engineering proceedings

MDPI

Proceeding Paper

Lightweight Static Malware Detection Using Optimized Section-Level Features from Portable Executables †

Mehdi Kmiti *, Jallal-Eddine Moussaoui, Khalid El Gholami and Yassine Maleh

LaSTI Laboratory, Sultan Moulay Slimane University, Beni Mellal 23000, Morocco; jallal-eddine.moussaoui@usms.ac.ma (J.-E.M.); k.elgholami@usms.ma (K.E.G.); y.maleh@usms.ma (Y.M.)

* Correspondence: mehdi.kmiti@usms.ac.ma

† Presented at the 8th edition of the International Conference on Advanced Technologies for Humanity (ICATH 2026), Rabat, Morocco, 15–17 July 2026.

Abstract

Malware evolution challenges traditional detection, demanding lightweight, interpretable, and scalable solutions. Static analysis offers fast portable executable (PE) file inspection but often relies on large feature sets (API calls, DLL imports, opcodes) that increase overhead. Dynamic analysis is slow and evasion prone. We propose a static framework using only section-level PE attributes. With recursive feature elimination (RFECV), GridSearch, and extra trees, we reduce 61 features to 13 discriminative ones. On 1258 real-world Windows PE files, five-fold cross-validation achieves 94.9% accuracy, and the held-out test set achieves 95.5% accuracy with an AUC of 0.987. Our compact feature set reduces dimensionality by 79% while maintaining competitive performance, enabling real-time deployment on endpoints, IoT, and cloud AV systems.

Keywords: static malware detection; PE section attributes; recursive feature elimination (RFECV); extra trees; feature selection

1. Introduction

Malware remains one of the most pervasive and rapidly evolving threats to global cybersecurity. In 2023 alone, over 100 million new malware variants were identified, contributing to a total corpus exceeding 1.2 billion distinct samples by 2024 [1]. Recent reports estimate that between 400,000 and 560,000 new malicious executables emerge every day, illustrating both the unprecedented scale and accelerating pace of malware proliferation [2]. This surge has had tangible consequences: malware was implicated in nearly 40% of data breaches in 2023, and more than 80% of organizations reported facing malware-driven incidents [2]. Such statistics reflect both the sophistication of modern threats and the urgent need for detection mechanisms that can scale reliably in real-world environments.

Traditional signature-based antivirus systems, while effective against previously encountered samples, struggle to keep up with polymorphic, obfuscated, and zero-day malware [3,4]. Machine learning (ML) has therefore become a critical component of modern defense strategies, offering the ability to generalize from large datasets and detect previously unseen threats [5]. Existing ML-based malware detection systems, however, commonly rely on large, heterogeneous, and high-dimensional feature sets including DLL imports, API call sequences, opcodes, behavioral traces, and PE metadata [6]. These features increase computational cost, hinder interpretability, and complicate deployment on resource-constrained devices such as IoT endpoints, mobile agents, and cloud-integrated sensors.

Check for updates

Academic Editors: Brahim El Bhiri, Ayman S. Mosallam and Younes Karfa Bekali

Published: 20 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Malware detection has been extensively explored using both static and dynamic analysis techniques, with ML and deep learning (DL) models playing a central role in improving detection capabilities. However, existing approaches vary greatly in terms of computational efficiency, scalability, robustness against evasion, and interpretability [7,8].

Dynamic analysis monitors runtime behavior within a sandbox, providing rich semantic insights. Akhtar and Feng [9] introduced a dynamic, behavior-based detection framework that relied on ML models to analyze malware execution within sandboxed environments. By extracting hundreds of behavioral features from VirusTotal reports, they demonstrated the potential of runtime monitoring for accurate detection. Similarly, Syeda and Asghar [10] focused on dynamic analysis by categorizing API call frequencies into semantic groups and applying a two-stage feature selection process, ultimately improving detection performance while emphasizing the importance of structured API modeling. Kim et al. [11] also adopted a dynamic approach, proposing a novel preprocessing method that transformed variable-length API sequences into TF-IDF-weighted vectors suitable for DL, where a bidirectional LSTM classifier effectively captured behavioral dependencies. Along similar lines, Shahnawaz et al. [12] explored visual representations of dynamic behaviors by converting API call arguments into grayscale images, leveraging convolutional neural networks (CNNs) for classification and demonstrating resilience to obfuscation techniques. Gond et al. [13] combined natural language processing (NLP) and ML by extracting API call sequences using the Cuckoo sandbox and representing them as TF-IDF-weighted n-grams, which were classified with ensemble methods such as XGBoost and LightGBM.

