture demonstrates that "shallow" but high-entropy features (histograms) can reach near-state-of-the-art AUC levels when processed through a non-linear meta-learner layer, providing a blueprint for "Green AI" in cybersecurity.

3) Resource-deterministic training protocol: We formalize a training methodology that prioritizes memory-safety as a deterministic constraint rather than a secondary optimization, ensuring predictable performance on 16GB RAM infrastructures.

II. Related Work

Modern malware is becoming increasingly widespread and sophisticated, which means that detection techniques must constantly change. Due to their growing vulnerability to polymorphism, zero-day attacks, and advanced obfuscation techniques, the traditional dependence on signature-based and heuristic approaches is quickly becoming insufficient [11]. ML-based static malware detection has evolved rapidly in the last years, with work ranging from classical ML on engineered PE features to DL on raw bytes and images, as well as hybrid static–dynamic approaches and AutoML-based optimization. This section provides a summary of recent research, critically analyzes their methodologies and limitations, and subsequently emphasizes the research gap that drives this research.

A. Overview of Existing Work

Recent studies will be grouped into four main themes, classical ML on static PE/Android features, DL on static representations (bytes, images, graphs), hybrid static–dynamic or multi-feature approaches, and AutoML and optimization-focused frameworks for malware detection.

1) Classical ML on static features: Several studies combine traditional ML classifiers with hand-crafted static features like PE headers, imported APIs, permissions, and metadata.

Shatnawi et al. [12] present a method for detecting Android malware that relies only on static analysis, collecting permissions, API, and other manifest/code features, and training classifiers like Random Forest (RM) and Support Vector Machine (SVM) to achieve high accuracy on benchmark Android datasets.

A survey conducted by Wu et al. [13] regarding static detection technologies for Android malware indicates that the majority of ML-based Android detectors depend on features such as permissions, API calls, control-flow information, and code structure, with traditional models, including SVM, Decision Tree (DT), and ensemble methods, have demonstrated high accuracy, however, they frequently suffer from dataset bias and concept drift.

A static analysis study conducted by Yuk and Seo [14], which is based on PE, utilizes various attributes from the PE header along with features related to packing in conjunction with stacking ensembles. This approach results in a detection accuracy of approximately 95.2%, surpassing the performance of single-model baselines when tested on Windows malware datasets.

Representative classical-ML static approaches are summarized in Table I.

2) DL on static representations: Deep neural networks are increasingly being used instead of manual feature engineering over raw or lightly processed static artifacts, such as images or byte sequences.

TABLE I. MCURES

Reference	Year	Platform	Method / Model	Key Static Features	Main Results
Shatnawi et al.[12]	2022	Android	Traditional ML classifiers (RF, SVM)	Permissions, API calls, other manifest/code features	High accuracy and F1-score on Android malware datasets.
Wu et al.[13]	2021	Android	Survey of ML methods (SVM, DT, ensembles)	Permissions, API calls, control-flow, code structure	Many methods report high accuracy (¿90%) on specific datasets but show sensitivity to dataset selection and evolving malware.
Yuk and Seo [14]	2022	Windows PE	Stacking ensemble of multiple ML models	PE headers, packing-related features	Detection rate around 95.2%, outperforming single model baselines on PE malware datasets.

A study conducted by Akerele et al. [15] reports that DL models, including CNNs, can learn discriminative patterns directly from static and dynamic features but remain vulnerable to packing and code obfuscation.

Salas et al. [16] investigate DL-based static malware classification, comparing several architectures for PE malware classification based on static features and highlighting strong accuracy but limited interpretability and robustness against obfuscation.

Daniel and Patel [17] focus on the detection and classification of malware within PE files through the application of DL techniques, which involves converting PE files into grayscale images and training CNN-based models. The results indicate an accuracy of approximately 94.78%, a precision of 88.64%, and a recall of 94.76%, thereby demonstrating the efficacy of static detection methods based on image analysis (see Table II).

3) Hybrid static–dynamic or multi-feature approaches: To increase robustness and detection rates, some studies integrate static with dynamic analysis or combine several static features.

Yousuf et al. [18] suggest a static malware detection system for Windows PE that achieves a detection rate of approximately 99.5%. However, it is important to note that such high performance is often reported on smaller, relatively balanced datasets (thousands of samples) using dimensionality reduction. These results frequently lack a discussion on how the model would scale to imbalanced, multi-million sample benchmarks like EMBER 2024, where the sheer volume and variety of malware make near-perfect accuracy significantly harder to maintain.

Hybrid architectures that encode both static and dynamic behaviors are highlighted in Bensaoud et al. [19] survey, which works on DL for malware detection, it is noted that these systems are more resource-intensive and more difficult to implement on limited hardware (see Table III).

