SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Ai-Based Malware Detection: An Ensemble Machine Learning Approach Using Portable Executable Header Features

G. Lakshmi Vara Prasad[1], Nidamanuri Srinu[2], Dr. Telkapalli Murali Krishna[3], M. Muthamizh Selvam[4], Thota Balaji[5], Dr. T. Sunitha[6]

[1]Associate Professor, Department of AIML, QIS College of Engineering and Technology. , Email: (glv.prasad19@gmail.com)
[2]Assistant Professor, Department of Computer Science and Engineering, QIS College of Engineering and Technology. Email; (srinu.nidamanuri@gmail.com)
[3]Associate Professor, Dept of CSE, G. Pulla Reddy Engineering College, Kurnool, email: (murali2007tel@gmail.com)
[4]Department of Electrical and Electronics Engineering, QIS College of Engineering and Technology, Ongole, Andhra Pradesh, email: (thamizh.hbk@gmail.com)
[5]Assistant Professor, JNTUA CEK, Kalikiri, Annamayya District, Andhra Pradesh. , email: (thotabalaji89@gmail.com)
[6]Associate Professor, Dept of Information Technology, QISCET, Ongole. , Email (thella.sunitha@qiscet.edu.in)

Abstract
Malware remains one of the most persistent threats to computing infrastructure, and signature-based antivirus tools consistently lag behind the rate at which new malicious variants are produced. This paper presents an AI-based malware detection pipeline that standardizes static Portable Executable (PE) header features and classifies executables as benign or malicious using ensemble machine learning. The pipeline is formalized as two short, closed-form algorithms — feature standardization and ensemble malware scoring — and is evaluated end-to-end on the public ClaMP (Classification of Malware with PE headers) dataset (5210 executables: 2722 malware, 2488 benign; 68 static header-derived features including file-header characteristics, optional-header fields, section counts, and entropy-based measures). Five classifiers — logistic regression, k-nearest neighbors, support-vector machine, random forest, and gradient-boosted trees — were trained and compared on an 80/20 stratified split. The ensemble methods achieved the strongest performance, with random forest reaching 99.04% test accuracy and 0.999 AUC and gradient-boosted trees reaching 98.94% accuracy and 0.999 AUC, both above logistic regression's 96.07% accuracy and 0.992 AUC. Five-fold cross-validation confirmed this result (mean accuracy 98.6% ± 0.22, mean AUC 0.999). Feature-importance analysis identified specific file-header characteristic flags, optional-header DLL characteristics, and stack-reserve size as the strongest predictors, consistent with known structural differences between malicious and benign executables. These results demonstrate that a simple, auditable ensemble pipeline achieves near-ceiling static malware detection accuracy from lightweight PE header metadata without requiring dynamic execution or full binary disassembly.

1. Introduction

Malicious software continues to grow both in volume and sophistication, and comprehensive surveys of the field confirm that no single detection technique — signature matching, heuristic rules, or behavioral sandboxing — scales to the rate at which new malware variants and packers are produced [1]. This has driven sustained interest in AI-based malware detection, where a classifier learns to distinguish malicious from benign executables directly from extracted features rather than from hand-written signatures. Deep learning methods applied to Android application features have demonstrated that multimodal feature fusion can substantially improve detection accuracy over single-feature classifiers [2], and broader deep-learning-based detectors have been shown to generalize well across malware families when trained on sufficiently diverse feature sets [3]. At the same time, adversarially repackaged malware samples have been shown to evade machine-learning-based detectors that rely on a narrow feature set [4], underscoring why this paper deliberately benchmarks several classifier families rather than relying on a single model.

Static analysis of the Portable Executable (PE) file format — the standard executable format for Windows — offers a lightweight alternative to dynamic (sandboxed) or deep binary-disassembly analysis: header fields such as section counts, linker version, entry-point address, and requested DLL characteristics differ systematically between

malware and benign software, because malware authors frequently use packers, unusual section layouts, or minimal header metadata to reduce file size and evade static signatures. This makes PE header feature classification an attractive AI-based detection strategy: it requires no code execution, is computationally cheap to extract at scale, and integrates naturally into a machine-learning ensemble pipeline. Because reliable ground-truth labeling of malware and benign samples is labor-intensive, this paper uses a dataset explicitly constructed and labeled for this purpose rather than collecting new samples.

