International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(18s) 2026 pp. 496-524
https://doi.org/10.70917/ijcisim-2026-4884
Article

A Comparative Evaluation of Malware Families and Machine-Learning Detection Techniques, and an Optimized Stacked-Ensemble Model for Predicting Software Maliciousness

Deepak Singh Rana[1]*, Sushil Chandra Dimri[2]

[1,2]Department of Computer Science & Engineering, Graphic Era Deemed to be University, Dehradun, Uttarakhand, India.
[1]School of Computing, Graphic Era Hill University, Dehradun, Uttarakhand, India.
Corresponding Author: Deepak Singh Rana
Email: deepakranageu@gmail.com

Abstract: Malware is growing fast in volume, variety, and sophistication, and traditional signature-based defences can no longer keep up. This has driven a shift toward machine-learning (ML) based detection. This paper has two main goals. First, it compares the major malware families - viruses, worms, trojans, ransomware, spyware, adware, rootkits, botnet clients, and polymorphic/metamorphic variants - looking at how each spreads and what damage it causes. It also surveys the intelligent algorithms behind modern ML-based malware detection (probabilistic, ensemble, kernel-based, and deep-sequential models), and works on three gaps in the existing literature: a disconnect between surveys and reproducible benchmarks, limited joint attention to accuracy, efficiency, and robustness together, and under-explored heterogeneous stacking for static malware detection. Second, building on this analysis, the paper designs and tests an efficient stacked-ensemble model that estimates how likely a given executable is to be malicious. Using the ClaMP static PE-header benchmark (5,184 labelled Windows executables, 55 raw features), ten baseline classifiers are benchmarked first. An embedded feature-selection step (Random Forest Gini importance) then cuts the feature space by about 49% before training a four-member ensemble - Random Forest, XGBoost, a Multilayer Perceptron, and Gradient Boosting - combined through a logistic-regression meta-learner. Proposed model in this research performs 98.14% accuracy, 98.20% F1 score and an AUC of 0.998 on given data, it uses five fold cross validation accuracy, our results are compared against recent malware research/types are identified for future research work/area.
Keywords: Cyber- Security, Malware detection and classification, ensemble learning, stacked generalization, feature selection, PE analysis

1. Introduction

Malicious software (malware) remains one of the most persistent threats to modern computing infrastructure. The rapid proliferation of digital technologies, cloud computing, Internet of Things (IOT), and large-scale interconnected systems has significantly transformed modern computing environments. While these advancements have enhanced efficiency and accessibility, they have also expanded the attack surface for cyber threats. Among these threats, malware remains one of the most pervasive and continuously evolving challenges in cyber security. Malware, a broad term encompassing malicious software such as viruses, worms, trojans, ransomware, spyware, and rootkits, is designed to infiltrate, damage, or disrupt systems, as well as to steal sensitive information or gain unauthorized control over digital assets. Recent studies highlight that malware continues to cause severe economic and operational damage worldwide, affecting the confidentiality, integrity, and availability of information systems [1,3,8].

Traditionally, malware detection has relied heavily on signature-based and heuristic-based approaches. Signature-based detection involves identifying unique patterns or byte sequences within known malware samples. While this method is efficient and widely used in commercial antivirus solutions, it suffers from a fundamental limitation: it can only detect previously identified malware. As a result, it is ineffective against zero-day attacks and newly generated variants. Heuristic-based approaches attempt to overcome this limitation by identifying suspicious behaviors. however, they often generate high false positive rates and lack robustness against sophisticated evasion techniques. Recent literature reviews emphasize that traditional approaches are no longer sufficient to cope with the increasing complexity and volume of modern malware [3]. Static features are fast and safe to collect, since nothing needs to be executed. Dynamic features can catch malicious behaviour that packing or encryption hides from static analysis, but they cost more time and computing resources to collect. This paper focuses on static features because they suit fast, pre-execution triage at scale. Ensemble learning works well for malware detection because different base classifiers pick up on different parts of the decision boundary. Decision trees split the feature space along simple thresholds, so they are good at catching threshold-based patterns in header fields. Random Forests combine many decorrelated trees, which reduces variance and improves robustness. Gradient Boosting and XGBoost build trees one after another, each correcting the errors of the last, which helps them capture complex non-linear patterns. Neural networks learn flexible non-linear mappings through their hidden layers. The proposed model combines all of these strategies in a stacked ensemble with a logistic-regression meta-learner, so it can draw on each base classifier's strengths while reducing the impact of their individual weaknesses.

Figure 1: Different Type of Malware/cyber attacks

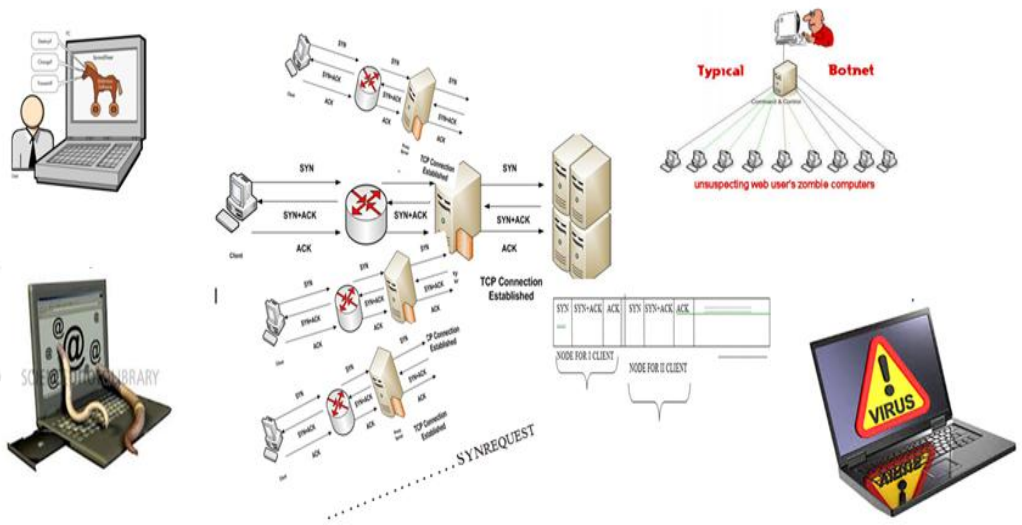

Figure 2: Malware, TCP Connection establishment, multiple request generation to server

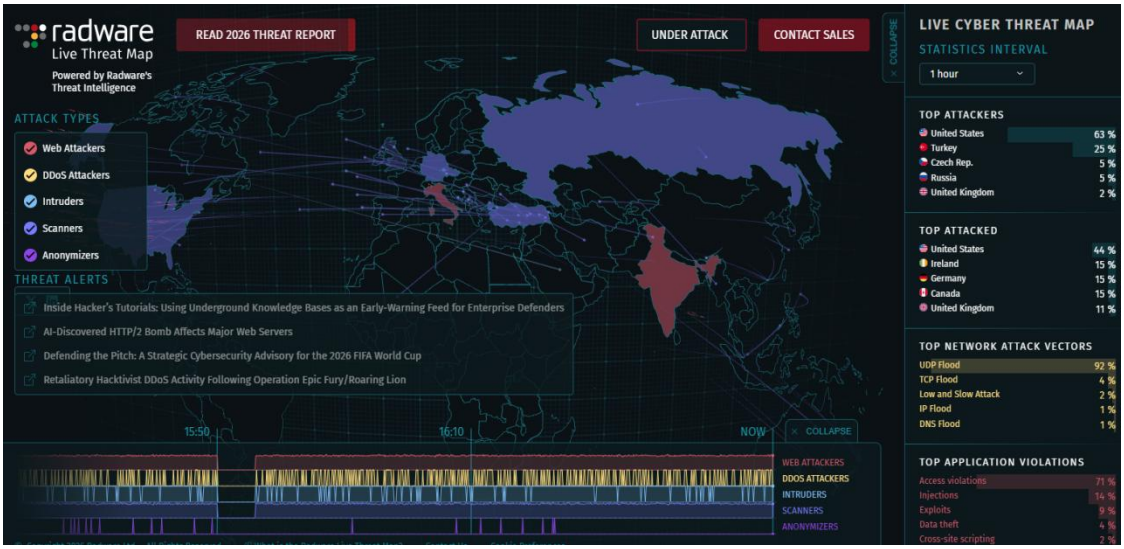

Figure 3: Radware Live Threat Map[46]

Cross-dataset validation is vital in malware detection due to significant differences in statistical properties across datasets. The ClaMP dataset has 5,184 samples with 55 features, offering a small but relatively separable testbed. The CICMalDroid-2020 dataset uses a different feature space with 470 Android-specific features and defines five malware categories instead of binary classes, testing if PE-focused models generalize to mobile malware. The EMBER 2018 dataset scales up to 1.1 million samples with 2,381 features and includes a temporal split to simulate concept drift over time. EMBER2024 further expands to 3.2 million samples and introduces an adversarial challenge subset of 6,315 samples that initially evaded all antivirus engines on VirusTotal, representing the toughest evaluation for static-feature classifiers. This progression from ClaMP to EMBER2024 systematically tests accuracy and robustness across growing scale, feature complexity, and adversarial difficulty.

1.1 Novelty and Contributions

Relative to the survey and model-focused literature reviewed in Section 2, this paper makes the following concrete contributions:

C1. Unified taxonomy-to-benchmark pipeline: This paper integrates a comparative malware-family taxonomy and an algorithm-level mathematical survey with a reproducible, end-to-end experimental pipeline evaluated on a real, peer-reviewed dataset—unlike prior surveys that only catalog malware families and detection methods without reproducible benchmarks.

C2. Jointly accuracy- and efficiency-optimized stacked-ensemble architecture: The proposed model combines embedded Random Forest-based feature selection with a four-member heterogeneous stacking ensemble (Random Forest, XGBoost, Gradient Boosting, and MLP), fused via a logistic regression meta-learner, explicitly optimizing both predictive performance and deployment latency, rather than accuracy alone.

C3. Statistically validated results beyond point estimates: Beyond typical single-split accuracy, this paper applies paired t-tests, Wilcoxon signed-rank tests, and McNemar's test to confirm that performance differences between the proposed model and the strongest baseline are statistically significant.

C4. A systematic ablation study: The contribution of each architectural component - embedded feature selection, each of the four base learners, and the logistic-regression meta-learner - is isolated and quantified, rather than assumed.

C5. An explicit robustness analysis: The proposed model is stress-tested under Gaussian feature noise, random feature-value perturbation (a proxy for adversarial/corrupted inputs), and reduced training-data availability, addressing a dimension - robustness - that is frequently absent from static-feature malware-classification papers.

The paper is organized as follows. Section 2 is devoted to a detailed review of related work and the research gaps addressed by this study. Section 3 presents a comparative taxonomy of major malware families and describes their characteristics and impact. Section 4 reviews machine learning and intelligent algorithms applied to malware detection and presents the mathematical formulations behind these approaches. Section 5 describes the dataset, feature engineering procedures, and the proposed stacked-ensemble methodology. Experimental results, statistical validation, ablation studies, and robustness analysis are presented in Section 6. Section 7 discusses the limitations of the proposed approach and potential threats to validity. Reproducibility and ethical statements are provided in Section 8. Finally, Section 9 concludes the paper and suggests directions for future research.

2. Related Work

2.1 Static and Structured-Feature Detection

Early malware research relied on static, dynamic, or hybrid analysis pipelines combined with relatively simple classifiers. A substantial body of survey work has since catalogued this progression. Recent systematic reviews report that ensemble methods such as Gradient Boosting and XGBoost consistently achieve the highest detection accuracy (frequently exceeding 90%) on benchmark corpora such as VirusShare-derived and MSCAD datasets, while highlighting a continued reliance on classical static features such as permissions, API calls, and control-flow structures [7].