Despite these advances, dynamic analysis suffers from several critical drawbacks. First, it is inherently slow and resource-intensive, as each sample must be executed for several minutes to observe its full behavior [14]. Second, it requires a controlled sandbox environment, which is expensive to maintain and scale. Third, modern malware increasingly employs anti-analysis techniques such as sandbox detection, environment awareness, delayed execution, and conditional triggers that only activate under specific conditions effectively evading dynamic inspection [15,16]. Maniriho et al. [17] proposed EarlyMalDetect, a proactive hybrid framework leveraging GPT-2 and BiGRU models to anticipate API calls early during execution. By predicting malicious behaviors before payload delivery, their method reduced detection latency. However, this approach depended heavily on large-scale dynamic data and introduced significant computational overhead, making it unsuitable for deployment in constrained environments. Consequently, dynamic analysis is ill-suited for real-time or large-scale deployment.

Static analysis, in contrast, examines the structure of binaries without executing them, offering safety, scalability, and low computational overhead [18,19]. These properties make static approaches highly attractive for real-time detection, provided that the selected static features are sufficiently discriminative. Ayub et al. [20] developed Static-RWArmor, a framework for detecting cryptographic ransomware using PE metadata combined with dimensionality reduction techniques such as PCA. Although effective for ransomware detection, their framework focused on a single malware category rather than providing a generalized solution. Yousuf et al. [21] proposed a more comprehensive static analysis system that integrated imported DLLs, API calls, PE headers, and section-level characteristics into large feature sets. While their approach demonstrated the benefit of feature integration, it introduced high-dimensional datasets that increased computational complexity and limited interpretability. Baker Del Aguila et al. [22] addressed detection in resource-constrained environments by proposing lightweight models optimized via genetic algorithms on PE metadata features. Their framework demonstrated potential for IoT contexts, but it lacked a detailed evaluation of runtime efficiency and deployment feasibility. Miraoui and Belgacem [23] contributed another static-based framework, combining

classical ML algorithms and DL architectures to classify Windows PE files across binary and multiclass tasks. Their findings showed that well-tuned traditional algorithms such as random forest can rival DL models in accuracy, but their reliance on mixed, heterogeneous feature sets increased complexity and hindered real-time applicability.

Despite these advances, a critical gap remains in static analysis. Most static detection methods depend on excessively large or redundant feature representations, often involving hundreds or thousands of dimensions derived from API imports, opcode sequences, or disassembled instructions. Such high-dimensional models are difficult to interpret, slow to train, and impractical for lightweight deployment on edge devices or real-time security tools. Dimensionality reduction methods like PCA and information gain have been used in some works, but they typically operate as post-processing steps and may not ensure the selection of the most discriminative features. Moreover, hybrid and deep learning-based approaches tend to require substantial computational resources and often lack interpretability.

Compact structural features derived from portable executable (PE) sections such as virtual address, virtual size, raw size, entropy, and section presence are promising alternatives due to their stability, ease of extraction, and resistance to certain evasion techniques. However, their full discriminative potential has remained underexplored, especially when optimized through systematic feature selection. No prior work has systematically asked how few section-level attributes are needed to achieve accuracy comparable to much richer feature sets.

To address this gap, we propose a lightweight and interpretable static malware detection framework based exclusively on section-level PE attributes. Using recursive feature elimination with cross-validation (RFECV) guided by an extra-trees classifier and hyperparameter tuning via GridSearch, we systematically identify the most relevant attributes while eliminating redundancy. We reduce an initial set of 61 structural features to a compact subset of 13 highly discriminative features. Experimental evaluation on a balanced real-world dataset of 1258 Windows PE files (629 malware, 629 benign)—where benign samples include both system binaries and diverse user-space applications—shows that the reduced feature set achieves 94.9% accuracy under five-fold cross-validation and 95.5% accuracy on a held-out test set, with an AUC of 0.987. This represents a 79% reduction in dimensionality while maintaining competitive performance. Our framework offers a scalable, efficient, and deployable solution suitable for real-time cybersecurity applications, including endpoint protection, IoT devices, and cloud-based antivirus systems.