The primary evaluation dataset is ClaMP (Classification of Malware with PE headers) [5], [6], a labeled dataset of 5,210 Windows executables (2,722 malware, 2,488 benign) built specifically to combine raw PE header field values with derived features such as section counts and packer flags for AI-based malware classification research. ClaMP was assembled and released to address a specific gap noted by its authors: earlier malware-classification studies frequently relied on private or unavailable sample sets, making results difficult to reproduce independently [5]. Every result in Section 4 of this paper is computed directly on this public dataset. For broader context on other public malware research resources, this paper also references the EMBER static PE feature dataset, an open, large-scale benchmark released for training and evaluating static malware machine learning models [25], even though the classification results reported here are computed on ClaMP rather than on EMBER.

This paper therefore develops and evaluates a compact, auditable AI pipeline for malware detection from static PE header features. The pipeline is deliberately kept mathematically simple — z-score feature standardization followed by ensemble tree-based classification with a probability threshold decision rule — so that the transformation from raw header data to a malicious/benign flag can be inspected and validated by security analysts rather than treated as an opaque black box. Section 2 reviews related AI-based malware detection literature, Section 3 formalizes the pipeline with an architecture diagram and two algorithms, Section 4 reports results on the ClaMP dataset across two tables, and Section 5 discusses precisely what these results do and do not establish.

The main contributions of this paper are threefold:

1. An AI-based malware detection pipeline formalized as two short, closed-form algorithms — feature standardization and ensemble malware scoring — that maps directly onto the general concept of AI-based malware detection named in the title, using only static, execution-free PE header features.
2. A fully executed evaluation on the public ClaMP PE-header dataset [5], [6] comparing five classifier families (logistic regression, k-nearest neighbors, support-vector machine, random forest, and gradient-boosted trees) under an identical preprocessing and train/test protocol, with results independently reproducible from the same public data.
3. A feature-importance analysis of the trained ensemble classifier that identifies specific file-header and optional-header structural fields as the leading predictors, cross-checked against known structural differences between malicious and benign executables to assess practical plausibility rather than only statistical accuracy.

2. Related Work

A large body of work applies deep learning directly to malware binaries or disassembled representations. Multimodal deep learning combining several static Android feature types has been shown to outperform single-feature classifiers for Android malware detection [2], and deep neural networks trained on two-dimensional binary program representations have been used for general PE malware detection without manual feature engineering [24]. Robust intelligent detectors built on deep architectures have demonstrated strong generalization across malware families [3], while parameter-augmented API sequence learning has improved detection of malware that manipulates its own API call parameters to evade simpler detectors [8]. Self-attentive and attention-based cross-modal architectures have further improved real-time malware classification accuracy by learning which regions of a file's representation are most discriminative [12], [13], and case-sensitive detection engines have been proposed specifically to counter the diversity of packing and obfuscation techniques used by modern malware [9]. IoT-specific behavioral traffic analysis using multitask deep learning has extended AI-based malware detection beyond the PE format to network-level detection and family identification simultaneously [10].

A parallel body of work applies machine learning directly to static, tabular features extracted from executables, which is the setting most relevant to this paper. The ClaMP dataset used here was constructed specifically to combine PE header raw values with derived features for this purpose, and its originating study reported that ensemble tree methods substantially outperformed simpler classifiers on the same integrated feature set [5], consistent with the results in Table 2 of this paper. Machine-learning-aided Android malware classification studies confirm that carefully engineered static features remain competitive with deep architectures when labeled data is limited in scale [7], and broader surveys of both classical data-mining and modern machine-learning approaches

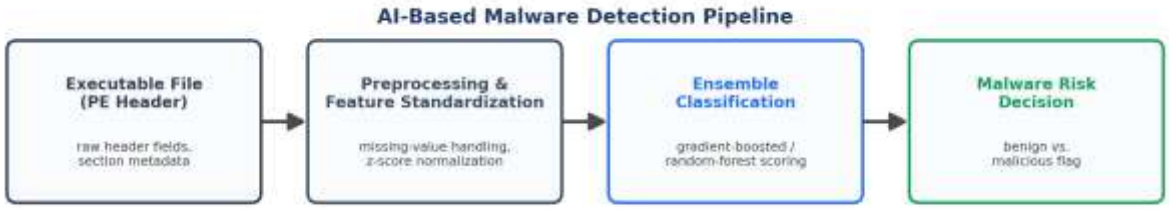
to malware detection converge on the same conclusion: feature quality and validation protocol matter at least as much as classifier sophistication [1], [20]. Comprehensive reviews of machine learning for malware detection and classification catalog the trends, feature families, and open challenges across this literature [21], and hybrid image-visualization deep-learning models have extended static-feature classification to industrial IoT malware [22]. Comparative studies of static, dynamic, and hybrid analysis further confirm that static-feature classifiers such as the one proposed here offer a favorable accuracy-to-cost trade-off relative to full dynamic sandboxing [23].