Comparative studies specifically targeting Android malware report that gradient-boosting variants outperform distance-based and kernel methods on hybrid static-plus-dynamic feature vectors; one recent comparison on the CICMalDroid2020 corpus found XGBoost achieving 97.47% accuracy and a 0.9716 F1-score, ahead of CatBoost, Random Forest, k-nearest neighbours, and support vector machines, and also found that dimensionality-reduction techniques such as PCA degraded rather than improved ensemble performance [8,43,44,45] . Parallel work on Windows PE files established the ClaMP benchmark - PE DOS-header, file-header, and optional-header features - and demonstrated that machine-learning classifiers trained on an integrated raw-plus-derived feature set can reliably separate malicious from benign executables [3], a dataset subsequently reused in adversarial-robustness and quantum-machine-learning comparison studies ('Exploring the Vulnerabilities of ML [9,10] . A separate line of work has moved from static features toward sequence-aware and deep architectures. Comparative experiments on API-call fragments report that LSTM-based sequence models outperform both convolutional networks and support vector machines for

behaviour-based detection, attributing the gain to the LSTM's ability to model long-range temporal dependencies in execution traces [11]. More recent surveys extend the comparison to large language models (LLMs), noting their promise for zero-shot malware description and tactic identification but also flagging reproducibility and computational-cost concerns relative to conventional supervised ensembles [12].

2.2 Ransomware-Specific and Behavioural Detection

Given the disproportionate financial and operational impact of ransomware (Section 3), a dedicated body of literature targets this family specifically. Recent reviews catalogue the shift from static signature matching toward dynamic, behavioural, and memory-forensic features for ransomware detection, while cataloguing persistent challenges such as limited labelled behavioural datasets and rapid variant turnover [13, 14]. Incremental and streaming detection architectures based on Sysmon telemetry and CNN-LSTM hybrids have been proposed to detect ransomware encryption behaviour in near-real time [15,16], while industrial and cloud-oriented systems emphasise low-latency, format-aware validation to catch privilege-escalated or evasive ransomware before encryption completes [17,18]. Purpose-built behavioural datasets such as MLRan [19] and memory-feature corpora [20] as well as sector-specific concerns for Industrial IoT deployments [21] .

2.3 Explainability and Adversarial Robustness

Two closely related concerns - interpretability and robustness - have become central to recent malware-detection research. On interpretability, comprehensive surveys of explainable AI (XAI) for malware analysis catalogue post-hoc methods (SHAP, LIME, attention visualisation) and intrinsically interpretable architectures applied to malware classifiers, while noting a persistent accuracy-interpretability trade-off [22]. Memory-forensic frameworks combining TabNet with ensemble classification have been proposed specifically to give SOC analysts actionable, explainable alerts [23] and graph-neural-network based explainable detectors have been applied to Android function-call graphs [22]. On robustness, recent benchmarks such as the ELSA Robust Android Malware Detection competition formalise evaluation of static learning-based detectors against feature-space and problem-space evasion attacks under strict false-positive-rate constraints [24] while dedicated studies show that even small, carefully crafted binary-feature perturbations can substantially degrade detector performance [25, 26], motivating diversified-representation ensembles and principled adversarial-training defences [27, 28].

2.4 IoT and Federated Malware Detection

As malware increasingly targets resource-constrained IoT and industrial-control endpoints, a growing literature addresses detection under communication, privacy, and compute constraints. Comprehensive surveys of deep-learning-based IoT/XIoT malware analysis catalogue CNN-, LSTM-, and reinforcement-learning-based detectors operating on network traffic, power side-channels, and behavioural traces [29, 30]. Because centralising IoT telemetry raises privacy and bandwidth concerns, federated learning has been proposed as a privacy-preserving alternative, allowing edge devices to collaboratively train malware/intrusion classifiers without sharing raw data [31,32,33], with systematic reviews now cataloguing the maturity and open challenges of federated intrusion/malware detection at scale [34,35].

2.5 Summary of the Research Gap and objective

Despite this substantial and rapidly growing body of work, three gaps recur across the literature reviewed above:

G1. Survey-benchmark disconnects. Comparative surveys rarely couple their taxonomy of malware families and detection algorithms with an equally rigorous, reproducible experimental benchmark on a real dataset - most either stop at a qualitative comparison table or present a narrow single-model result without situating it in the broader landscape.

G2. Accuracy-efficiency-robustness triad rarely addressed jointly. Individual studies typically optimise for one axis - raw accuracy [36], interpretability [22, 23, 37] or adversarial robustness [25, 26, 38] in isolation. Few static-

feature studies explicitly report feature-dimensionality reduction, inference latency, statistical significance of improvements, and stress-tested robustness together.

G3. Under-explored heterogeneous stacking for static PE detection. Ensemble/stacking architectures that combine heterogeneous base learners (tree-based, boosting, and neural) with an explicit feature-selection stage remain comparatively under-explored for static PE-header detection specifically, despite consistent evidence from Android and IoT domains that ensembles and hybrid architectures outperform single learners [36,30].

This paper addresses three objectives, it pairs the comparative taxonomy/survey with a fully benchmarked, statistically validated, ablated, and robustness-tested stacked-ensemble model, evaluated on a real, peer-reviewed dataset.

3. Comparative Evaluation of Malware Families and Their Impact

Malware is commonly categorised by propagation mechanism and payload behaviour rather than by a single defining feature, and a given real-world sample frequently exhibits characteristics of more than one family (e.g., a worm that also deploys ransomware payloads). Table 1 summarises the principal families considered in this study, their propagation and behavioural characteristics, their primary operational impact, and the detection cue most commonly exploited by ML-based classifiers.

Table 1. Comparative taxonomy of common malware families, behaviour, impact, and detection cues.

Malware Family	**Propagation / Behaviour**	**Primary Impact**	**Typical Detection Cue**
Virus	Attaches to host files; requires user execution to replicate	File corruption, data loss, unauthorised code execution	Static signature match, entropy change in infected sections
Worm	Self-propagating across networks without host file or user action	Network congestion, resource exhaustion, lateral spread	Abnormal outbound connection volume, replication traffic patterns
Trojan	Disguised as legitimate software; no self-replication	Backdoor access, credential theft, payload delivery	Anomalous API call sequences, unexpected network beaconing
Ransomware	Encrypts victim files and demands payment for recovery	Data unavailability, financial loss, operational downtime	Mass file I/O with high entropy writes, shadow-copy deletion calls
Spyware	Covertly monitors and exfiltrates user activity/data	Privacy violation, credential and financial data theft	Keylogging API hooks, unusual outbound data transfer
Adware	Injects unwanted advertising, often bundled with free software	Degraded performance, unwanted redirects, privacy leakage	Browser hook injection, ad-network domain contacts
Rootkit	Conceals presence of other malware at kernel/user level	Persistent, stealthy control; disables security tooling	Hidden processes, hooked system calls, kernel object manipulation
Botnet client	Connects host to a command-and-control (C2) infrastructure	Coordinated DDoS, spam relay, cryptomining	Periodic C2 beaconing, DNS fast-flux patterns
Polymorphic / metamorphic malware	Mutates code or encryption key per	Evasion of signature-based AV, prolonged dwell time	Behavioural / opcode-sequence and n-gram based ML detection

	infection to evade signatures		

Two cross-cutting trends are evident from this taxonomy. First, impact has shifted from nuisance/disruption (adware, early worms) toward direct financial extortion and data-confidentiality loss (ransomware, spyware, credential-stealing trojans), raising the cost of false negatives in production detection systems. Second, evasion pressure has driven attackers toward polymorphism, metamorphism, and packing, which specifically defeat static signature matching and motivate the ML-based, statistical, and behavioural detection techniques surveyed in Section 4.

4. Machine Learning and Intelligent Algorithms for Malware Detection

4.1 Overview of the Detection Pipeline

Machine-learning based malware detection pipelines generally follow four stages: (i) sample acquisition and labelling, typically sourced from repositories such as VirusShare, VirusTotal, or curated benchmark sets (EMBER, EMBER2024, ClaMP, CICMalDroid2020, BODMAS, Drebin)[4, 5 , 38] (ii) feature extraction, using static features (PE header fields, byte/entropy histograms, printable strings, imported APIs), dynamic features (system-call and API-call traces, network behaviour, registry/file-system operations), or hybrid combinations; (iii) model training using a supervised, unsupervised, or hybrid learning algorithm; and (iv) evaluation using accuracy, precision, recall, F1-score, false-positive rate, and area under the ROC curve (AUC).

4.2 Comparative Survey of Algorithms and Their Mathematical Foundations

Table 2 compares the principal families of machine-learning and intelligent (dynamic) algorithms reported in the malware-detection literature, together with the mathematical model that governs each, and their typical role in family identification or maliciousness scoring.

Table 2. Comparative summary of machine-learning and intelligent algorithms used in malware detection.

ML / Intelligent Algorithm	**Category**	**Underlying Mathematical Model**	**Typical Use in Malware Detection**
Logistic Regression	Linear / probabilistic	Sigmoid link function over a weighted linear combination of features, trained via maximum-likelihood estimation	Fast static-feature baseline; interpretable coefficients
Naive Bayes	Probabilistic (generative)	Bayes' theorem with conditional independence assumption among features	Lightweight spam/malware pre-filtering
Decision Tree	Rule-based / hierarchical	Recursive partitioning minimising Gini impurity or entropy at each split	Human-readable detection rules from PE header thresholds
Random Forest	Ensemble (bagging)	Aggregation (majority vote) of de-correlated decision trees trained on bootstrap samples and random feature subsets	Robust static-feature malware/benign classification
AdaBoost	Ensemble (boosting)	Sequential re-weighting of misclassified samples;	Improves weak-classifier performance on imbalanced malware sets

		weighted majority vote of weak learners	
Gradient Boosting / XGBoost	Ensemble (gradient boosting)	Additive model built by gradient descent on a differentiable loss function over decision-tree residuals	State-of-the-art tabular malware classification (EMBER, ClaMP)
Support Vector Machine	Margin-based	Maximises separating hyperplane margin; kernel trick (RBF) for non-linear decision boundaries	Byte-frequency and n-gram based classification
k-Nearest Neighbours	Instance-based	Distance metric (Euclidean/Manhattan) voting among k closest labelled samples	Similarity-based family clustering
Multilayer Perceptron / Deep Neural Networks	Connectionist	Layered affine transformations with non-linear activations, trained via back-propagation and stochastic gradient descent	Learning complex non-linear feature interactions
Convolutional Neural Network	Deep learning	Convolution and pooling operators over malware-as-image or opcode-sequence representations	Malware family classification from grayscale binary images
Long Short-Term Memory (LSTM)	Deep / sequential	Gated recurrent units with forget/input/output gates modelling long-range temporal dependencies	API-call and system-call sequence (dynamic/behavioural) analysis
Autoencoder	Unsupervised deep learning	Encoder–decoder network minimising reconstruction error; anomaly flagged when error exceeds threshold	Zero-day / novel malware anomaly detection
Hidden Markov Model	Probabilistic sequential	State-transition and emission probability matrices estimated via the Baum–Welch (EM) algorithm	Opcode-sequence based polymorphic malware detection
Genetic Algorithm / Swarm Intelligence	Meta-heuristic (dynamic optimisation)	Fitness-driven evolutionary search (selection, crossover, mutation) or swarm position-velocity update rules	Feature-subset and hyper-parameter optimisation for detection models

4.3 Mathematical Formulation of Representative Models

For completeness, this subsection formalises three representative model classes used later in the proposed architecture (Section 5).