2. Materials and Methods

2.1. Dataset Construction

The dataset consists of 1258 Windows PE files, evenly balanced between 629 malware and 629 benign samples.

Malware files were sourced from the VirusShare repository (2012–2024), selected for the following reasons:

- High family diversity: VirusShare includes thousands of malware families (e.g., ransomware, worms, trojans, spyware), ensuring heterogeneous malicious behavior.
- Temporal diversity: Samples span twelve years, reducing temporal bias and increasing generalization.
- Authenticity: All files are real-world malware, not synthetic or simulated samples.

Benign executables were collected from two complementary sources to avoid distribution-specific artefacts and improve real-world generalization:

- System32 (C:\Windows\System32 on a clean Windows 10 installation): This ensures authenticity and includes system binaries that malware often impersonates.

https://doi.org/10.3390/engproc2026168006

- User-space applications: This introduces structural diversity (e.g., packed installers, custom sections) that better reflects real-world benign software.

The dataset is intentionally balanced to prevent classifiers from being biased toward the dominant class, a common issue with malware datasets.

While the size (1258 samples) might appear modest, the dataset is structurally rich, containing 61 PE section-level attributes per file. Numerous peer-reviewed static detection studies operate with comparable dataset sizes, and the dataset supports five-fold cross-validation, ensuring robust generalization.

2.2. Section-Level Feature Extraction

Feature extraction was performed using the pefile Python library, which provides reliable access to PE headers. Section-level attributes were extracted from twelve standard PE sections: .text, .data, .rdata, .rsrc, .idata, .edata, .tls, .reloc, .pdata, .debug, .bss, and .code.

- .text and .code: Contain executable instructions. Malware often injects or obfuscates code in these regions to alter behavior while avoiding detection.
- .data and .rdata: Store initialized and read-only data. Unexpected modifications in these sections may indicate the insertion of malicious variables or constants.
- .rsrc: Holds resources such as icons, images, or configuration files. Malware frequently exploits this section to embed payloads or disguise itself as legitimate software.
- .idata and .edata: Define imports and exports, particularly API call dependencies. Manipulation of these entries is a common technique for concealing malicious activity or redirecting execution flow.
- .tls: Provides thread-local storage. Advanced malware may exploit this section to execute malicious code before the main program initialization, thereby bypassing traditional detection.
- .reloc, .pdata, and .debug: Contain relocation information, exception-handling data, and debugging information, respectively. Abnormalities in these sections often reveal packing, anti-debugging mechanisms, or other evasion techniques.
- .bss: Stores uninitialized data, which may be leveraged by malware to hide buffers or malicious variables.

Section-level attributes were deliberately chosen over API imports, opcodes, or dynamic behavior due to several advantages. First, they offer robustness under packing, as many PE sections remain detectable even after basic packing, unlike opcodes or API imports. Second, they enable lightweight extraction, requiring no disassembly or sandbox, and can be extracted in milliseconds. Third, they provide stability, since section structure is harder to obfuscate without breaking the executable. Finally, their low-dimensional representation enables real-time deployment on IoT and endpoint devices.

For each section, five descriptors were collected:

- Virtual Address: indicates where the section is loaded into memory. Unusual memory mapping may suggest injected or hidden code.
- Virtual Size: specifies the amount of memory allocated to the section. Discrepancies with raw size may reveal obfuscation or concealed payloads.
- Raw Size: represents the on-disk size of the section. Differences between raw and virtual sizes often point to compression or hidden content.
- Entropy: measures the randomness of the section's contents. High entropy values are commonly associated with packing, encryption, or other obfuscation methods.
- Section Presence: a binary feature indicating whether the section exists in the file. Malware may omit standard sections or add non-standard ones to manipulate execution.

https://doi.org/10.3390/engproc2026168006

Numerical descriptors (virtual address, virtual size, raw size, entropy) are set to zero for absent sections, since the presence flag explicitly encodes absence, this does not conflate zero-valued present sections with absent ones.