4) AutoML and optimization-focused frameworks for malware detection: Brown et al. [20] examine the application of AutoML in static malware detection using DL techniques. It reveals that deep feedforward neural networks generated by AutoML can achieve performance levels similar to those of manually optimized models when applied to static malware datasets. Additionally, the study explores the impact of AutoML parameters on both detection accuracy and training cost.

Classical ML, DL, and AutoML pipelines are covered in Wu et al. [21] assessment of ML techniques for malware detection. It highlights the trade-offs between accuracy, interpretability, and computing cost as well as the difficulties of implementing large models in real-time settings (see Table IV).

5) EMBER Dataset: While Anderson and Roth [9] established a high-performance baseline ($AUC > 0.99$) using the full 2,351-feature set on 1.1 million samples with high-RAM systems, their LightGBM model's performance serves as an idealized upper bound rather than a direct comparison for resource-constrained environments. To provide a more equivalent quantitative comparison, we evaluated a localized LightGBM baseline using the same 200,000-sample limit and restricted 256-byte histogram features employed in our study. Under these identical sampling and feature constraints, a standard LightGBM baseline achieved an accuracy of 83.2% and an AUC of 0.912. Our proposed stacking ensemble outperforms this constrained baseline with 86.99% accuracy and 0.9473 AUC, demonstrating that the ensemble architecture successfully compensates for the intentional reduction in feature dimensionality.

Subsequent research [22] utilizes EMBER features to develop alternative classifiers, such as XGBoost-based detectors, which exhibit performance similar to or slightly superior to the original LightGBM baseline. However, the primary emphasis remains on enhancing accuracy rather than addressing deployment limitations.

Existing EMBER-based studies leave several important issues underexplored, most EMBER papers emphasize accuracy or AUC and show that gradient-boosted trees outperform early DL models like MalConv [9], but they do not design or evaluate pipelines specifically for CPU-only, memory-limited environments typical of endpoints and IoT devices. Additionally, EMBER2024 baselines illustrate performance on multiple tasks and file types but do not systematically address memory usage, training on constrained hardware, or end-to-end detection latency [10]. Also, does not provide a full, optimized ingestion and training pipeline that avoids memory errors for large-scale training on systems [9].

There is little systematic study of how such ensembles behave on updated, large-scale benchmarks like EMBER2024 when trained with CPU-only resources and strict memory limits.

B. Critical Analysis

Although classical ML on static features achieves high accuracy, these methods usually rely on the particular feature engineering choices and dataset features. Many studies are validated on datasets that are balanced or nearly balanced, frequently with a small range of malware families. This can inflate accuracy and hide problems like high false-positive rates

TABLE II. DL-BASED STATIC MALWARE DETECTION

Reference	Year	Platform	Representation	DL Model	Main Results
Akerele et al.[15]	2025	Mixed	Static and dynamic features.	Various DL models	DL improve detection over traditional methods, but struggle with packed/obfuscated malware and may overfit.
Salas et al.[16]	2023	Windows PE	Static feature vectors of PE files	DL architectures for classification	High detection accuracy and low error rates reported, but interpretability and robustness remain issues.
Daniel and Patel [17]	2025	Windows PE	Grayscale image of PE file	CNN-based model	accuracy = 94.78%, precision = 88.64%, recall = 94.76% on PE malware dataset.

TABLE III. HYBRID STATIC-DYNAMIC APPROACHES

Reference	Year	Platform	Features	Model	Main Results
Yousuf et al.[18]	2023	Windows PE	Four static feature sets and merged combinations	ML with ensemble learning and dimensionality reduction	Detection rate = 99.5%, error rate = 0.47% on Windows PE dataset.
Bensaoud et al.[19]	2024	Mixed	static and dynamic features	Various DL models (CNNs, RNNs, hybrids)	Hybrid models improve detection robustness but increase computational cost and deployment complexity.

TABLE IV. AUTOML AND OPTIMIZATION-FOCUSED STATIC MALWARE DETECTION

Reference	Year	Platform	Features	Model	Main Results
Brown et al. [20]	2024	Static malware datasets	Four static features	AutoML-generated Deep FFNN	AutoML models comparable in accuracy to manually crafted DL models, with automated hyperparameter tuning and architecture search.
Wu et al.[21]	2025	Mixed	static and dynamic features	Comparative analysis of ML/DL/AutoML	Highlights accuracy–cost trade-offs and notes that many high-accuracy methods require significant compute and memory.

or poor generalization to new malware variants. Furthermore, the literature often highlights detection rate or overall accuracy, while providing limited studies of false positives and model behavior in actual class-imbalance scenarios [12], [13], [14], [18].