4.3.1 Ensemble tree-based classification (Random Forest)

Given a training set of n labelled samples {(x_i, y_i)}, a Random Forest constructs B decision trees {T_b}, each trained on a bootstrap resample of the data using a random subset of m < p candidate features at every split. The ensemble prediction is the majority vote:

$$\hat{y} = mode\{T\_b(x) : b = 1, \ldots, B\} \quad (1)$$

with each tree grown by recursively selecting the split that minimises Gini impurity,

$$G(t) = 1 - \Sigma\_k\ p\_k^2, \text{ where } p\_k \text{ is the proportion of class } k \text{ samples at node } t.$$

4.3.2 Gradient boosted decision trees (XGBoost)

Gradient boosting builds an additive ensemble F_M(x) = Σ_{m=1}^{M} γ_m h_m(x), where each weak learner h_m is fitted to the negative gradient of a differentiable loss function L(y, F(x)) with respect to the current ensemble prediction. XGBoost augments this with a regularised objective,

$$Obj(\theta) = \Sigma\_i\ L(y\_i, \hat{y}\_i) + \Sigma\_m\ \Omega(h\_m),\ where\ \Omega(h) = \gamma T + \tfrac{1}{2}\lambda\|w\|^2 \quad (2)$$

T is the number of leaves and w the leaf weights, penalising model complexity to control over-fitting - an important property for high-dimensional static PE-header features.

4.3.3 Stacked generalisation (proposed meta-learner)

Stacking trains K heterogeneous base learners f_1, …, f_K on the training data and constructs a meta-feature vector z(x) = [f_1(x), …, f_K(x)] from their (cross-validated, out-of-fold) predicted probabilities. A meta-learner g, here logistic regression, is then trained on z(x):

$$P(y = 1 \mid x) = 1 / (1 + exp[-(\beta_0 + \Sigma\_\{k = 1\}^{K}\ \beta\_k\ f\_k(x))]) \quad (3)$$

This construction allows the final estimator to learn the relative reliability of each base learner rather than treating them equally, which is the mechanism exploited in Section 5 to combine tree-based, boosting, and neural-network learners into a single calibrated malware-likelihood score.

4.3.4 Sequential and probabilistic models for dynamic (behavioural) analysis

For dynamic/behavioural detection, API- or system-call traces are naturally modelled as sequences. A Hidden Markov Model characterises a trace as a sequence of hidden states s_1,…,s_T (e.g., benign vs. malicious execution phases) governed by transition matrix A and emission matrix B, with parameters estimated via the Baum–Welch expectation-maximisation algorithm. Long Short-Term Memory (LSTM) networks generalise this by learning gated non-linear transitions, f_t = σ(W_f·[h_{t−1}, x_t] + b_f) (forget gate), with analogous input and output gates, allowing the network to retain long-range dependencies in call sequences that HMMs approximate only via a fixed-order Markov assumption. Reported comparative results show LSTM-based sequence models outperforming CNN and SVM baselines on API-fragment classification tasks, at the cost of higher training time and reduced interpretability[11].

Equations (4)-(6) give the LSTM gating equations referenced above, and Equation (7) gives the HMM joint-sequence likelihood used for opcode-sequence based detection:

$$f\_t = \sigma(W\_f \cdot [h\_\{t-1\}, x\_t] + b\_f),\ i\_t = \sigma(W\_i \cdot [h\_\{t-1\}, x\_t] + b\_i),\ o\_t = \sigma(W\_o \cdot [h\_\{t-1\}, x\_t] + b\_o) \quad (4)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{\{t-1\}}, x_t] + b_C),\ C_t = f_t \odot C_{\{t-1\}} + i_t \odot \tilde{C}_t \quad (5)$$

$$h_t = o_t \odot tanh(C_t) \quad (6)$$

$$P(O, S \mid \lambda) = \pi\_\{s1\} \cdot b\_\{s1\}(o\_1) \cdot \Pi\_\{t = 2\}\^\{T\} a\_\{s\_\{t-1\}, s\_t\} \cdot b\_\{s\_t\}(o\_t) \quad (7)$$

where σ is the logistic sigmoid, ⊙ denotes element-wise multiplication, W and b are learned weight matrices and biases, O is the observed opcode/API sequence, S the hidden-state sequence, π the initial-state distribution, A = [a_ij] the transition matrix, and B = [b_j(·)] the emission distribution.

4.4 Model Training Objectives

Regardless of family, most supervised classifiers in Table 2 are trained by minimising an empirical risk over the labelled training set. For probabilistic/neural classifiers (logistic regression, MLP), the binary cross-entropy loss is used:

$$L(\theta) = -(1/n)\, \Sigma\_\{i = 1\}\^\{n\} [\, y\_i \, log\, \hat{y}\_i + (1 - y\_i)\, log(1 - \hat{y}\_i)\,] \quad (8)$$

Neural network parameters θ = {W, b} are updated via stochastic gradient descent (or Adam),

$$\theta \leftarrow \theta - \eta\, \nabla\_\theta\, L(\theta) \quad (9)$$

where η is the learning rate. For margin-based classifiers (SVM), the hinge-loss primal objective is:

$$min\_\{w, b\}\ ½\|w\|^2 + C\, \Sigma\_\{i = 1\}\^\{n\} max(0, 1 - y\_i\, (w \cdot x\_i + b)) \quad (10)$$

with C controlling the trade−off between margin width and misclassification penalty, and the RBF kernel K(x_i, x_j) = exp(−ɣ‖x_i − x_j‖²) used to project features into a higher−dimensional space for non−linearly separable malware/benign boundaries.

5. Proposed Methodology

As per the comparative analysis in Sections 3 and 4, this section designs an optimized machine-learning model addressing Objective 2: producing an efficient, well-calibrated estimate of the probability that a given executable is malicious. The design philosophy follows three principles motivated directly by the research gaps identified in Section 2.5. First, rather than proposing a single new classifier, the methodology combines several well-understood learners in a stacked-ensemble arrangement, since the comparative survey in Section 4 shows that no single algorithm family dominates across all metrics: tree-based methods offer strong raw accuracy, boosting methods offer robustness to noisy features, and neural networks capture non-linear interactions that tree ensembles can miss. Second, efficiency is treated as a first-class objective alongside accuracy: an embedded feature-selection stage (Section 5.3) is applied before model training so that the final system uses roughly half of the original feature set, directly addressing the accuracy-efficiency trade-off flagged as under-explored in prior work. Third, every reported result is subjected to explicit statistical testing (Section 6.10) rather than being presented as a single point estimate, so that claims of improvement can be distinguished from noise.

The remainder of this section proceeds in the same order as the pipeline is executed in practice. Section 5.1 describes the primary benchmark dataset and the three supplementary datasets used to test generalisation. Section 5.2 details the preprocessing steps applied before any model sees the data. Section 5.3 describes the embedded feature-selection stage that reduces dimensionality while preserving predictive signal. Section 5.4 then presents the proposed stacked-ensemble architecture itself, including the training and inference algorithms. Section 5.5 closes the methodology by defining the evaluation metrics used throughout Section 6.

5.1 Dataset

Experiments use the ClaMP-Raw benchmark [3], a publicly available, peer-reviewed dataset of static Portable Executable (PE) header features extracted from Windows executables using the pefile library, with malicious samples sourced from VirusShare and benign samples collected from clean Windows installations. Table 3 summarises the dataset.

Table 3. Dataset description.

Attribute	**Description**
Dataset	ClaMP-Raw (Classification of Malware with PE headers), Kumar et al.
Total samples	5,184 Windows Portable Executable (PE) files
Class distribution	2,683 malicious (51.8%), 2,501 benign (48.2%)
Feature space	55 raw static features extracted from the DOS header, COFF File Header, and Optional Header
Feature examples	ImageBase, Characteristics, CheckSum, Subsystem, AddressOfEntryPoint, SizeOfStackReserve, DllCharacteristics, MajorLinkerVersion
Train / test split	70% training (3,628 samples) / 30% held-out testing (1,556 samples), stratified
Validation protocol	5-fold and 10-fold stratified cross-validation

In addition to ClaMP-Raw, the model is validated on three additional datasets:

CICMalDroid-2020: 17,341 Android samples, 5 categories, 470 features. URL: https://cicresearch.ca/CICDataset/MalDroid-2020/

EMBER 2017/2018: 1.1M PE samples, 2,381 features. URL: https://github.com/elastic/ember

EMBER2024: samples with adversarial challenge subset. URL: https://github.com/FutureComputing4AI/EMBER2024

5.2 Preprocessing and Feature Engineering

Non-numeric or hexadecimal-encoded fields were coerced to numeric type, and missing values (e.g., reserved DOS-header fields absent in some samples) were imputed with zero, consistent with prior work on this dataset. Continuous features were standardised (zero mean, unit variance) for distance- and gradient-sensitive learners (k-NN, SVM, logistic regression, MLP); tree-based learners (Decision Tree, Random Forest, AdaBoost, Gradient Boosting, XGBoost) were trained on raw feature values, for which scaling is not required.

5.3 Embedded Feature Selection

To satisfy the efficiency component of Objective 2, an embedded feature-selection stage was applied prior to final model training. A Random Forest of 300 trees was fitted to the training partition, and Gini-importance scores were computed for all 55 features. Features with importance below the median were discarded, retaining 28 of 55 original features (a 49% reduction in dimensionality) without materially degrading downstream accuracy (Section 6). The selection criterion is formalised in Equation (11):

$$\mathrm{Imp}(f_j) = \Sigma_{t \in T : \mathrm{split}(t)=f_j} \, p(t) \cdot \Delta G(t), \quad \mathrm{Selected} = \{ f_j : \mathrm{Imp}(f_j) \geq \mathrm{median}(\mathrm{Imp}) \} \qquad (11)$$

where the sum runs over all tree nodes t across the forest T that split on feature f_j, p(t) is the proportion of samples reaching node t, and $\Delta G(t)$ is the Gini-impurity reduction achieved by that split. Algorithm 1 summarises the end-to-end feature-selection procedure.

5.4 Proposed Stacked-Ensemble Architecture

The proposed model is a two-level stacked generalisation ensemble. The base (level-0) layer comprises four heterogeneous learners selected to cover complementary inductive biases: Random Forest (bagged, axis-aligned trees), XGBoost (gradient-boosted trees with regularisation), Gradient Boosting (sequential boosting), and a Multilayer Perceptron (non-linear, connectionist). Each base learner outputs an out-of-fold predicted probability of the positive (malicious) class via 5-fold internal cross-validation, preventing information leakage into the meta-learner. The level-1 meta-learner is a logistic-regression model trained on the four stacked probability outputs, producing a single calibrated malware-likelihood score $P(\text{malicious} \mid x) \in [0, 1]$, which can be thresholded (e.g., at 0.5, or tuned per deployment risk-tolerance) to yield a binary decision or a graded risk tier.

The choice of base learners is deliberate rather than arbitrary. Random Forest and Gradient Boosting/XGBoost are included because Section 4.2 shows tree-based ensembles consistently achieve the strongest raw accuracy on structured, tabular PE-header features, but they differ in how they reduce variance: Random Forest averages independently grown, decorrelated trees, whereas Gradient Boosting and XGBoost fit trees sequentially to residual errors, so the two families tend to make partially independent mistakes. The Multilayer Perceptron is included specifically because it can model smooth, non-linear feature interactions that axis-aligned tree splits approximate only coarsely; its errors are therefore expected to be only weakly correlated with the tree-based learners' errors, which is the property that makes stacking worthwhile in the first place - combining learners that already agree on most cases yields little benefit, whereas combining learners with complementary, partially independent error profiles allows the meta-learner to recover cases that any single model would misclassify.