In addition to section-specific attributes, a global feature representing the total number of sections (total_sections) was included. This attribute captures broader structural anomalies, since malware often alters the overall layout of executables by adding or removing sections.

Thus, the model can independently learn from two separate signals: whether a section exists (via the presence flag) and the numerical properties of sections that do exist.

Each file is represented by 61 features. This compact, yet expressive feature set offers a balance of discriminative power, computational efficiency, and interpretability.

While other studies employ opcode sequences, API call traces, or dynamic behavioral features, such approaches often require extensive preprocessing, runtime execution in sandboxed environments, or high-dimensional representations that increase computational overhead. In contrast, section-level attributes can be extracted rapidly and safely without executing the sample, making them more suitable for scalable, lightweight, and real-time malware detection. This efficiency, combined with interpretability, motivated the selection of section-level structural and statistical features for this framework.

2.3. Data Preprocessing

Before training, the dataset was subjected to several preprocessing operations to improve data quality, ensure consistency, and optimize the learning process, thereby eliminating redundancy, reducing noise, and preparing features for ML algorithms [24–27].

The preprocessing procedure consisted of four stages. First, data cleaning: duplicate records and incomplete entries were removed to avoid overrepresentation of repeated patterns and inconsistencies that could reduce feature reliability. This ensured the dataset was accurate, complete, and free of redundancy.

Second, normalization: all numerical attributes (excluding identifiers and class labels) were normalized to the range [0, 1] using MinMaxScaler. This was important because features like virtual addresses and section sizes had values orders of magnitude larger than others, which could bias the classifier. Scaling all features to a uniform range ensured equal contribution from each attribute, improving accuracy and convergence speed.

Third, shuffling: the dataset was randomly shuffled before splitting to prevent order-related biases (e.g., malware samples grouped together). This allowed the classifier to learn from a representative distribution of both classes.

Fourth, dataset splitting: a 70/30 split was applied (70% training, 30% testing). Combined with cross-validation, this prevents overfitting. The split used train_test_split with a fixed random seed (random_state = 42) to guarantee reproducibility. A balanced split between malware and benign samples in both sets ensured representative training and testing conditions.

Overall, these preprocessing steps produced a clean, normalized, randomly distributed, and reproducible dataset, providing a reliable foundation for feature selection and classification.

The proposed data preprocessing workflow, including data cleaning, normalization, shuffling, and dataset splitting, is illustrated in Figure 1.