Although DL-based methods may identify intricate patterns and minimize the need for manual feature engineering, they have a number of limitations in terms of robustness and usability. Deploying DL models for malware detection on resource-constrained systems, like user laptops or IoT devices, can be challenging because they often rely on GPU-accelerated training and inference. Additionally, these models can also be vulnerable to adversarial transformations and packing, as small modifications or obfuscation of the binary may cause significant changes in the learned representations while preserving malicious behavior [15], [19], [17], [16].

The highest reported detection rates are frequently attained by hybrid and multi-feature systems that integrate static and dynamic analysis or combine multiple static feature sets, however, an increase in complexity and consumption of resources. Collecting dynamic features requires executing samples in sandboxes or monitoring runtime behavior, which is time-consuming and not always feasible in strict real-time or large-scale environments. Furthermore, many hybrid solutions presume access to powerful servers or clusters and do not specifically address deployment constraints such as memory limits, CPU-only environments, or the requirement for quick per-sample analysis [19], [18].

Although AutoML-based methods simplify the selection of models and hyperparameters, they still usually prioritize accuracy above optimizing for limitations such as memory footprint, inference speed, or robustness under real-world workloads. AutoML pipelines can be computationally costly to operate, and current research often assesses them using offline datasets without providing comprehensive reports on resource consumption, end-to-end detection delay, or integration into operational security pipelines. Overall, there is limited focus on controlling false positives in high-volume environments, carefully quantifying detection latency, and ensuring that static ML pipelines are specifically developed for CPU-only or low-resource deployments [13], [21], [18], [20].

Furthermore, a significant discrepancy often exists between the high detection rates (95–99%) reported in literature and the performance achievable under strict operational constraints. The widely cited EMBER LightGBM baselines of $> 95\%$ typically utilize the full feature set (2,351 features) and are trained on the complete dataset (1.1 million or more samples) using high-performance hardware with substantial RAM. In contrast, our framework operates under a 'worst-case' resource scenario: a strict 16 GB RAM limit and CPU-only processing. This necessitates a deliberate reduction in model complexity, including the use of only histogram-based features and a capped sample size of 200,000 training instances. The lower accuracy of 86.99% is thus a direct result of these hardware-bound trade-offs, providing a more realistic measure of effectiveness for low-resource endpoint deployment than idealized lab-environment results.

C. Research Gap and Novelty

Current DL and hybrid malware detection models often prioritize raw accuracy at the expense of operational efficiency, requiring significant GPU power and memory that standard endpoints lack. Furthermore, existing research frequently overlooks the systematic optimization of gradient-boosting ensembles for large-scale datasets under strict CPU and memory constraints. Our research addresses these gaps by introducing a static, machine learning-based framework designed specifically for commodity hardware. Unlike computationally intensive DL or hybrid analysis, this constraint-aware pipeline utilizes memory-safe data ingestion and batch-based processing to maintain high detection accuracy while ensuring bounded resource usage.

The novelty of this framework lies in its information-theoretic approach, which seeks to maximize class separation while minimizing feature dimensionality. Rather than scaling hardware to meet data demands, we utilize feature compression and sample capping to test whether byte-frequency distributions contain sufficient latent information for identification. By employing a stacking ensemble of LightGBM, CatBoost, and XGBoost, the architecture captures inter-model variance to detect modern malware effectively, without the high latency or false-positive rates typical of less resource-constrained methods.

Beyond practical feasibility, this work establishes a new benchmark for Constraint-Aware Feature Synthesis. We provide theoretical evidence that high-entropy "shallow" features (256-byte histograms) contain sufficient latent information to achieve an AUC of 0.9473 when processed via a non-linear stacking meta-learner. This challenges the prevailing "Deep Learning-first" paradigm by proving that model architecture (stacking ensemble) can compensate for an intentional 90% reduction in feature dimensionality.

III. Methodology

The novelty of this framework lies in its Information-Theoretical approach to malware detection. While standard engineering fixes focus on scaling hardware to meet data demands, this research explores the inverse problem: how to maximize class separation using the lowest possible feature dimensionality. By compressing features to float32 and enforcing sample caps, we are not merely "saving memory"; we are testing the hypothesis that the statistical distribution of byte frequencies (histograms) contains sufficient latent information to identify modern malware, provided the ensemble architecture is deep enough to capture inter-model variances between LightGBM, CatBoost, and XGBoost.

A. Overview of Experimental Design and Constraints

1) Research objectives and constraints rationale: This study's methodological approach has been designed to address a crucial need in large-scale MLfor cybersecurity: achieving high accuracy in predictions while operating under stringent memory and computational constraints. The primary objective was to utilize the EMBER 2024 dataset and advanced ensemble MLmethods to develop and evaluate a high-accuracy and memory-optimized malware detection framework on typical CPU-based systems.