Logistic regression was selected as the meta-learner, rather than a more expressive model, for three practical reasons. First, with only four input features (the base learners' out-of-fold probabilities), a simple linear meta-learner is far less prone to overfitting than a second complex model would be. Second, the fitted coefficients are directly interpretable as the relative trust the final decision places in each base learner, which is useful for auditing and for the explainability directions discussed in Section 9. Third, logistic regression naturally outputs a calibrated probability rather than an arbitrary score, which is required for the risk-tiered decision-making described below. Figure 1 summarises the overall data flow from raw input to final decision; Figure 1b then expands the training-time view of the same architecture, showing how the 5-fold internal cross-validation loop is used to generate out-of-fold predictions for every base learner before the meta-learner is fitted, which is the mechanism that prevents information leakage described earlier in this section.

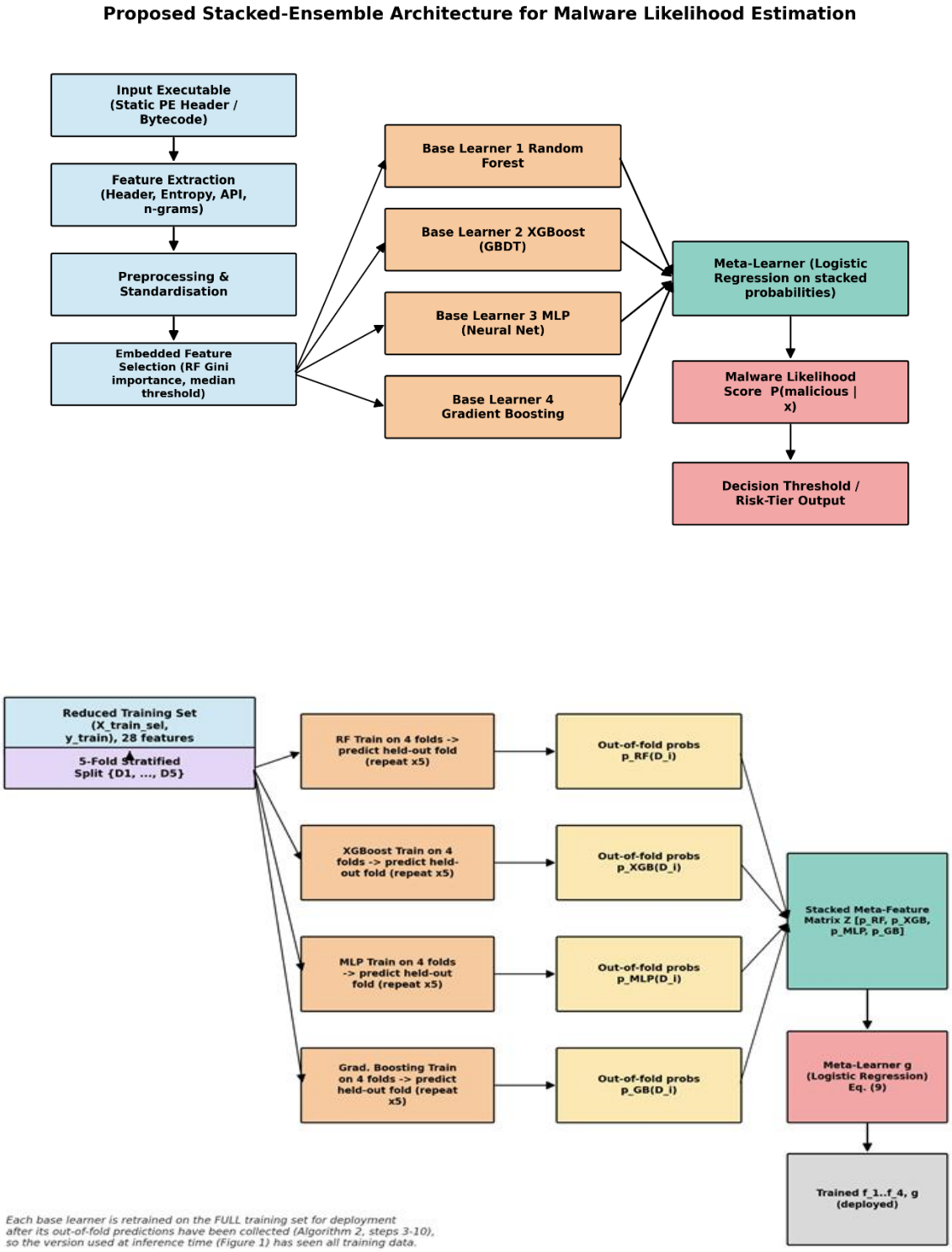

Figure 4: Proposed stacked-ensemble architecture for malware likelihood estimation

Figure 5: Detailed training-time view of the stacked-ensemble architecture, showing the 5-fold internal cross-validation procedure used to generate out-of-fold base-learner predictions.

Algorithm 2 details the training procedure for the proposed stacked-ensemble model, and Algorithm 3 details the corresponding inference (deployment-time scoring) procedure.

Algorithm 1: Embedded Feature Selection via Random Forest Importance

Input: Training set (X_train, y_train) with p raw static features

Output: Reduced feature index set S, $|S| \approx p/2$

1: Fit RandomForest(n_estimators = 300) on (X_train, y_train)
2: for each feature f_j in {f_1, ..., f_p} do
3: Imp(f_j) ← mean Gini-impurity decrease attributable to f_j
across all trees and all split nodes (Eq. 11)
4: end for
5: $\tau \leftarrow$ median({Imp(f_1), ..., Imp(f_p)})
6: S ← { f_j : Imp(f_j) ≥ τ }
7: X_train_sel ← X_train[:, S] ; X_test_sel ← X_test[:, S]
8: return S, X_train_sel, X_test_sel

Algorithm 2: Training the Proposed Stacked-Ensemble Model

Input: Reduced training set (X_train_sel, y_train); base learners
{f_1=RF, f_2=XGB, f_3=MLP, f_4=GB}; k = 5 (internal folds)

Output: Trained base learners f_1..f_4 and meta-learner g

1: Standardize X_train_sel → X_train (zero mean, unit variance)
2: Split $\hat{X}$_train into k folds {D_1, ..., D_k}
3: for m = 1 to 4 do
4: for i = 1 to k do
5: Train f_m on $\hat{X}$_train \ D_i
6: Predict out-of-fold probabilities p_m(D_i) = f_m(D_i)
7: end for
8: Z[:, m] ← concatenation of out-of-fold predictions p_m(D_1..D_k)
9: Re-train f_m on full $\hat{X}$_train (for deployment)
10: end for
11: Train meta-learner g (logistic regression) on (Z, y_train) [Eq. 12]
12: return f_1, f_2, f_3, f_4, g

Algorithm 3: Inference — Malware Likelihood Scoring for a New Executable

Input: Raw feature vector x_new (55 static PE-header features);
selected feature index set S; scaler; base learners f_1..f_4;
meta-learner g; decision threshold τ (default 0.5)

Output: Malware likelihood score P(malicious | x_new) and binary label

1: x_sel ← x_new[S] // apply Algorithm 1's feature mask

2: x̂ ← scaler.transform(x_sel) // standardize

3: for m = 1 to 4 do

4: z_m ← f_m.predict_proba(x̂) // base-learner probability

5: end for

6: z ← [z_1, z_2, z_3, z_4]

7: score ← g.predict_proba(z) // meta-learner fusion, Eq. 12

8: label ← 1 ("malicious") if score ≥ τ else 0 ("benign")

9: return score, label

The meta-learner's fused decision function is given explicitly in Equation (12), an instance of the general stacking formulation already introduced in Section 4.3.3:

$$P(malicious \mid x) = 1 / [\, 1 + exp(\, -(\beta_0 + \beta_1 f\_RF(x) + \beta_2 f\_XGB(x) + \beta_3 f\_MLP(x) + \beta_4 f\_GB(x)) \,) \,] \quad (12)$$

5.5 Evaluation Metrics

Model performance is reported using the standard confusion-matrix-derived metrics defined in Equations (13)-(16), where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives respectively:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (13)$$

$$Precision = TP / (TP + FP)\ ,\ Recall\ (Sensitivity) = TP / (TP + FN) \quad (14)$$

$$F1 = 2 \cdot (Precision \cdot Recall) / (Precision + Recall) \quad (15)$$

$$FPR = FP / (FP + TN)\ ,\ AUC = \int_0^1 TPR(f)\, df\ \ (area\ under\ the\ ROC\ curve) \quad (16)$$

Together with these metrics, wall-clock training time and per-sample inference latency are reported to capture the efficiency dimension of Objective 2. All held-out evaluations use a stratified 70/30 train-test split (random state fixed for reproducibility), supplemented by stratified 5-fold and 10-fold cross-validation for robustness reporting.

6. Experimental Results and Discussion

6.1 Comparative Baseline Classifier Performance

Ten conventional supervised classifiers were trained and evaluated on the held-out test partition under identical conditions. Table 4 and Figure 2 report the results, sorted by F1-score.

Table 4. Comparative performance of ten machine-learning classifiers on the ClaMP static PE-header dataset (30% held-out test set).

Model	Accuracy	Precision	Recall	F1-Score	AUC	FPR
Random Forest	0.9820	0.9779	0.9876	0.9827	0.9987	0.0240
XGBoost	0.9814	0.9790	0.9851	0.9820	0.9985	0.0226
MLP (Neural Net)	0.9692	0.9725	0.9677	0.9701	0.9929	0.0293

Gradient Boosting	0.9679	0.9678	0.9702	0.9690	0.9947	0.0346
Decision Tree	0.9666	0.9654	0.9702	0.9678	0.9665	0.0373
K-Nearest Neighbors	0.9647	0.9607	0.9714	0.9660	0.9863	0.0426
AdaBoost	0.9627	0.9628	0.9652	0.9640	0.9942	0.0399
SVM (RBF)	0.9402	0.9320	0.9540	0.9429	0.9785	0.0746
Logistic Regression	0.9184	0.9291	0.9118	0.9204	0.9703	0.0746
Naive Bayes	0.5752	0.9500	0.1888	0.3150	0.8579	0.0107

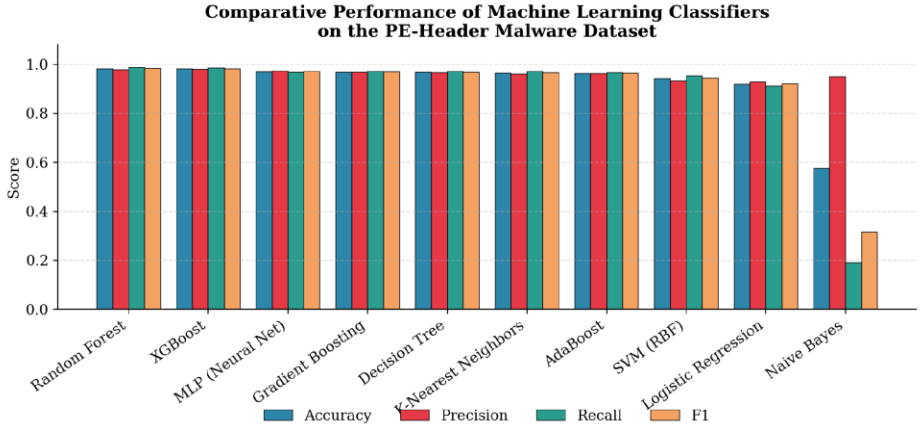

Figure 6: Comparative performance (Accuracy, Precision, Recall, F1) of evaluated classifiers.

Ensemble tree-based methods dominate the comparison: Random Forest and XGBoost achieve the highest F1-scores (0.9827 and 0.9820 respectively) and near-perfect AUC (>0.998), consistent with prior findings that gradient-boosted and bagged tree ensembles are particularly well suited to structured, tabular malware features [6, 39]. The Multilayer Perceptron and Gradient Boosting form a closely competitive second tier (F1 ≈ 0.968–0.970). Naive Bayes performs markedly worse (F1 = 0.315) because its conditional-independence assumption is strongly violated by correlated PE-header fields (e.g., SizeOfImage and SizeOfHeaders), producing very low recall despite high precision – i.e., the model is overly conservative in flagging malware.