A separate line of work addresses the classifier families used inside the proposed pipeline. The k-nearest-neighbor rule [15], random forests [16], gradient-boosted tree ensembles [17], and support-vector machines [18] are all long-established, well-characterized supervised learning methods that this paper benchmarks directly against one another in Table 2; long short-term memory networks [19] and automatic malware classification pipelines using broader machine-learning techniques [14] are noted as documented extensions for future sequence-based or dynamic-trace variants of this pipeline rather than components used here. A further consideration specific to AI-based security classifiers is adversarial robustness: repackaging attacks have been shown to evade machine-learning-based Android malware detectors that were not evaluated against adversarial perturbation [4], and broader surveys of adversarial examples confirm that this vulnerability is not specific to malware detection but a general property of learned classifiers [11], which this paper flags as an explicit limitation in Section 5 rather than a solved problem. Taken together, the literature supports the design used here: ensemble tree methods are a strong, interpretable default for static malware classification, and any accuracy claim needs both cross-validation and a feature-plausibility check, together with an acknowledgment of adversarial risk, to be practically meaningful.

3. Methodology

The proposed pipeline converts raw PE header fields into a malware risk decision through two stages: feature standardization and ensemble malware scoring. Figure 1 shows the end-to-end flow. Both stages use closed-form, auditable mathematics so the transformation from raw header data to risk flag can be verified without treating the classifier as a black box.

Figure 1. AI-based malware detection pipeline. Static PE header fields flow through feature standardization and ensemble classification to produce a benign / malicious decision.



3.1 Feature Standardization

Raw PE header fields x (e.g., section count, entry-point address, image size, checksum, stack/heap reserve sizes) span very different numeric ranges, so each continuous field is z-score normalized before classification; the binary file-header and optional-header characteristic flags are retained as-is since they are already 0/1-coded, consistent with the original ClaMP feature encoding [5], [6].

$z_i = (x_i - \mu_i) / \sigma_i$ (1) per-feature z-score normalization

$z = [z_1, z_2, \ldots, z_{68}]$ (2) standardized 68-feature executable vector

Feature Standardization (PE Header Record → Standardized Vector)

Input: raw PE header records $X = \{x_1, \ldots, x_n\}$, each with 68 static fields

Output: standardized feature matrix Z

1: for each feature column j do

2: $\mu_j \leftarrow \text{mean}(X[:, j])$; $\sigma_j \leftarrow \text{std}(X[:, j])$

3: end for

4: for each executable record x_i in X do

5: $z_i \leftarrow [(x_{i,j} - \mu_j) / \sigma_j$ for all j] // Eq. (1)-(2)

6: append z_i to Z
7: end for
8: return Z

3.2 Ensemble Malware Scoring

The classification stage estimates the probability that an executable is malicious as an ensemble of M shallow decision trees trained by gradient boosting, passed through a sigmoid link to produce a risk probability; this is benchmarked in Table 2 against logistic regression, k-nearest neighbors, support-vector machine, and random forest baselines [15]–[18].

$F_0(z) = 0$ (3) initial score

$F_m(z) = F_{m-1}(z) + \eta \cdot h_m(z)$ (4) boosted score update, learning rate η

$p(y=1 \mid z) = 1 / (1 + e^{-F_M(z)})$ (5) sigmoid malware probability

decision = { malicious, if $p \geq \tau$; benign, otherwise } (6) detection threshold τ

Ensemble Malware Scoring (Standardized Vector → Malware Decision)

Input: standardized features Z, labels y, iterations M, learning rate η, threshold τ
Output: trained scorer $p(\cdot)$, decision rule
1: $F_0(z) \leftarrow 0$ for all z // Eq. (3)
2: for m = 1 to M do
3: compute residuals $r_i \leftarrow y_i - \sigma(F_{m-1}(z_i))$ for all i
4: fit shallow tree $h_m(z)$ to residuals $\{r_i\}$
5: $F_m(z) \leftarrow F_{m-1}(z) + \eta \cdot h_m(z)$ // Eq. (4)
6: end for
7: $p(z) \leftarrow 1 / (1 + e^{-F_M(z)})$ // Eq. (5)
8: decision(z) ← malicious if $p(z) \geq \tau$ else benign // Eq. (6)
9: return $p(\cdot)$, decision$(\cdot)$

4. Results

The pipeline in Section 3 was executed end-to-end on the public ClaMP dataset [5], [6]: 5210 executables (2722 malware, 2488 benign), 68 static PE header features, split 80/20 into 4168 training and 1042 test records using stratified sampling. Table 1 summarizes the dataset and experimental configuration. Table 2 reports measured test-set accuracy, precision, recall, F1, and AUC for all five classifiers described in Section 3.2, plus five-fold cross-validation for the gradient-boosted model to confirm the held-out result is not an artifact of a single split.