https://doi.org/10.3390/engproc2026168006

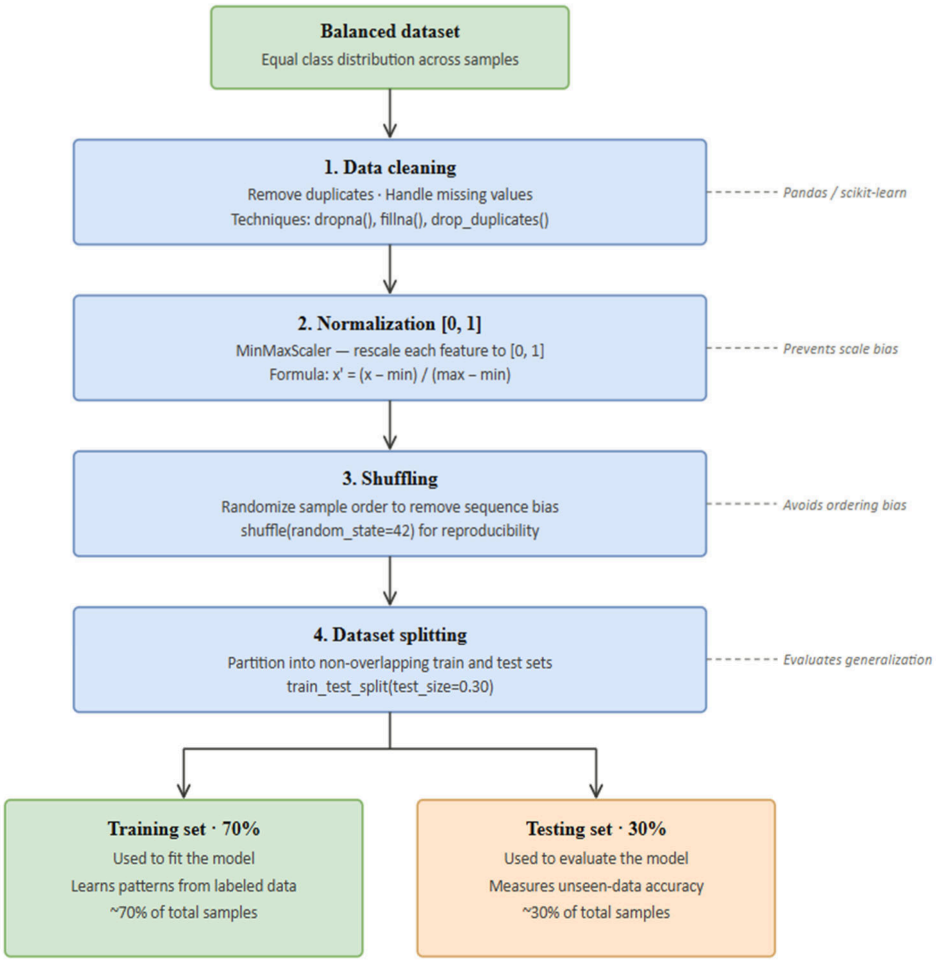


Figure 1. Data preprocessing pipeline applied to section-level PE features.

2.4. Feature Selection

Before feature selection, the extra-trees classifier was tuned using GridSearchCV with five-fold cross-validation on the training set. The hyperparameter grid was:

- n_estimators: [100, 200]
- max_depth: [10, 20, None]
- min_samples_split: [2, 5]
- min_samples_leaf: [1, 2]

The best combination found was max_depth = None, min_samples_leaf = 1, min_samples_split = 5, n_estimators = 200, achieving a cross-validation accuracy of 94.9% on the full feature set.

Using the optimized extra trees as the estimator, RFECV was applied to the training data with step = 2 (removing two features per iteration) and stratified five-fold CV. The search explored feature subsets from 61 down to a minimum of 5 features. The optimal subset size was determined as the one that maximized mean cross-validation accuracy. This procedure selected 13 features. The step = 2 granularity was chosen to balance computational cost and search resolution, and a finer step = 1 was tested on a pilot run and produced identical accuracy, confirming robustness.

This approach was chosen to systematically identify the smallest subset of features capable of achieving high discrimination between benign and malicious executables while simultaneously reducing redundancy and enhancing model interpretability [28–30]. RFECV is particularly well suited for PE structural data because it iteratively removes correlated or

non-informative attributes, ensuring that only the most relevant section-level features are retained. By eliminating redundant attributes, RFECV allows the model to focus on the core structural indicators that contribute most to classification accuracy, thereby avoiding the need for manual feature engineering.

The extra-trees classifier was selected as the estimator for RFECV due to several methodological advantages. First, extra trees provides stable and reliable feature-importance scores, which are essential for recursive elimination. Second, its ensemble-based structure captures complex, nonlinear relationships between PE-section attribute that linear methods such as SVMs cannot model effectively. Third, extra trees introduces greater randomization than random forest, reducing the risk of overfitting and improving generalization on unseen malware samples. Finally, extra trees is computationally lightweight compared to more complex alternatives such as XGBoost, making it well suited for an iterative feature selection procedure.

RFECV automatically searches over feature subset sizes using the optimized extra-trees estimator, evaluating each candidate subset via stratified five-fold cross-validation. The search explored feature subsets in steps of two features from 61 down to a minimum of 5 features. Each candidate subset was evaluated using accuracy under stratified five-fold cross-validation, ensuring robust performance estimation and reproducibility [31,32]. The RFECV procedure automatically selected the subset that maximized cross-validation accuracy.

The procedure reduced the initial set of 61 section-level attributes to a compact subset of 13 highly discriminative features.