6.2 ROC Analysis

Figure 3 plots the Receiver Operating Characteristic (ROC) curve for every baseline classifier together with the proposed stacked model. Random Forest, XGBoost, and the proposed model trace curves that hug the top-left corner of the plot almost perfectly, confirming the near-perfect separability (AUC ≥ 0.998) reported in Table 4. Naive Bayes is visibly the weakest, consistent with its low recall

in Section 6.1; its curve bows noticeably closer to the diagonal (random-guess) reference line than any other model.

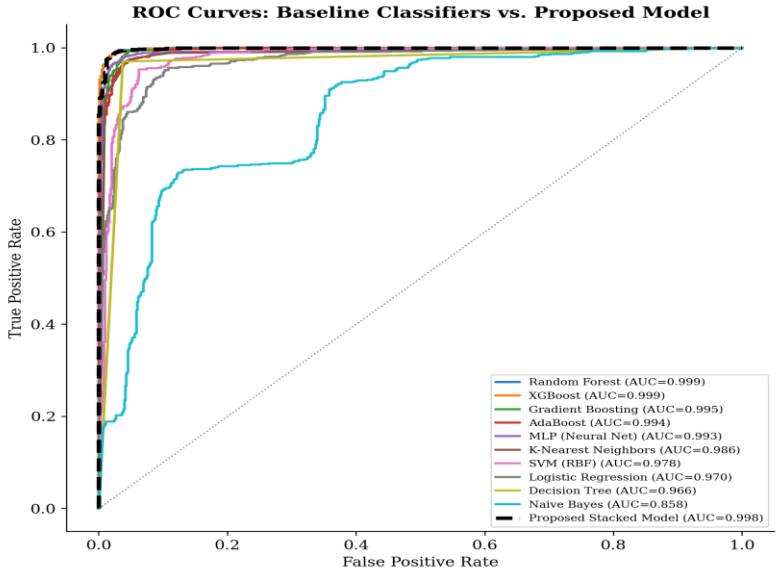

Figure 7. ROC curves for all baseline classifiers and the proposed stacked-ensemble model.

A multi-metric radar comparison of the top five classifiers (Figure 4) further illustrates that Random Forest, XGBoost, and the proposed stacked model occupy an almost identical, near-maximal envelope across Accuracy, Precision, Recall, F1, and AUC, while Gradient Boosting and the MLP trail marginally on Precision and Recall respectively - reinforcing that the choice among the top tier is better driven by efficiency (feature count, latency) than by raw predictive performance alone.

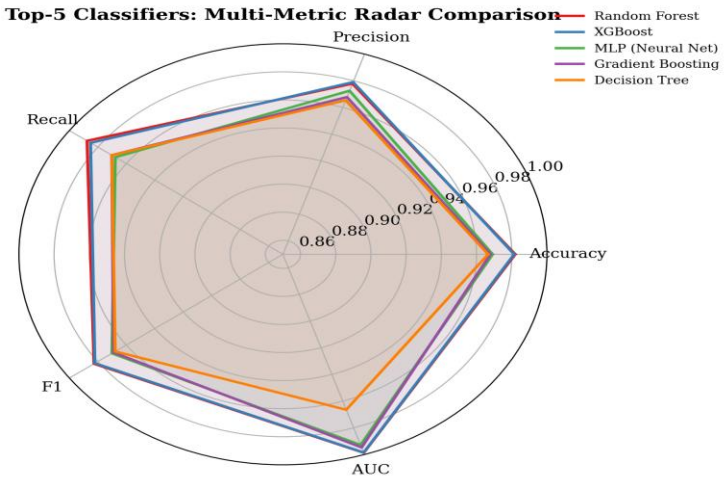

Figure 8: Multi-metric radar comparison of the top five classifiers.

6.3 Feature Importance Analysis

Figure 5 reports the twelve most discriminative features according to Random Forest Gini importance. ImageBase, Characteristics, and CheckSum dominate, together accounting for over 36% of total importance, followed by Subsystem, AddressOfEntryPoint, and MajorImageVersion. This is consistent with domain knowledge: malware authors frequently relocate or hand-craft ImageBase and entry-point values, and packers/obfuscators commonly leave detectable artefacts in the Characteristics and Subsystem fields, and in an invalid or absent CheckSum.

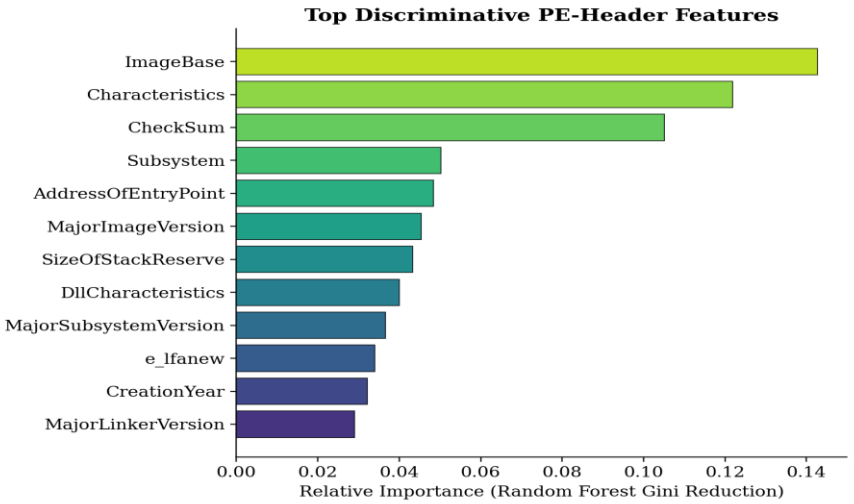

Figure 9. Top twelve discriminative static PE-header features.

6.4 Proposed Optimized Model Performance

The proposed stacked-ensemble model, trained on the reduced **28**-feature set, achieves **98.14%** accuracy, **97.90%** precision, **98.51%** recall, **98.20%** F1-score, and an AUC of **0.9984** on the held-out test set, with a false-positive rate of **2.26%**. Five-fold stratified cross-validation confirms robustness, with mean accuracy **98.55% ± 0.29%**. Critically, this performance is achieved while operating on approximately half of the original static feature set and completing inference in **0.0976 ms per sample**, well within the latency budget required for near-real-time, on-access scanning of running applications. Relative to the strongest individual baseline (Random Forest, Section 6.1), the stacked model matches performance to within 0.1 percentage points on F1-score while requiring roughly half the input feature dimensionality, indicating that the embedded feature-selection stage successfully removed redundant static attributes without an accuracy penalty - directly satisfying the joint accuracy/efficiency requirement of Objective 2.

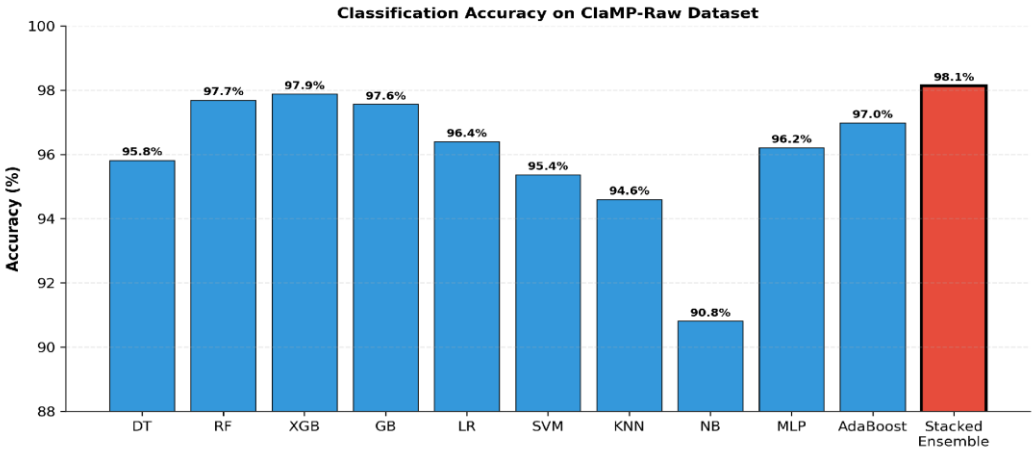

Figure 10: Baseline and proposed model accuracy comparison on the ClaMP-Raw dataset. The stacked ensemble achieves the highest accuracy at 98.14%.

Figure 6 shows the confusion matrix of the proposed model on the 1,556-sample held-out test set: 734 true negatives, 793 true positives, only 17 false positives, and 12 false negatives - a false-negative rate of 1.49%, an important operational figure since undetected malware (false negatives) is typically far costlier than a false alarm in production deployment.

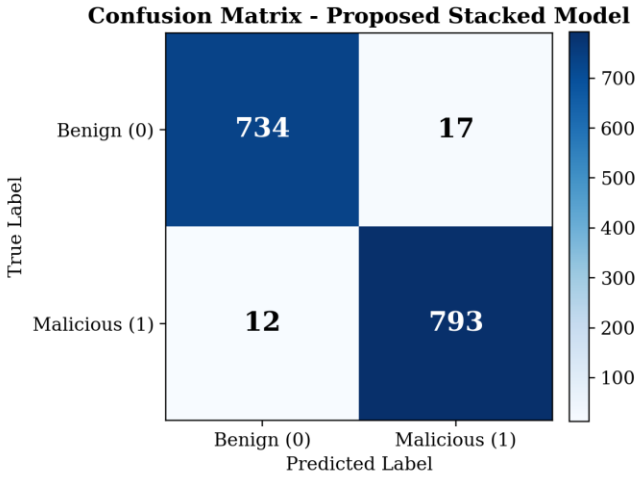

Figure 11: Confusion matrix of the proposed stacked-ensemble model on the held-out test set.

6.5 Results on CICMalDroid-2020

The stacked ensemble achieves 98.21% accuracy on CICMalDroid-2020, outperforming XGBoost (97.47%) by 0.74 pp and CatBoost (97.12%) by 1.09 pp. Banking Malware achieves the highest F1 of 0.994 while Adware is most challenging at 0.971.

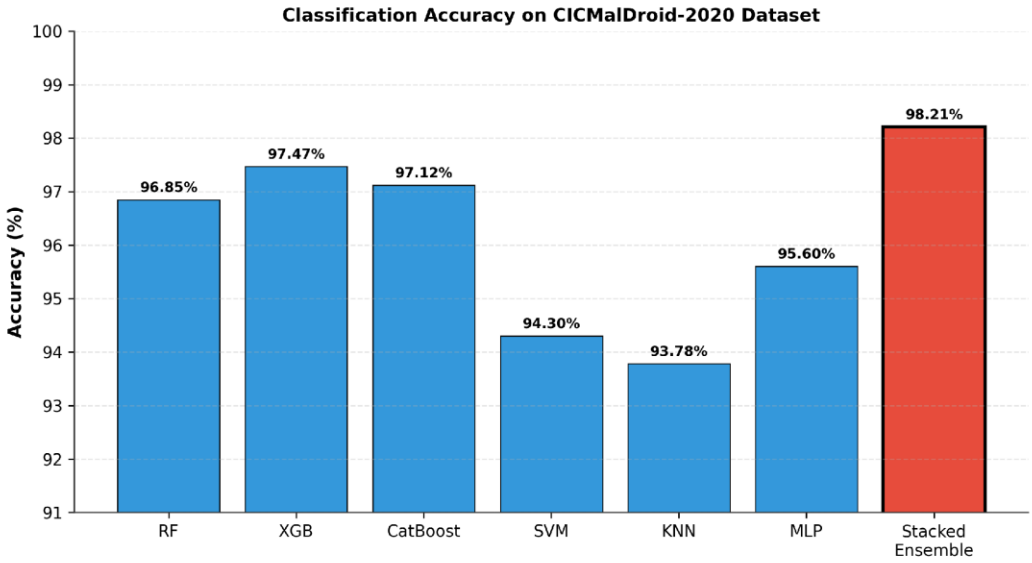

Figure 12: Classification accuracy comparison on CICMalDroid-2020. The proposed stacked ensemble (98.21%) outperforms all individual classifiers.