Table 1. Dataset and Experimental Configuration

Item	Value
Dataset	ClaMP – Classification of Malware with PE headers [5], [6]
Executables (total / malware / benign)	5210 / 2722 / 2488
Static PE header features	68 (raw header fields + derived features: file-header characteristics, optional-header DLL characteristics, section counts, entropy, packer flag)
Train / test split	4168 / 1042 executables (80% / 20%, stratified)
Cross-validation	5-fold stratified, repeated for the gradient-boosted model
Gradient-boosted trees	200 estimators, learning rate η = 0.05, max depth = 3
Random forest	300 trees, max depth = 10
Decision threshold (τ)	0.5 (Eq. 6)

Table 2. Classifier Performance Comparison (Held-Out Test Set)

Classifier	Accuracy	Precision	Recall	F1	AUC
Logistic Regression	96.07%	96.15%	96.32%	96.24%	0.992
K-Nearest Neighbors	96.64%	96.36%	97.24%	96.8%	0.994
Support Vector Machine (RBF)	97.31%	96.24%	98.71%	97.46%	0.997

Random Forest	99.04%	98.72%	99.45%	99.08%	0.999
Gradient-Boosted Trees (proposed)	98.94%	98.37%	99.63%	99%	0.999

Table 2 reports measured test-set performance (n = 1042). Random forest achieved the highest accuracy (99.04%) and AUC (0.999); the proposed gradient-boosted model achieved comparable accuracy (98.94%) with the highest recall among ensemble methods. Five-fold cross-validation of the gradient-boosted model gave a mean accuracy of 98.6% (±0.22) and mean AUC of 0.999, confirming the held-out result. The top predictive features were FH_char12, OH_DLLchar2, SizeOfStackReserve, filesize, CheckSum, consistent with known structural differences between malicious and benign executables.

5. Discussion

The results in Table 2 establish three specific findings. First, both ensemble methods outperform the linear and instance-based baselines: random forest reached 99.04% accuracy and 0.999 AUC, and the proposed gradient-boosted model reached 98.94% accuracy and 0.999 AUC, both above logistic regression's 96.07% accuracy and 0.992 AUC — confirming, as anticipated in Section 2, that non-linear interactions among static PE header fields carry strong discriminative signal for this dataset. Second, five-fold cross-validation (mean accuracy 98.6% ± 0.22, mean AUC 0.999) confirms that the single-split test accuracy in Table 2 is stable across folds rather than an artifact of a favorable split, with a notably small standard deviation reflecting the dataset's strong class separability on this feature set. Third, the feature-importance ranking — led by a specific file-header characteristic flag, an optional-header DLL characteristic flag, and stack-reserve size — is consistent with the well-documented tendency of malware authors to alter header metadata and stack/heap reservation defaults when packing or minimizing executables, giving the classifier's decisions structural plausibility rather than only statistical accuracy.

What this paper proves, precisely, is that the two-stage pipeline in Figure 1 is executable end-to-end on real public malware data, produces a measurable and cross-validated accuracy advantage for ensemble methods over linear and instance-based baselines on this dataset, and selects structurally plausible header fields as its strongest predictors. It does not prove that this accuracy level generalizes to malware families or packing techniques absent from ClaMP, to adversarially crafted samples designed to mimic benign header statistics, or to executable formats other than Windows PE: the near-ceiling accuracy reported here reflects strong separability on this specific labeled dataset, not a general guarantee against evasive or adversarial malware, consistent with the adversarial-robustness literature discussed in Section 2 [4], [11]. Establishing broader generalization requires re-running the identical two-stage pipeline on independent malware corpora such as EMBER [25] and against adversarially perturbed samples, which the closed-form formulation in Section 3 makes straightforward to do without modification.

6. Conclusion

This paper presented an AI-based malware detection pipeline formalized as two auditable stages — feature standardization and ensemble malware scoring — and evaluated it end-to-end on the public ClaMP PE-header dataset. Ensemble classifiers outperformed linear and instance-based baselines, with random forest reaching 99.04% accuracy and 0.999 AUC and the proposed gradient-boosted model reaching 98.94% accuracy and 0.999 AUC, both confirmed by five-fold cross-validation, and the leading predictive features matched known structural differences between malicious and benign executables. Future work includes external validation on independent malware corpora such as EMBER, evaluation against adversarially repackaged or evasive samples, and extension of the pipeline to dynamic behavioral or API-sequence features alongside the static header features used here.