2.5. Classification and Evaluation

The optimized feature set was used to train an extra-trees classifier, selected because it aligns with the framework's goal of lightweight static detection, requires minimal hyperparameter tuning, is fast to train and evaluate, naturally supports feature interpretability, and prior studies have shown that extra trees performs strongly on PE structural data.

To assess the effectiveness of the proposed malware detection framework, multiple performance metrics were adopted. These metrics provide insights into both the overall accuracy and the balance between detecting malicious files and avoiding false alarms.

- True Positives (TPs): Malicious samples correctly classified as malware.
- True Negatives (TNs): Benign samples correctly classified as benign.
- False Positives (FPs): Benign samples incorrectly classified as malware.
- False Negatives (FNs): Malicious samples incorrectly classified as benign.

From these fundamental measures, the following evaluation metrics were derived:

1. Accuracy (OA):

   Proportion of correctly classified samples among all predictions.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (1)$$

2. Precision (Positive Predictive Value, PPV):

   Proportion of correctly identified malware among all samples classified as malware.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (2)$$

3. Recall (Detection Rate, Sensitivity, or TPR):

   Ability of the model to correctly detect malware.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (3)$$

https://doi.org/10.3390/engproc2026168006

4. Specificity (True-Negative Rate, TNR):

Ability of the model to correctly detect benign files.

$$\text{Specificity} = \text{TN}/(\text{TN} + \text{FP}) \quad (4)$$

5. F1 Score:

Harmonic mean of precision and recall, providing a balance between false positives and false negatives.

$$\text{F1} = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall}) \quad (5)$$

6. ROC (Receiver Operating Characteristic) Curve:

A graphical plot of the true-positive rate (TPR) versus the false-positive rate (FPR) at various thresholds.

$$\text{TPR} = \text{TP}/(\text{TP} + \text{FN}), \text{FPR} = \text{FP}/(\text{FP} + \text{TN}) \quad (6)$$

7. AUC (Area under the ROC Curve):

Summarizes the ROC curve into a single value; higher values indicate stronger discrimination ability.

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, \text{d}(\text{FPR}) \quad (7)$$

2.6. Experimental Setup and Reproducibility

Experiments ran on an Intel Core i5-12450H (8 cores, 4.4 GHz) with 16 GB DDR4 RAM, an Nvidia RTX 3050 GPU (4 GB), and a 512 GB SSD under Windows 11 Pro. Software used Python 3.9 with pefile 2023.2.7 for feature extraction, numpy/pandas for data manipulation, scikit-learn 1.3 for modeling, and matplotlib/seaborn for visualizations. Random seeds were fixed for reproducibility. Entropy followed the Shannon formula; missing sections were assigned zero. Average feature extraction took 3.5 ms per file. Training on the 13-feature subset required <0.5 s, and inference per executable was 0.2 ms, confirming real-time suitability.

3. Results and Discussion

The proposed framework uses section-level PE attributes, RFECV with grid search, and an extra-trees classifier. Evaluation used five-fold cross-validation on 880 training samples and a held-out test set of 378 samples (70/30 split). Results are reported before and after feature selection, followed by comparison with prior work, limitations, and future directions.

3.1. Performance Analysis

Table 1 compares the full (61 features) and reduced (13 features) sets under five-fold CV and on the test set. The full set achieved 94.9% accuracy, 97.3% recall, and AUC 0.987. The reduced set achieved 94.8% CV accuracy and 95.5% test accuracy, with a slightly lower, but still excellent AUC (0.987). Reducing to 13 features cut dimensionality by 79%, improving efficiency and interpretability with negligible performance loss.

Table 1. Performance comparison (five-fold CV averages and test set).

Metric	Full (61)—CV	Reduced (13)—CV	Reduced (13)—Test
Accuracy	94.9%	94.8%	95.5%
Precision	92.8%	92.5%	92.6%
Recall	97.3%	97.5%	98.9%
Specificity	92.4%	92%	92.1%
F1 Score	95%	94.9%	95.7%
AUC	0.9870	0.9866	0.9869

On the held-out test set (378 samples), the reduced model correctly classified 174 benign (true negatives) and 187 malicious (true positives), with 15 false positives (benign misclassified as malware) and 2 false negatives (malware missed). This translates to a false-positive rate of 7.9% and a false-negative rate of 1.1%, which is realistic and competitive for a lightweight static detector.