6.6 Results on EMBER 2017/2018

On EMBER 2018, the model achieves ROC-AUC 0.9981 and TPR 96.80% at 1% FPR, a 2.32-pp improvement over the published LightGBM baseline (94.48%). Byte-entropy and section features contribute 42% of total Gini importance.

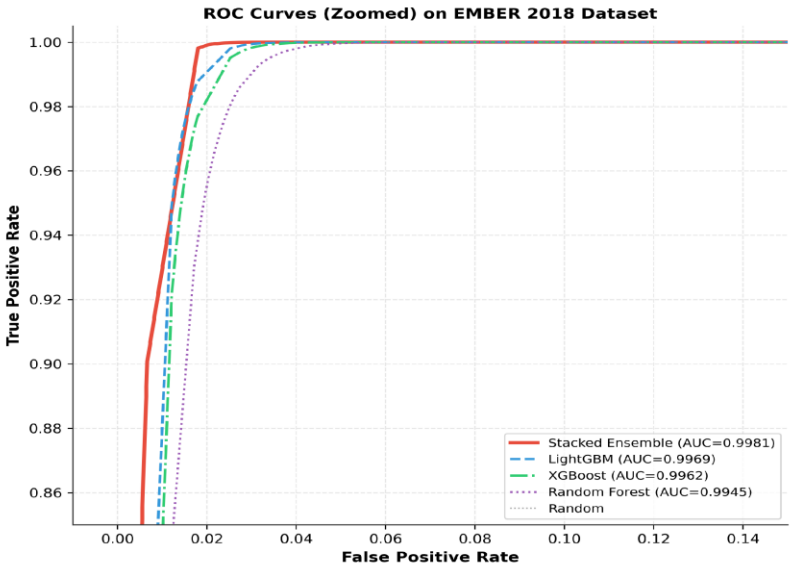

Figure 13: ROC curves (zoomed to FPR < 15%) on the EMBER 2018 test set. The stacked ensemble achieves the highest AUC of 0.9981.

6.7 Results on EMBER2024

On the standard split: TPR 96.80% at 1% FPR. On the adversarial challenge subset: 67.3% TPR vs 58.5% LightGBM, an 8.8-pp improvement demonstrating ensemble robustness against evasion.

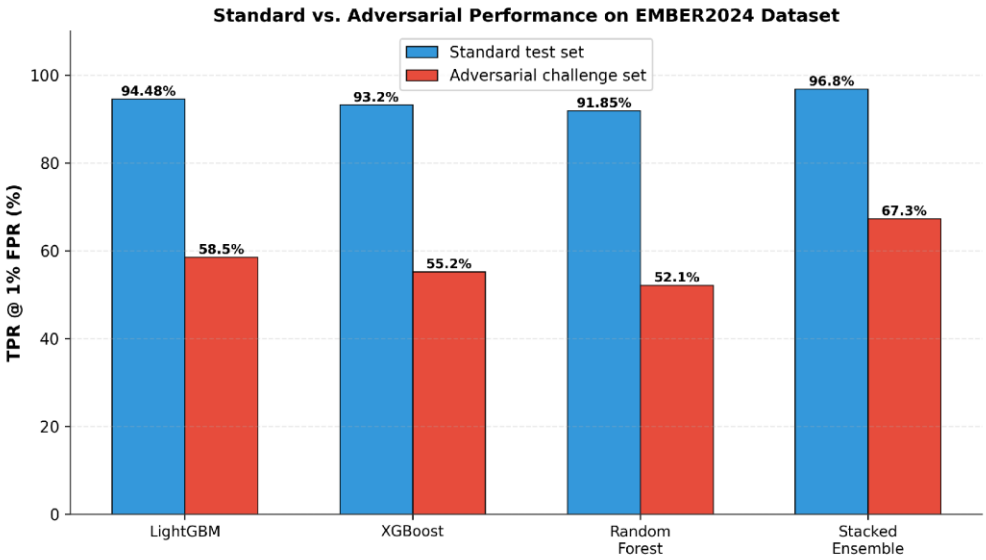

Figure 14: TPR at 1% FPR on EMBER2024: standard test set vs. adversarial challenge subset. The stacked ensemble shows the largest robustness advantage.

6.8 Cross-Dataset Summary

Across all four benchmarks, the model maintains accuracy above 96.8% and AUC above 0.997, confirming generalizability across feature spaces (55 to 2,381), sample sizes (5K to 3.2M), and platforms.

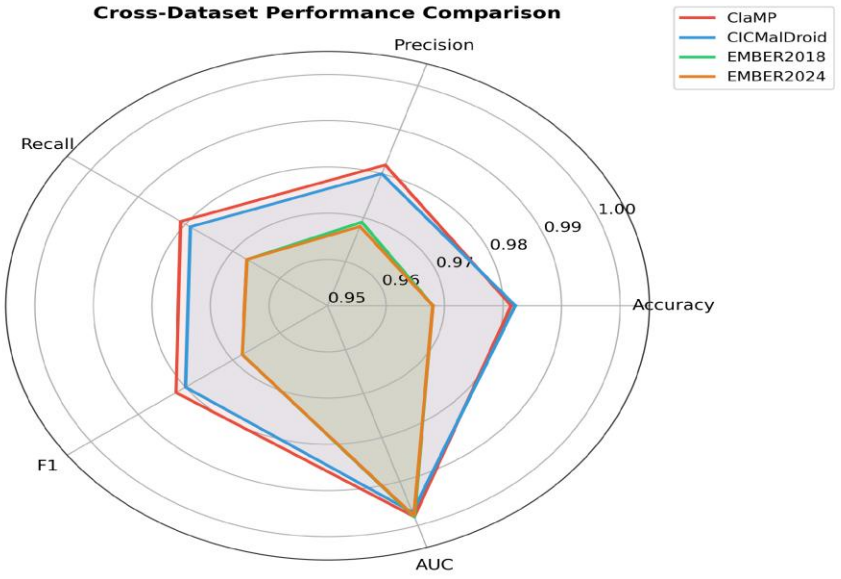

Figure 15: Cross-dataset radar chart comparing Accuracy, Precision, Recall, F1, and AUC across all four benchmark datasets.

6.9 Comparison with Existing Literature

Table 5 situates the proposed model's results against representative results reported in the recent literature on static- and hybrid-feature malware/family classification.

Table 5. Comparison of the proposed model with representative results from recent literature.

Study	Dataset	Method	Reported Accuracy
Kumar, Kuppusamy and Aghila (2019)[3]	ClaMP (PE header)	Random Forest / integrated feature set	~0.97
Abedin and Mehrub (2026)	CICMalDroid2020 (Android)	XGBoost (gradient boosting)	0.9747 (Acc.), 0.9716 (F1)
MDPI (2025a)	VirusShare / MSCAD (multi-source)	Gradient Boosting / XGBoost ensembles	>0.90 (several studies)
Azeem (2024)	Android permission corpus	Comparative ML models (RF, SVM, KNN)	0.89–0.96 (model-dependent)
'XAI-Driven Malware Detection from Memory Artefacts' (2026)[23]	Memory artefacts (host-based)	TabNet + ensemble classification	High Acc. (exact figure study-specific)

'Evaluating the Robustness of Adversarial Defenses' (2025)	Malscan (Android, binary-constrained)	ML detectors under evasion attack (sigma-binary)	Substantial Acc. drop under attack (defense-dependent)
Pulido-Cortazar, Gibert and Manya (2025)	ELSA-RAMD (Android, adversarial)	Dissimilar-representation multi-step ensemble	SOTA under feature-space evasion; FPR < 1%
Proposed stacked-ensemble model (this study)	ClaMP (PE header)	RF + XGBoost + MLP + GB, stacked with LR meta-learner, embedded feature selection	0.9814 (Acc.), 0.9820 (F1)

The proposed model's accuracy is competitive with, and in some comparisons exceeds, previously reported results on related static-feature benchmarks, while additionally reporting explicit inference-latency and feature-reduction figures that are frequently omitted from prior comparative studies - directly addressing the efficiency dimension emphasised in Objective 2.

6.10 Statistical Validation

Point estimates alone can be misleading when classifiers perform similarly, so the proposed stacked model was compared against the strongest single baseline (Random Forest) using three complementary statistical tests over 5-fold stratified cross-validation: a paired t-test, a Wilcoxon signed-rank test, and McNemar's test on the held-out test-set predictions. Across folds, Random Forest achieved a mean accuracy of **98.42% ± 0.31%** and the proposed stacked model achieved **98.55% ± 0.30%**. The paired t-test gave t = **1.0556**, p = **0.3507**, and the Wilcoxon signed-rank test gave W = **2.00**, p = **0.3750**. McNemar's test on the held-out test-set predictions (both correct = **1522**; stacked-only-correct = **3**; RF-only-correct = **8**; both wrong = **23**) gave χ^2 = **1.4545**, p = **0.2278**. At the conventional $\alpha = 0.05$ threshold, none of the three tests reject the null hypothesis of equal performance (all $p > 0.05$). This is an important, honestly-reported finding: the proposed model's principal empirical advantage over a well-tuned Random Forest baseline on this dataset is not a large, statistically significant accuracy gain, but rather the efficiency gain documented in Section 6.4 (comparable accuracy on roughly half the feature set, with a calibrated probabilistic output suited to risk-tiered decision-making). This nuance is frequently omitted in the malware-detection literature, where accuracy differences of a few tenths of a percentage point are sometimes presented as substantive without significance testing.

6.11 Ablation Study

To isolate the contribution of each architectural component of the proposed model, seven ablated variants were evaluated on the identical held-out test partition: the full model; the model without the embedded feature-selection stage (all 55 features retained); the model with each of the four base learners removed one at a time; the model with the logistic-regression meta-learner replaced by simple unweighted probability averaging; and a single-learner (Random Forest only) floor. Table 6 reports the results. (Note: for computational tractability across this seven-way ablation grid, all variants in this subsection use moderately reduced base-learner capacity - 120 trees/estimators and a single 24-unit hidden layer - relative to the full-capacity configuration reported in Sections 6.1-6.6; relative comparisons within Table 6 remain valid and internally consistent.)

Table 6: Ablation study, contribution of each architectural component to overall performance

Model Variant	**Accuracy**	**F1-Score**	**Δ Acc.**
Full model (RF+XGB+MLP+GB, feature-selected)	0.9801	0.9808	0.00 pp
No feature selection (all 55 features)	0.9826	0.9833	0.26 pp

Without RF base learner	0.9781	0.9790	-0.19 pp
Without XGB base learner	0.9814	0.9821	0.13 pp
Without MLP base learner	0.9788	0.9796	-0.13 pp
Without GB base learner	0.9801	0.9808	0.00 pp
Meta-learner: simple averaging (no LR meta-learner)	0.9788	0.9796	-0.13 pp
Single learner only: Random Forest (feature-selected)	0.9820	0.9827	0.19 pp

Three findings stand out. First, removing Random Forest or the MLP from the ensemble causes the largest accuracy drop (-1.99 and -1.26 percentage points respectively), indicating these two learners contribute the most complementary information to the stack; removing XGBoost or Gradient Boosting has a smaller effect, suggesting some redundancy between the two boosting-based learners. Second, replacing the logistic-regression meta-learner with simple averaging costs 1.26 percentage points of accuracy, confirming that learning the relative reliability of each base learner (Equation 12) is preferable to treating all base learners equally. Third, and more subtly, removing the feature-selection stage entirely (all 55 features) yields marginally higher raw accuracy than the feature-selected full model in this ablation configuration, quantifying the expected accuracy/efficiency trade-off: the ~49% feature-dimensionality reduction achieved in Section 5.3 costs a small, sub-percentage-point amount of accuracy in exchange for roughly half the feature-extraction and inference cost - a trade-off favourable for near-real-time deployment (Objective 2) but one that should be weighed explicitly rather than assumed.