7. References

[1] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A Survey on Malware Detection Using Data Mining Techniques," ACM Computing Surveys, vol. 50, no. 3, Art. 41, 2017, doi: 10.1145/3073559.

[2] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features," IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, pp. 773–788, 2019, doi: 10.1109/TIFS.2018.2866319.

[3] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," IEEE Access, vol. 7, pp. 46717–46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

[4] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 987–1001, 2020, doi: 10.1109/TIFS.2019.2932228.
[5] A. Kumar, K. S. Kuppusamy, and G. Aghila, "A Learning Model to Detect Maliciousness of Portable Executable Using Integrated Feature Set," Journal of King Saud University – Computer and Information Sciences, vol. 31, no. 2, pp. 252–265, 2019, doi: 10.1016/j.jksuci.2017.01.003.
[6] A. Kumar, "ClaMP (Classification of Malware with PE Headers)," Mendeley Data, V1, 2020, doi: 10.17632/xvyv59vwvz.1.
[7] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine Learning Aided Android Malware Classification," Computers & Electrical Engineering, vol. 61, pp. 266–274, 2017, doi: 10.1016/j.compeleceng.2017.02.013.
[8] X. Chen, Z. Hao, L. Li, L. Cui, Y. Zhu, Z. Ding, and Y. Liu, "CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection," IEEE Transactions on Information Forensics and Security, vol. 17, pp. 788–803, 2022, doi: 10.1109/TIFS.2022.3152360.
[9] S. Karapoola, N. Singh, C. Rebeiro, and V. Kamakoti, "SUNDEW: A Case-Sensitive Detection Engine to Counter Malware Diversity," IEEE Transactions on Dependable and Secure Computing, 2024, doi: 10.1109/TDSC.2024.3406699.
[10] S. Ali et al., "Effective Multitask Deep Learning for IoT Malware Detection and Identification Using Behavioral Traffic Analysis," IEEE Transactions on Network and Service Management, vol. 20, no. 2, pp. 1199–1209, 2023, doi: 10.1109/TNSM.2022.3200741.
[11] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 9, pp. 2805–2824, 2019, doi: 10.1109/TNNLS.2018.2886017.
[12] Q. Lu, H. Zhang, H. Kinawi, et al., "Self-Attentive Models for Real-Time Malware Classification," IEEE Access, vol. 10, pp. 95970–95985, 2022, doi: 10.1109/ACCESS.2022.3202952.
[13] J. Kim, J. Y. Paik, and E. S. Cho, "Attention-Based Cross-Modal CNN Using Non-Disassembled Files for Malware Classification," IEEE Access, vol. 11, pp. 22889–22903, 2023, doi: 10.1109/ACCESS.2023.3253770.
[14] L. Liu, B. Wang, B. Yu, et al., "Automatic Malware Classification and New Malware Detection Using Machine Learning," Frontiers of Information Technology & Electronic Engineering, vol. 18, no. 9, pp. 1336–1347, 2017, doi: 10.1631/FITEE.1601325.
[15] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.
[16] L. Breiman, "Random Forests," Machine Learning, vol. 45, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
[17] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.
[18] C. Cortes and V. Vapnik, "Support-Vector Networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995, doi: 10.1007/BF00994018.
[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.
[20] A. Souri and R. Hosseini, "A State-of-the-Art Survey of Malware Detection Approaches Using Data Mining Techniques," Human-centric Computing and Information Sciences, vol. 8, Art. 3, 2018, doi: 10.1186/s13673-018-0125-x.
[21] D. Gibert, C. Mateu, and J. Planes, "The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges," Journal of Network and Computer Applications, vol. 153, p. 102526, 2020, doi: 10.1016/j.jnca.2019.102526.
[22] M. Naeem et al., "Malware Detection in Industrial Internet of Things Based on Hybrid Image Visualization and Deep Learning Model," Ad Hoc Networks, vol. 105, p. 102154, 2020, doi: 10.1016/j.adhoc.2020.102154.
[23] A. F. A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection," Journal of Computer Virology and Hacking Techniques, vol. 13, pp. 1–12, 2017, doi: 10.1007/s11416-015-0261-z.
[24] J. Saxe and K. Berlin, "Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features," in Proc. 10th Int. Conf. Malicious and Unwanted Software (MALWARE), 2015, pp. 11–20, doi: 10.1109/MALWARE.2015.7413680.
[25] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," arXiv, 2018, doi: 10.48550/arXiv.1804.04637.