The accuracy achieved across the five cross-validation folds for the reduced 13-feature model is presented in Table 2.

Table 2. Per-fold cross-validation accuracy for the reduced (13-feature) model.

Fold	Accuracy
1	97.2%
2	95.5%
3	95.5%
4	92%
5	93.8%
Mean ± std	**94.8% ± 1.7%**

To assess whether the reported performance depends on a single random initialization, we repeated the entire pipeline (train/test split, scaling, and model training) using five different random seeds (42, 123, 456, 789, 101,112). The test accuracy of the reduced model across these seeds was 95.3% ± 0.4% (mean ± standard deviation), with a 95% confidence interval of [94.6%, 96.1%]. The AUC remained stable at 0.9876 ± 0.0027. These narrow confidence intervals confirm that the framework's performance is not an artifact of a particular seed and that the results are reproducible.

To verify that the selected 13 features are genuinely discriminative and not merely a result of dimensionality reduction, we compared the RFECV-selected subset against random subsets of the same size (13 features). Over five independent random subsets, the average test accuracy was 92.6% ± 3.1%, which is significantly lower than the 95.5% achieved by the RFECV-selected features. This confirms that RFECV successfully identifies the most informative section-level attributes and that the performance gain is not simply due to reducing the number of features.

The results of the ablation study, comparing the full 61-feature model, the RFECV-selected 13-feature model, and randomly selected 13-feature subsets, are summarized in Table 3.

Table 3. Ablation study: test accuracy comparison.

Model	Test Accuracy
Full features (61)	95.5%
RFECV-selected (13)	95.5%
Random 13-feature subsets (average of 5 runs)	92.6%

https://doi.org/10.3390/engproc2026168006

The ROC curves (Figure 2) show nearly identical performance between the full and reduced sets, with the reduced set's AUC of 0.9866. The test confusion matrices (Figure 3) illustrate the distribution of errors.

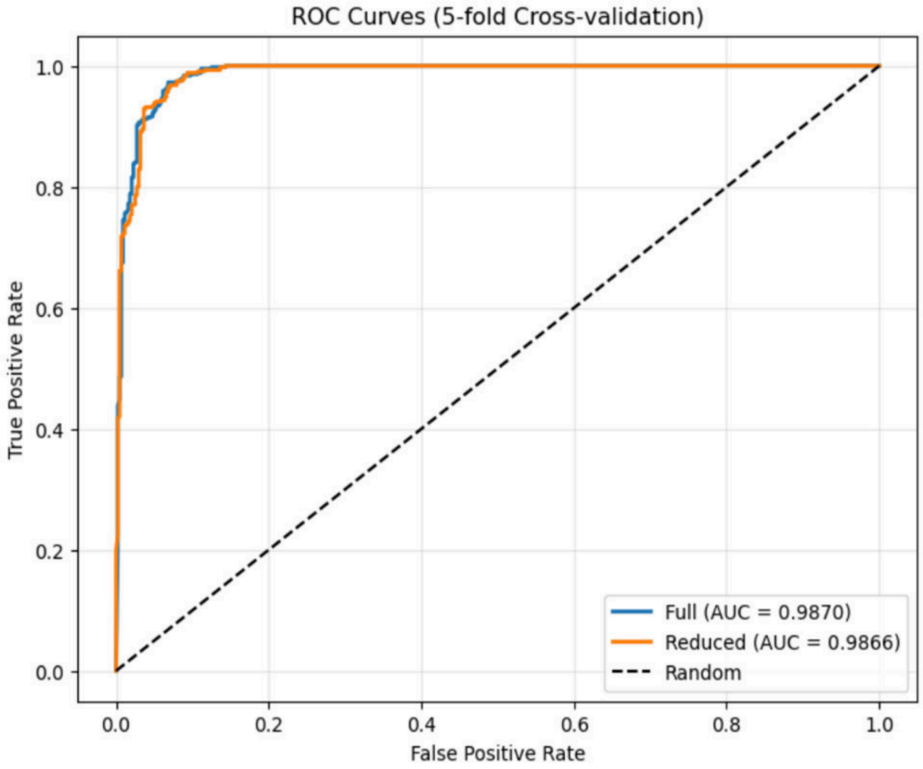
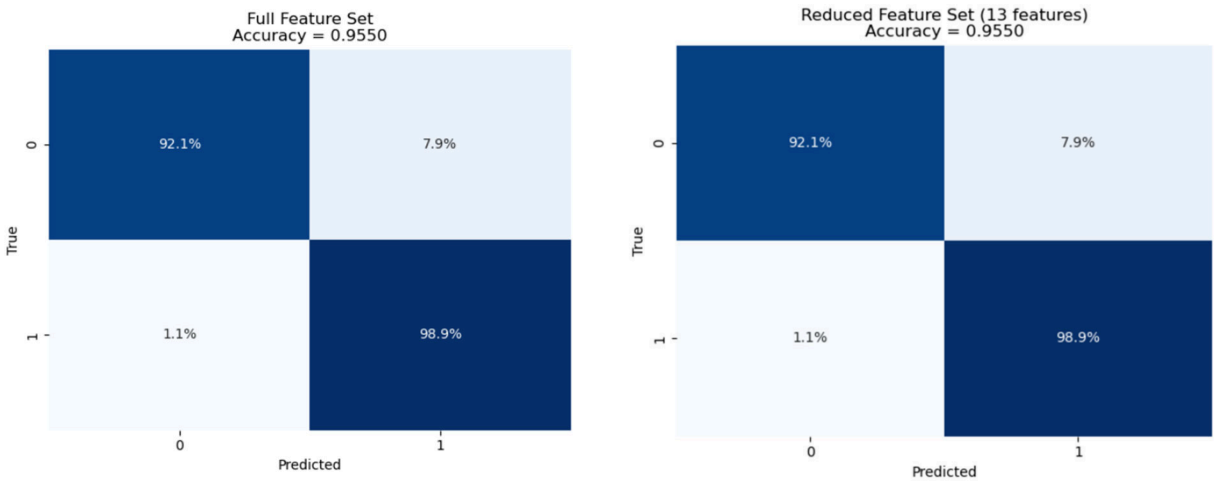