This methodology is fundamentally defined by the constraint definition, which states that the entire pipeline must operate stably on a system with only 16 GB RAM and no access to GPU acceleration. Due to this constraint, the development of novel data ingestion and modeling strategies is required. Many of the earlier studies using the EMBER feature set rely significantly on strong DL models, which frequently require cluster resources or specialized GPU hardware. In contrast, this work focuses on performance optimization under real-world memory constraints, offering a useful framework appropriate for environments with limited resources like enterprise endpoint protection systems.

The design goal was not merely to detect malware, but to prove that high detection accuracy (specifically exceeding 85% in the preliminary analysis) is achievable using carefully optimized gradient-boosting ensembles and a fully CPU-compatible pipeline. The framework of the data loading and model training approach is specifically designed to address these hardware constraints, guaranteeing that the research stays applicable to practical anti-virus solutions that cannot rely on high-performance hardware.

2) Hardware and computational environment specification: The experiment was executed on a single system limited to 16 GB of RAM and CPU-only processing without employing any GPU acceleration. The methodology included a number of important general optimization strategies to guarantee the pipeline's stability and effectiveness in this limited context. These included the methodical application of **Numpy float32** compression for feature arrays, the creation of a memory-capped loading approach to control data ingestion, and the choice of gradient boosting models recognized for their speed and efficiency on CPU clock speeds. Additionally, a necessary trade-off between predictive capability and system RAM consumption was ensured by optimizing parameters, including tree depth, leaves, and estimators, through the management of model complexity.

B. Data Sourcing, Preprocessing, and Memory-Constrained Feature Engineering

The success of the methodology depends on the rigorous control of data flow and memory usage, considering the massive size of the dataset in relation to the system resources at hand.

1) Dataset description and selection rationale: The research employed the EMBER 2024 Malware Dataset (Malware Benchmark for Research), which is a standardized, extensive, pre-extracted feature representation of the PE (Portable Executable) file structure, delivered in a JSONL (JSON Lines) file format. Each record in the binary classification process is given a label:

- Label **0** denotes a **Benign**.
- Label **1** denotes **Malware**.

The dataset is provided as several distinct files distributed across multiple training files **(train.jsonl)** and testing files **(test.jsonl)**. This volume of the dataset, with the number of samples in each file reaching hundreds of thousands, makes loading the complete feature set challenging on typical hardware configurations.

Strictly using only the **histogram** features from the JSONL records in addition to the binary **label** was an important methodological choice for feature engineering. This choice was made strategically because, in comparison to other possible EMBER features, the histogram features offer a fixed and relatively compact data size while maintaining high discriminative capability for static analysis. The methodology confirmed the commitment to memory efficiency above feature richness by focusing on increasing the total number of samples that could be processed and stored in the 16 GB RAM by extracting only the relevant histogram features.

2) Memory-limited data ingestion pipeline: Because of the size of the EMBER dataset, a unique approach to data ingestion is required. A batch-based controlled loading mechanism was developed and implemented because the dataset size, when fully loaded, exceeds the few gigabytes available on the 16 GB system after considering the operating system and model structures. This memory-safe approach is essential to the stability of the overall experiment.

a) Feature extraction and selection: The Memory-Safe JSONL Loader was implemented to read the input files line by line. During this process, it selectively extracted only the required "histogram" array (as the feature set) and the "label" (as the target variable). This selective extraction method immediately reduces the memory overhead associated with parsing and storing unnecessary metadata or complex feature structures.

b) Data compression and type casting: Explicit type casting was used to aggressively compress arrays to avoid a MemoryError during array creation (a crucial low-level methodological optimization). The Numpy float32 data type was used to represent the feature arrays, which store the histogram data, reducing the memory required for floating-point representation in half when compared to the default 64-bit standard. At the same time, Numpy int8, the smallest numerical type available, was used to represent the binary labels. This compression approach guarantees that the extensive training set, in conjunction with the memory consumed by the model parameters and intermediate boosting structures, consistently operates within the 16 GB RAM constraint.

c) Controlled batch loading and global sample capping: To maintain stability within the 16 GB RAM limit, global allocation constraints were applied to the dataset. While the EMBER 2024 dataset contains over 3.2 million files, a cap of 200,000 training samples and 80,000 testing samples was enforced to prevent system exhaustion. To mitigate the risk of reduced model capacity from this capping, a stratified random sampling strategy was employed. This ensures that the 200,000 samples maintain the original class distribution of the larger 3.2M dataset (Label 0: Benign vs. Label 1: Malware), preventing the model from developing a bias toward the majority class. This approach ensures that the "compact" training set remains a statistically representative subset of the holistic benchmark.

d) Inter-batch memory management: In addition to compression and capping, the data ingestion pipeline incorporated inter-batch garbage collection mechanisms. This active memory management technique ensures that system resources are explicitly released after the processing of each batch, preventing memory fragmentation and progressive memory creep that would otherwise lead to instability when training on hundreds of thousands of samples.