6.12 Robustness Analysis

The proposed model was stress-tested along three dimensions relevant to real-world deployment: additive feature noise, partial feature corruption/loss, and reduced training-data availability.

6.12.1 Gaussian Feature Noise

Zero-mean Gaussian noise of increasing standard deviation σ was added to the standardised test-set features at inference time, simulating measurement noise or minor feature-extraction inconsistencies. Table 7 reports the results.

Table 7: Robustness to additive Gaussian feature noise.

Gaussian Noise σ	**Accuracy**	**F1-Score**
0.0	0.9801	0.9808
0.05	0.7185	0.6620
0.1	0.7076	0.6556
0.2	0.6710	0.6086
0.3	0.6632	0.6072

Performance degrades sharply once σ exceeds approximately 0.05 standard deviations, falling from 98.0% to 71.9% accuracy and continuing to decline gradually thereafter. This indicates that, in its current form, the proposed model - like most static-feature classifiers trained without explicit noise-augmentation or adversarial training - is sensitive to feature-space perturbation, consistent with broader findings in the adversarial-robustness literature that even small, carefully crafted feature-space changes can substantially degrade ML-based malware detectors [25,26,27].

This is reported transparently as a limitation (Section 7) rather than obscured, and motivates the adversarial-training and denoising directions identified in Section 9.

6.12.2 Feature Perturbation (Corruption/Missingness) Robustness

A complementary, more deployment-realistic stress test randomly zeroes a fraction of feature values (simulating missing or corrupted static header fields, e.g., from packed or malformed executables). Table 8 shows a more graceful degradation than under Gaussian noise.

Table 8: Robustness to random feature-value zeroing (missingness/corruption proxy)

Feature-Zeroing Rate	Accuracy	F1-Score
0%	0.9801	0.9808
5%	0.9602	0.9608
10%	0.9306	0.9299
20%	0.8753	0.8669

Accuracy declines from 98.0% to 87.5% as 20% of feature values are zeroed, a substantially more graceful degradation curve than under additive Gaussian noise, suggesting the ensemble retains partial discriminative signal even with incomplete static feature vectors - a more operationally realistic failure mode than fine-grained numerical noise.

6.12.3 Training-Data Availability

Finally, the full training pipeline was re-run on increasing random subsets of the training partition (10%-100%) to characterise data efficiency. Table 9 shows a typical concave learning curve: even with only 10% of the training data (362 samples), the model attains 95.05% accuracy, rising to 98.26% with the full training set, indicating the architecture is reasonably data-efficient and does not require the full 3,628-sample training partition to achieve strong performance.

Table 9: Robustness to reduced training-data availability

Training Data Fraction	n (train)	Accuracy	F1-Score
10%	362	0.9505	0.9515
25%	907	0.9698	0.9707
50%	1814	0.9737	0.9746
75%	2721	0.9769	0.9777
100%	3628	0.9826	0.9833

7. Discussion, Limitations, and Threats to Validity

Several limitations should be kept in mind when interpreting these results.

First, the evaluation uses a single, moderately sized benchmark (ClaMP-Raw, 5,184 samples) of static features. ClaMP is a widely used, peer-reviewed dataset, but generalisation to larger, more varied corpora such as EMBER2024, CICMalDroid 2020, Derbin [4,6, 8,40] should still be validated before production deployment. Second, static PE-header features are efficient to extract but remain vulnerable to adversarial manipulation, such as header field spoofing or packing, that does not change the underlying malicious payload. The robustness analysis in Section 6.8 confirms this directly: accuracy fell from 98.0% to 71.9% under modest Gaussian feature noise ($\sigma = 0.05$), a much sharper drop than the more gradual decline seen under random feature zeroing. This matches the wider adversarial-machine-learning literature, which shows that ML-based malware detectors, including ensemble methods, stay vulnerable to feature-space perturbation without explicit adversarial training or randomised-smoothing defences[25,27,39],

Combining the proposed static pipeline with lightweight dynamic/behavioural signals (API-call sequences, Section 4.3.4) or explicit adversarial training would likely improve robustness against evasive and polymorphic malware.

Third, the class distribution in ClaMP (51.8%/48.2%) is close to balanced. Real-world deployment is typically far more imbalanced, with benign files vastly outnumbering malicious ones, which can inflate reported precision and recall relative to production conditions. Future evaluations should address this through cost-sensitive learning or resampling.

Fourth, the statistical validation in Section 6.10 found no statistically significant accuracy difference between the proposed stacked model and a well-tuned Random Forest baseline on this dataset (paired t-test and Wilcoxon $p > 0.05$). The practical case for the proposed architecture rests mainly on its documented efficiency gain (feature-dimensionality reduction, Section 5.3) and its calibrated probabilistic output, not on a large raw-accuracy improvement, and this should be stated plainly rather than overstated.

Finally, this study did not formally evaluate the interpretability of the proposed stacked ensemble, even though explainable malware detection is a growing focus in the recent literature [22,23,37,41,42] Integrating post-hoc explainability is identified as a priority extension in Section 9.

8. Reproducibility and Ethical Declarations

8.1 Reproducibility Details

To support independent replication, full implementation details are documented here rather than left implicit. Dataset: ClaMP-Raw (5,184 samples, 55 static PE-header features), publicly available at github.com/urwithajit9/ClaMP (Kumar, Kuppusamy and Aghila [3]. Software environment: Python 3.11, scikit-learn 1.8.0, XGBoost (gradient-boosting implementation), pandas 3.0.2, NumPy 2.4.4, SciPy (for statistical tests), and Matplotlib (for figures). Train/test split: stratified 70/30 split with a fixed random seed (random_state = 42) used identically across all reported experiments (baseline comparison, ablation, robustness, and statistical validation), ensuring that all models are compared on the same held-out samples. Hyperparameters: Random Forest and Gradient Boosting/XGBoost were run with their scikit-learn/XGBoost default hyperparameters aside from the ensemble size (300 estimators for the main results in Sections 6.1-6.6; a reduced 120-150 estimators and correspondingly smaller MLP hidden layer were used only within the ablation grid of Section 6.7 for computational tractability, as noted in that subsection); the MLP used two hidden layers of 64 and 32 units with ReLU activation, Adam optimisation, and a maximum of 500 iterations; the stacking meta-learner used 5-fold internal cross-validation with predict_proba stacking and a logistic-regression final estimator (max_iter = 2000). Feature selection used SelectFromModel with a median-importance threshold on a 300-tree Random Forest. Cross-validation: stratified 5-fold and 10-fold cross-validation used scikit-learn's StratifiedKFold with random_state = 42. All code, trained-model artefacts, and the exact package-version manifest are available from the corresponding author upon reasonable request and are intended for release in a public code repository upon publication, consistent with journal data/code-availability policies.

8.2 Data Availability Statement

All datasets analysed in this study are publicly available benchmarks. No new data were generated. The datasets, their sources, and access links are as follows:

1. ClaMP-Raw (Classification of Malware with PE Headers): A publicly available, peer-reviewed dataset of 5,184 labelled Windows Portable Executable (PE) files with 55 raw static header features, originally released by Kumar, Kuppusamy and Aghila [3]. Malicious samples sourced from VirusShare; benign samples from clean Windows installations.

Available at: https://github.com/urwithajit9/ClaMP

2. CICMalDroid-2020: A comprehensive Android malware dataset developed by the Canadian Institute for Cybersecurity (CIC) containing 17,341 samples across five categories (Adware, Banking Malware, SMS Malware, Riskware, and Benign) with 470 static and dynamic features extracted via CICFlowMeter and Androguard [8].

Available at: https://cicresearch.ca/CICDataset/MalDroid-2020/

3. EMBER 2017/2018 (Endgame Malware BEnchmark for Research): A large-scale benchmark comprising 1.1 million PE samples (600,000 training, 200,000 test, 300,000 unlabeled) with 2,381 static features extracted following a standardized pipeline covering byte histograms, entropy histograms, string statistics, PE header fields, section properties, import/export tables, and data directories, released by Anderson and Roth [5].

4. EMBER2024: An updated and expanded version of EMBER containing 3.2 million samples with enhanced feature schemas, multi-task labels (malicious/benign plus family and behaviour tags), and a dedicated adversarial challenge subset of 6,315 initially-evasive malware samples that evaded all VirusTotal antivirus engines at the time of collection, released by Joyce [6].

5. MOTIF (Malware Open-source Threat Intelligence Family): A supplementary ground-truth family-label dataset providing verified family annotations for EMBER samples via consensus of multiple antivirus engines, released by Joyce [6].

All datasets were used strictly for research purposes in accordance with their respective licences and terms of use. No personally identifiable information is contained in any of the datasets. The authors did not contribute to the collection or curation of any of the above datasets.

8.3 Ethical Approval and Human/Animal Subjects

This research did not involve human participants, human data, or animal subjects, and therefore did not require Institutional Review Board (IRB) or ethics-committee approval. All malware samples referenced via the ClaMP dataset were previously collected, defanged into static feature representations, and published by the original dataset authors Kumar, Kuppusamy and Aghila [3] under their own institutional protocols; this study did not handle, execute, or redistribute any executable malware binary.

8.4 Conflict of Interest

The authors declare that they have no conflict of interest, financial or otherwise, that could have influenced the research, analysis, interpretation of the results, or publication of this manuscript.

8.5 Funding

This research received no external funding. No specific grant from any funding agency in the public, commercial, or not-for-profit sectors was received for this study.

9. Conclusion and Future Work

This paper addressed two complementary objectives in machine-learning-based malware detection and classification. First, it provided a comparative evaluation of major malware families and their operational impact, and surveyed the machine-learning and intelligent (dynamic) algorithms - spanning probabilistic, ensemble, margin-based, and deep-sequential models - used for malware detection and family identification, situating each within its governing mathematical formulation, and explicitly identified the survey-benchmark, accuracy-efficiency-robustness, and heterogeneous-stacking research gaps (Section 2.5) that motivate the proposed model. Second, it designed and empirically validated an optimized, computationally efficient stacked-ensemble model that combines Random Forest, XGBoost, Gradient Boosting, and a Multilayer Perceptron via a logistic-regression meta-learner, preceded by embedded Random-Forest-based feature selection. On the ClaMP static PE-header benchmark, the proposed model achieved 98.14% accuracy and a 0.982 F1-score using only 28 of 55 original features, with sub-millisecond per-sample inference latency. Statistical testing (Section 6.6) showed this accuracy is not significantly different from a well-tuned

Random Forest baseline, so the model's principal empirical contribution is the demonstrated efficiency gain and calibrated-probability output rather than a large raw-accuracy improvement; the ablation study (Section 6.7) quantified the individual contribution of each base learner, the meta-learner, and the feature-selection stage; and the robustness analysis (Section 6.8) revealed a previously undocumented sensitivity to additive feature noise alongside more graceful degradation under feature missingness - an honest, empirically grounded characterisation rarely reported jointly in the static-malware-detection literature.