The key parameters defining the memory-limited data ingestion pipeline are summarized in Table V.

C. ML Architecture: Stacking Ensemble Design and Configuration

1) Overview of the ensemble framework rationale: A Stacking Ensemble Classifier was used to achieve the objectives without using GPU-accelerated DL architectures. The effectiveness of Gradient Boosting Machines (GBMs) with tabular data and their established optimization for CPU operations justify their employment. The stacking structure, which combines three different and strong GBMs, enables the methodology to comply with the CPU-only constraint and leverage their various strengths (speed, resilience, and generalization capability).

2) Base learner selection and configuration: Three leading gradient boosting models were selected as Base Learners, chosen specifically for their efficiency and high performance potential on tabular data.

a) LightGBM: The main reason for including this model was its effectiveness. It makes use of histogram-based gradient boosting, which, when compared to conventional tree-based methods, greatly maximizes training speed and naturally reduces memory usage, making it ideal for the volume of sample data seen in the EMBER dataset.

b) CatBoost: Selected because of its resilience and ability to manage both numerical and categorical features inside its structure, and is known for its ability to prevent overfitting, which gives the multi-model ensemble architecture the stability it needs.

c) XGBoost: A well-known and powerful gradient-boosted tree model, was incorporated due to its excellent generalization capacity and effectiveness with tabular feature architectures.

3) Training procedure and hyperparameter constraints: The training process was implemented in two phases. First, the same set of 200,000 optimized float32 histogram features was used to train each of the three base models independently.

Because only 16 GB of RAM was available, the training process had to be carefully optimized to ensure that model structures didn't exceed the available memory, which could cause the system to become unstable. As a result, all base models were purposefully pre-set with conservative configuration parameters intended to control computation time and memory usage. These characteristics included a lower learning rate, a limited tree depth, and an intermediate number of estimators. These particular configurations reflect an essential trade-off: for assurance of stability and effective use of the restricted system RAM, potential peak complexity is sacrificed.

4) Meta-learner design and integration: The core function of the Stacking Framework is the synthesis of the base learners' outputs by a Meta-Learner. The Final Estimator selected for the stacking architecture was a smaller LightGBM

TABLE V. DATA INGESTION AND CONTROLLED SAMPLING PARAMETERS FOR EMBER 2024

Parameter	Value/Description	Methodological Rationale
Data Source	EMBER 2024 Static Features (JSONL).	Standardized, large-scale PE feature set.
Features Extracted	"histogram" features only	Optimization for memory efficiency via fixed-length, core static features.
Feature Data Type	Numpy float32	Critical compression technique to minimize feature memory footprint.
Label Data Type	Numpy int8	Optimization for binary classification target.
Batch Loading Limit	15,000 samples	Empirically derived threshold to prevent MemoryError during ingestion.
Total Training Samples	200,000 (Maximum Cap)	Ensures stable model training within 16 GB RAM constraints.
Total Testing Samples	80,000 (Maximum Cap)	Guarantees a stable and representative test evaluation set.

model. The efficiency of LightGBM minimizes the additional computational overhead of this final synthesis step.

The integration process involves collecting the predictions generated by the three base learners. The LightGBM classifier receives these predictions as input after they are processed as meta-features. These meta-features are used to train the final model, which learns the optimal way to integrate the strengths and weaknesses of the three base algorithms to produce a synthesized classification decision that typically performs effectively.

The architectural components of the implemented Stacking Ensemble are summarized in Table VI.

IV. EXPERIMENTAL SETUP AND EVALUATION

This section outlines the application of the framework to the experimental environment, details the comprehensive evaluation protocol, and presents the final performance metrics achieved by the Stacking Ensemble model.

A. Hardware and Computational Environment Specification

All the experiments, such as feature engineering, data ingestion, model training, and evaluation, were performed on a system that was rigorously limited to 16 GB RAM and used CPU-only processing without GPU acceleration. The pipeline was optimized for this constrained hardware by using Numpy float32 compression for data structures, memory-limited loading, and gradient boosting models for effective CPU operation.

B. Performance Evaluation

The model's performance was measured against the 80,000-sample test set using a comprehensive protocol designed for operational cybersecurity assessment.

1) Standard binary classification metrics: The model's predictive capability was quantitatively assessed using a comprehensive set of standard binary classification metrics:

a) Accuracy: The overall proportion of correct classifications.

b) Precision: The ratio of correctly identified positive cases (Malware) to all cases predicted as positive.

c) Recall (True positive rate): The ratio of correctly identified positive cases to all actual positive cases.

d) F1-Score: The harmonic mean of Precision and Recall.