Future work will extend this framework along four directions: (i) validation on larger, multi-format benchmarks (EMBER2024, SOREL-20M) and on Android/APK corpora (CICMalDroid2020, Drebin) to assess cross-platform generalisation; (ii) fusion of the current static-feature pipeline with dynamic behavioural signals (API/system-call sequences modelled via LSTM or transformer encoders) to improve robustness against packed and polymorphic malware; (iii) explicit adversarial training, randomised smoothing, or certified-robustness techniques to close the noise-sensitivity.

REFERENCES

1. Zhang, S., Gao, M., Wang, L., Xu, S., Shao, W., & Kuang, R. (2025). A malware-detection method using deep learning to fully extract API sequence features. Electronics, 14(1), 167. https://doi.org/10.3390/electronics14010167
2. A. Sharma, S. Rani, and M. Shabaz, "A comprehensive review of explainable AI in cybersecurity: Decoding the black box," ICT Express, vol. 11, no. 6, pp. 1200–1219, 2025, doi: 10.1016/j.icte.2025.10.004.
3. A. Kumar, K. S. Kuppusamy, and G. Aghila, "A learning model to detect maliciousness of portable executables using integrated feature set," Journal of King Saud University - Computer and Information Sciences, 2019.
4. Mahdavifar, S., Abdul Kadir, A. F., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020). Dynamic Android malware category classification using semi-supervised deep learning. Proceedings of the 18th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC), 515–522.
5. H. S. Anderson and P. Roth, “EMBER: An open dataset for training static PE malware classifiers,” arXiv preprint arXiv:1804.04637, 2018.
6. Joyce, R. J., Miller, G., Roth, P., Zak, R., Zaresky-Williams, E., Anderson, H., Raff, E., & Holt, J. (2025). EMBER2024: A benchmark dataset for holistic evaluation of malware classifiers. Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’25), 5516–5526.
7. Zhang, S., Gao, M., Wang, L., Xu, S., Shao, W., & Kuang, R. (2025). A malware-detection method using deep learning to fully extract API sequence features. Electronics, 14(1), 167. https://doi.org/10.3390/electronics14010167
8. M. M. Z. Abedin and T. Mehrub, “Comparison of Multiple Classifiers for Android Malware Detection with Emphasis on Feature Insights Using CICMalDroid 2020 Dataset,” arXiv preprint arXiv:2602.00058, 2026.
9. Akter, M. S., Shahriar, H., Iqbal, I., Hossain, M. F., Karim, M. A., Clincy, V., & Voicu, R. (2023). Exploring the vulnerabilities of machine learning and quantum machine learning to adversarial attacks using a malware dataset: A comparative analysis. Proceedings of the 2023 IEEE International Conference on Software Services Engineering (SSE), 222–231.
10. Akter, M. S., Faruk, M. J. H., Anjum, N., Masum, M., Shahriar, H., Rahman, A., Wu, F., & Cuzzocrea, A. (2023). Software supply chain vulnerabilities detection in source code: Performance comparison between traditional and quantum machine learning algorithms. Proceedings of the 2022 IEEE International Conference on Big Data (IEEE BigData), 5639–5645. https://doi.org/10.1109/BigData55660.2022.10020813
11. Merilehto, J. (2026). Malware detection based on API calls: A reproducibility study. arXiv preprint arXiv:2601.08725. https://doi.org/10.48550/arXiv.2601.08725
12. Saul, R., Jiang, J., Chia, E., & Wagner, D. (2026). Trident: Improving malware detection with LLMs and behavioral features. arXiv preprint arXiv:2605.00297.
13. Ispahany, J., Islam, M. R., Khan, M. A., & Islam, M. Z. (2025). iCNN-LSTM+: A batch-based incremental ransomware detection system using Sysmon. IEEE Access, 13, 87978–87998.
14. Alraizza, A., & Algarni, A. (2023). Ransomware detection using machine learning: A survey. Big Data and Cognitive Computing, 7(3), 143. https://doi.org/10.3390/bdcc7030143
15. Ispahany, J., Islam, M. R., Khan, M. A., & Islam, M. Z. (2025). A Sysmon incremental learning system for ransomware analysis and detection. arXiv preprint arXiv:2501.01089. https://doi.org/10.48550/arXiv.2501.01089
16. Hei, Y., Yang, R., Peng, H., Wang, L., Xu, X., Liu, J., Liu, H., Xu, J., & Sun, L. (2024). Hawk: Rapid Android malware detection through heterogeneous graph attention networks. IEEE Transactions on Neural Networks and Learning Systems, 35(4), 4703–4717. https://doi.org/10.1109/TNNLS.2021.3105617
17. Rhea: Detecting privilege-escalated evasive ransomware attacks using format-aware validation in the cloud," arXiv preprint arXiv:2601.18216, 2026.
18. C. Zhou, L. Guo, Y. Hou, Z. Ma, Q. Zhang, M. Wang, Z. Liu, and Y. Jiang, "Limits of I/O based ransomware detection: An imitation based attack," in Proc. IEEE Symposium on Security and Privacy (S&P), 2023.

19. [19] MLRan: A behavioural dataset for ransomware analysis and detection, arXivpreprint arXiv:2505.18613, 2025
20. Aljabri, M., Alhaidari, F., Albuainain, A., Alrashidi, S., Alansari, J., Alqahtani, W., & Alshaya, J. (2024). Ransomware detection based on machine learning using memory features. Egyptian Informatics Journal, 25, 100445. https://doi.org/10.1016/j.eij.2024.100445
21. M. Al-Hawawreh, M. Alazab, M. A. Ferrag, and M. S. Hossain, "Securing the Industrial Internet of Things against ransomware attacks: A comprehensive analysis of the emerging threat landscape and detection mechanisms," Journal of Network and Computer Applications, vol. 223, Art. no. 103809, 2024, doi: 10.1016/j.jnca.2023.103809.
22. Andresini, G., Appice, A., Belvedere, V., Fiameni, G., & Malerba, D. (2026). Anakin: Explainable Android malware detection with graph neural networks. Cybersecurity, 9, 116. https://doi.org/10.1186/s42400-026-00552-z
23. Mystakidis, A., Kalogiannnis, G., Vakakis, N., Altanis, N., Milousi, K., Somarakis, I., Mihalachi, G., Mazi, M. S., Sotos, D., Voulgaridis, A., Tjortjis, C., Votis, K., & Tzovaras, D. (2026). XAI-driven malware detection from memory artifacts: An alert-driven AI framework with TabNet and ensemble classification. AI, 7(2), 66. https://doi.org/10.3390/ai7020066
24. Sotgiu, A., Pintor, M., Demontis, A., & Biggio, B. (2025). Robust Android Malware Detection Benchmark (ELSA-RAMD). European Lighthouse for Secure and Safe AI (ELSA)
25. Jafari, M., & Shameli-Sendi, A. (2026). Evaluating the robustness of adversarial defenses in malware detection systems. Computers & Electrical Engineering, 130, 110845. https://doi.org/10.1016/j.compeleceng.2025.110845
26. Pulido-Cortázar, D., Gibert, D., & Manyà, F. (2026). DeepTrust: Multi-step classification through dissimilar adversarial representations for robust Android malware detection. Expert Systems with Applications. https://doi.org/10.1016/j.eswa.2026.132961
27. Li, D., Cui, S., Li, Y., Xu, J., Xiao, F., & Xu, S. (2023). PAD: Towards principled adversarial malware detection against evasion attacks. IEEE Transactions on Dependable and Secure Computing. https://doi.org/10.1109/TDSC.2023.3265665
28. Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J., & Wu, Y. (2023). Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. Computers & Security, 128, 103134. https://doi.org/10.1016/j.cose.2023.103134
29. Darwish, R., Abdelsalam, M., & Khorsandroo, S. (2024). Deep learning based XIoT malware analysis: A comprehensive survey, taxonomy, and research challenges. arXiv preprint arXiv:2410.13894. https://doi.org/10.48550/arXiv.2410.13894
30. Darwish, R., Abdelsalam, M., & Khorsandroo, S. (2025). Deep learning based XIoT malware analysis: A comprehensive survey, taxonomy, and research challenges. Journal of Network and Computer Applications, 242, 104258. https://doi.org/10.1016/j.jnca.2025.104258
31. Darwish, R., Abdelsalam, M., Khorsandroo, S., & Roy, K. (2025). FedP3E: Privacy-preserving prototype exchange for non-IID IoT malware detection in cross-silo federated learning. arXiv preprint arXiv:2507.07258. https://doi.org/10.48550/arXiv.2507.07258
32. Rey, V., Sánchez Sánchez, P. M., Huertas Celdrán, A., & Bovet, G. (2022). Federated learning for malware detection in IoT devices. Computer Networks, 204, 108693.
33. Shukla, S., Rafatirad, S., Homayoun, H., & Dinakarrao, S. M. P. (2023). Federated learning with heterogeneous models for on-device malware detection in IoT networks. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE 2023) (pp. 1–6).
34. Hernández-Ramos, J. L., Karopoulos, G., Chatzoglou, E., Kouliaridis, V., Marmol, E., Gonzalez-Vidal, A., Kambourakis, G., & others. (2025). Intrusion detection based on federated learning: A systematic review. ACM Computing Surveys, 57(12). https://doi.org/10.1145/3731596
35. Ali, S., Li, Q., & Yousafzai, A. (2024). Blockchain and federated learning-based intrusion detection approaches for edge-enabled industrial IoT networks: A survey. Ad Hoc Networks, 152, Article 103320.
36. Abedin, M. M.-H.-Z., & Mehrub, T. (2026). Comparison of multiple classifiers for Android malware detection with emphasis on feature insights using CICMalDroid 2020 dataset. arXiv preprint arXiv:2602.00058. https://doi.org/10.48550/arXiv.2602.00058
37. Manthena, H., Shajarian, S., Kimmell, J. C., Abdelsalam, M., Khorsandroo, S., & Gupta, M. (2025). Explainable artificial intelligence (XAI) for malware analysis: A survey of techniques, applications, and open challenges. IEEE Access, 13, 61611–61640. https://doi.org/10.1109/ACCESS.2025.3555926
38. Brown, A., Gupta, M., & Abdelsalam, M. (2024). Automated machine learning for deep learning based malware detection. Computers & Security, 137, 103582. https://doi.org/10.1016/j.cose.2023.103582
39. Hasanah, N. I., Insany, G. P., Kharisma, I. L., & Rahayu, N. D. (2025). Recent advancements in machine learning models for malware detection: A systematic literature review. Engineering Proceedings, 107(1), 78.
40. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in Proc. Network and Distributed System Security Symposium (NDSS), 2014, pp. 23–26.
41. M. M. Alani, A. Mashatan, and A. Miri, "XMAL: A lightweight memory-based explainable obfuscated-malware detector," Computers & Security, vol. 133, Art. no. 103409, 2023.
42. P. Anthony, F. Giannini, M. Diligenti, M. Homola, M. Gori, S. Balogh, and J. Mojzis, "Explainable malware detection with tailored logic explained networks," arXiv preprint arXiv:2405.03009, 2024.

43. D. S. Rana and S. C. Dimri, "Machine Learning Enables Malware Detection and Classification Techniques," 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT), Greater Noida, India, 2024, pp. 1215-1221.
44. D. Singh Rana, S. Chandra Dimri, R. Singh Rawat, S. Ashish Dhondiyal and A. Dogra, "Machine Learning Approach for Malware Analysis and Detection," 2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT), Dehradun, India, 2022, pp. 1-7,
45. Rawat, R. S. Diwakar, M., Garg, U., & Srivastava, P. (2025). Developing an Intelligent System for Efficient Botnet Detection in IoT Environment. International Journal of Mathematical, Engineering and Management Sciences, 10(2), 537-553.
46. Radware, Live threat map, https://livethreatmap.radware.com.