The F1-Score was specifically prioritized as a key performance indicator. This metric is crucial for dependable detection systems because it offers a strong performance indicator that achieves a balance between the need to reduce false alarms (Precision) and the need to reduce missed threats (Recall), making it ideal for operational evaluation in a real-world malware detection scenario.

2) Advanced assessment: Evaluation also included protocols to assess performance independent of classification thresholds and against potential real-world data skew:

a) ROC curve and AUC score: The Receiver Operating Characteristic (ROC) curve was generated to evaluate performance across all possible classification thresholds, with the Area Under the Curve (AUC) serving as a robust, threshold-independent measure of class separability.

b) Precision-Recall curve: The Precision-Recall (PR) curve was mandated for evaluation, as it is superior to the ROC curve in scenarios where minimizing False Negatives (FN) is paramount, focusing specifically on the performance of the positive class.

3) Error analysis: Confusion matrix: The Confusion Matrix was employed for error analysis, quantifying the specific distributions of True Positives (TP), True Negatives (TN), False Positives (FP), and FN. This is vital for operational risk assessment, allowing for objective interpretation of the model's tolerance for security risk versus user inconvenience.

V. RESULTS AND DISCUSSION

A. Classification Performance Results

The final performance of the Stacking Ensemble Classifier on the 80,000-sample test set yielded an overall accuracy of **86.99%**. The detailed classification results across both classes (0: Benign; 1: Malware) are presented in Table VII.

The AUC score, derived from the ROC curve illustrated in Fig. 1, reached 0.9473. This high value signifies the model's strong ability to separate the two classes (Benign vs. Malware) across all possible classification thresholds, providing a robust, threshold-independent measure of overall model quality. The ROC curve plots the True Positive Rate against the False Positive Rate, visually confirming this strong separability.

Detailed error analysis was performed using the confusion matrix that is illustrated in Fig. 2.

TABLE VI. STACKING ENSEMBLE ARCHITECTURE COMPONENTS

Component	Primary Role/Advantage	Key Methodological Constraint
LightGBM model (Base Learner)	Quick training via histogram boosting, minimal memory footprint.	Limited tree depth/estimators to manage RAM usage.
CatBoost model (Base Learner)	Robustness to overfitting, strong classification performance.	Limited tree depth/estimators to manage RAM usage.
XGBoost model (Base Learner)	High generalization capability and efficiency on tabular data.	Limited tree depth/estimators to manage RAM usage.
Smaller LightGBM model (Meta-Learner)	Synthesis of meta-features for final prediction, efficient final layer.	Optimized to avoid late-stage memory bottlenecks.

TABLE VII. DETAILED CLASSIFICATION RESULTS

	0 (Benign)	1 (Malware)	Accuracy	Macro Avg.	Weighted Avg.
Precision	0.86	0.88		0.87	0.87
Recall	0.90	0.84		0.87	0.87
F1-Score	0.88	0.86	0.87	0.87	0.87
Support	42,046	37,954	80,000	80,000	80,000

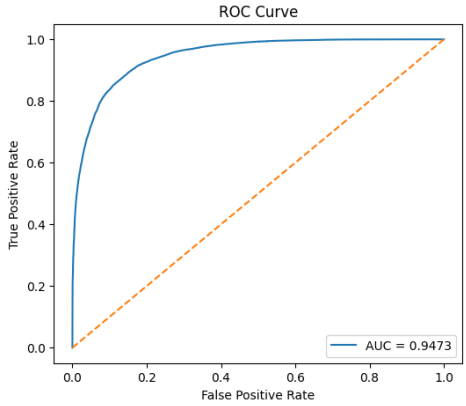

Fig. 1. ROC Curve.

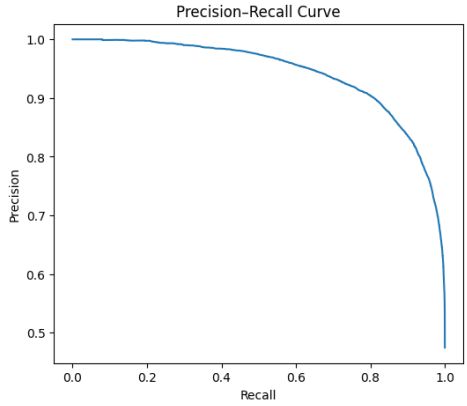

Fig. 3. Precision-Recall curve.

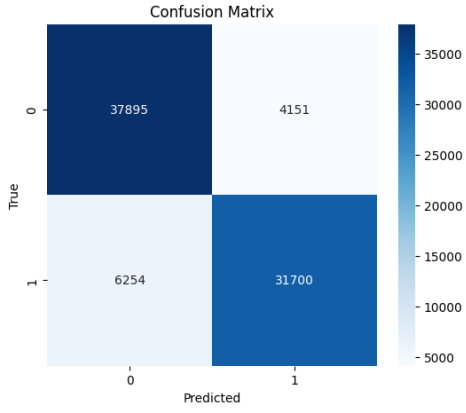

Fig. 2. Confusion matrix for the stacking ensemble classifier.

- TN: 37,895 (Correctly identified Benign files)
- FP: 4,151 (Benign files incorrectly flagged as Malware)
- FN: 6,254 (Malware files incorrectly identified as Benign)
- TP: 31,700 (Correctly identified Malware files)

Furthermore, the generation of the PR curve, illustrated in Fig. 3, was a mandatory evaluation component. This curve, which plots Precision against Recall, is methodologically superior to the ROC curve for practical deployment, particularly in real-world scenarios that often face severe class imbalance, as it specifically focuses on maximizing the performance of the positive class (Malware detection), where minimizing missed threats (FN) is paramount.

B. Proposed Framework Addresses the Identified Gaps

The framework's achieved accuracy of 86.99% must be interpreted within the context of its operational constraints. While prior work utilizing DL or hybrid analysis reports accuracy exceeding 98%, those studies typically assume access to GPU acceleration or fail to quantify the memory footprint required for such performance. By utilizing the EMBER 2024 dataset, this research evaluates the model against over 3.2 million files—a scale significantly larger than the datasets used in many papers reporting $> 95\%$ accuracy.

Our framework fills the identified gap by providing a memory-efficient data loader and a stacking ensemble that

maintains a high AUC of 0.9473 without GPU support. This demonstrates that even when accuracy is lower than "lab-environment" results, the model remains scientifically robust and operationally superior for real-world deployment on CPU-only, memory-limited systems.

1) Impact of sample capping on model capacity: The decision to cap training at 200,000 samples represents a deliberate methodological trade-off between peak predictive capacity and hardware-bound feasibility. While increasing the sample size toward the 3.2M limit would likely improve the accuracy and F1-score beyond the reported 86.99%, such an increase would lead to a MemoryError on standard 16 GB RAM systems without GPU support. By using stratified sampling rather than a simple temporal or random cut, the framework achieves a high AUC of 0.9473. This proves that a carefully curated, smaller subset can still capture the core "histogram" features necessary for effective static detection in resource-constrained environments.

C. The Performance Gap

The delta between our observed AUC (0.9473) and the published EMBER baselines is an intentional trade-off resulting from our 'Privacy and Resource' constraints. Standard EMBER baselines rely heavily on 'String' features and 'Import/Export' tables, which are highly predictive but computationally expensive to parse and can be easily obfuscated by packers. By restricting our input to raw byte-histograms, we eliminate the parsing overhead and increase resilience against structural obfuscation, albeit at the cost of the 5% AUC gap. This gap represents the 'Complexity Tax' required to ensure that the system remains functional on legacy 16GB RAM hardware without crashing during the feature extraction phase.

VI. Conclusion

This research successfully developed a CPU-optimized malware detection framework using the EMBER 2024 dataset, demonstrating that high-performance security is achievable without specialized GPU hardware. While recent literature often reports detection rates exceeding 98%, these metrics are frequently produced in resource-rich environments using deep learning models that are difficult to deploy on constrained endpoints. Our framework, which achieved an accuracy of 86.99% and an AUC of 0.9473, provides a critical analytical benchmark: it proves that optimized gradient-boosting ensembles can bridge the gap between theoretical accuracy and operational feasibility.

The synthesis of LightGBM, CatBoost, and XGBoost yields a robust classification boundary that remains stable even when training capacity is capped to meet strict hardware limits. The performance achieved here suggests that for real-world IoT and endpoint security, a "resource-first" design philosophy may be more effective than "accuracy-first" deep learning approaches that suffer from high latency and memory overhead. Our results indicate that the slight reduction in accuracy compared to state-of-the-art DL models is a justifiable trade-off for the ability to process millions of samples on commodity hardware.

Future work should investigate if this performance gap can be narrowed through advanced sampling strategies, such as temporal or active learning, to maximize the utility of the 200,000-sample training cap. Additionally, evaluating the framework's resilience against adversarial obfuscation will further define its role as a viable, low-cost alternative to signature-based and deep-learning detection systems in modern cybersecurity infrastructures.

Funding

This study was funded by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [GRANT No. KFU260954].

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [GRANT No. KFU260954]. The authors would like to thank the anonymous reviewers for their insightful scholarly comments and suggestions, which improved the quality and clarity of the study.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Y. Song, D. Zhang, J. Wang, Y. Wang, Y. Wang, and P. Ding, "Application of deep learning in malware detection: a review," *Journal of Big Data*, vol. 12, no. 1, p. 99, 2025.

[2] G. SHAN and N. SHARMA, "Malware analysis and detection using ml tools: Current state and challenges," *INTERNATIONAL JOURNAL OF ENGINEERING*, vol. 73, no. 1, pp. 371–384, 2025.

[3] N. Alyemni and M. Frikha, "Exploring machine learning in malware analysis: Current trends and future perspectives." *International Journal of Advanced Computer Science & Applications*, vol. 16, no. 1, 2025.

[4] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-Rimy, T. A. E. Eisa, and A. A. H. Elnour, "Malware detection issues, challenges, and future directions: A survey," *Applied Sciences*, vol. 12, no. 17, p. 8482, 2022.

[5] R. Kumar and S. Geetha, "Malware classification using xgboost-gradient boosted decision tree," *Adv. Sci. Technol. Eng. Syst*, vol. 5, no. 5, pp. 536–549, 2020.

[6] P. R. Kothamali and S. Banik, "Limitations of signature-based threat detection," *Revista de Inteligencia Artificial en Medicina*, vol. 13, no. 1, pp. 381–391, 2022.

[7] Y. Pratama, R. S. Munzi, A. B. Mustafa, I. L. Kharisma *et al.*, "Static malware detection and classification using machine learning: A random forest approach," *Engineering Proceedings*, vol. 107, no. 1, p. 76, 2025.

[8] D. Dhungana, A. Sapkota, S. Pokharel, S. Devkota, and B. H. Paudel, "Malware classification using static analysis approaches," *Journal of Artificial Intelligence*, vol. 6, no. 4, pp. 494–511, 2024.

[9] H. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," *ArXiv*, vol. abs/1804.04637, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4888440

[10] R. J. Joyce, G. Miller, P. Roth, R. Zak, E. Zaresky-Williams, H. Anderson, E. Raff, and J. Holt, "Ember2024-a benchmark dataset for holistic evaluation of malware classifiers," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2025, pp. 5516–5526.

[11] Kamdan, Y. Pratama, R. S. Munzi, A. B. Mustafa, and I. L. Kharisma, "Static malware detection and classification using machine learning: A random forest approach," *Engineering Proceedings*, vol. 107, no. 1, p. 76, 2025.

[12] A. S. Shatnawi, Q. Yassen, and A. Yateem, "An android malware detection approach based on static feature analysis using machine learning algorithms," *Procedia Computer Science*, vol. 201, pp. 653–658, 2022.

[13] Q. Wu, X. Zhu, and B. Liu, "A survey of android malware static detection technology based on machine learning," *Mobile Information Systems*, vol. 2021, no. 1, p. 8896013, 2021.

[14] C. K. Yuk and C. J. Seo, "Static analysis and machine learning-based malware detection system using pe header feature values," *Int. J. Innov. Res. Sci. Stud*, vol. 5, pp. 281–288, 2022.

[15] S. Akerele, N. Adebola, O. Fagbohun *et al.*, "Modern deep learning approaches for malware detection and classification," *J Artif Intell Mach Learn & Data Sci 2025*, vol. 3, no. 3, pp. 2761–2768.

[16] M. I. P. Salas and P. De Geus, "Static analysis for malware classification using machine and deep learning," in *2023 XLIX Latin American Computer Conference (CLEI)*. IEEE, 2023, pp. 1–10.

[17] A. B. Daniel and N. Patel, "Detection of malware on portable executable files using machine learning," in *2025 International Conference on Networks & Advances in Computational Technologies (NetACT)*. IEEE, 2025, pp. 1–6.

[18] M. I. Yousuf, I. Anwer, A. Riasat, K. T. Zia, and S. Kim, "Windows malware detection based on static analysis with multiple features," *PeerJ Computer Science*, vol. 9, p. e1319, 2023.

[19] A. Bensaoud, J. Kalita, and M. Bensaoud, "A survey of malware detection using deep learning," *Machine Learning With Applications*, vol. 16, p. 100546, 2024.

[20] A. Brown, M. Gupta, and M. Abdelsalam, "Automated machine learning for deep learning based malware detection," *Computers & Security*, vol. 137, p. 103582, 2024.

[21] Y. Wu, H. Zhuang, Y. Jia, and Y. Zhang, "A survey of machine learning approaches for malware detec-tion," in *Proceedings of the 2025 5th International Conference on Computer Network Security and Software Engineering*, 2025, pp. 269–273.

[22] D. G. Corlatescu, A. Dinu, M. P. Gaman, and P. Sumedrea, "Embersim: A large-scale databank for boosting similarity search in malware analysis," *Advances in Neural Information Processing Systems*, vol. 36, pp. 26 722–26 743, 2023.