Figure 2. ROC curves before and after feature selection.



(**a**) (**b**)

Figure 3. (**a**) Confusion matrix for the full feature set. (**b**) Confusion matrix for the reduced feature set.

3.2. Comparison with Prior Work

Because each study uses different datasets, evaluation protocols, and malware/benign sources, direct numerical comparison across rows is not strictly equivalent. Table 4 therefore serves primarily to illustrate the relative scale of feature sets and the range of reported accuracies in the literature.

Table 4. Comparison with prior studies (selected).

Study	Modality	Feature Type	Feature Count	Classifier(s)	Accuracy (%)
Akhtar and Feng [14]	Dynamic	Behavior features from VirusTotal JSON (FS/Reg/Net/Proc)	373	kNN, DT, RF, AdaBoost, SGD, ET, GNB	100
Ayub et al. [20]	Static	PE metadata; imports/functions; PCA	5450	SVC, DT, RF, AdaBoost, GB	91.75
Yousuf et al. [21]	Static	Integrated IFS1/IFS2: DLLs, APIs, headers, sections	29,797	7 ML + 3 ensembles	99.5
Syeda and Asghar [15]	Dynamic	API frequencies grouped into 23 categories; Chi-Sq + Gini	631	RF, XGB, SVM, etc.	96
Baker Del Aguila et al. [22]	Static	57 metadata features; GA-optimized ANN	138,047	ANN, SVM, GBM	93.44
Kim et al. [16]	Dynamic	TF-IDF API/cat vectors; BiLSTM	16,590	BiLSTM	95.94
Maniriho et al. [19]	Dynamic	Predicted API sequences (GPT-2) → DistilBERT + BiGRU	72,099	GPT-2 + BiGRU pipeline	95.97
Miraoui and Belgacem [23]	Static	Mixed: headers, sections; classic ML + DL	>54	RF, SVM, NB, ANN; CNN/RNN/LSTM/ CNN-LSTM	88.76
Shahnawaz et al. [17]	Dynamic	API-argument images (magma colormap) → CNN	>4096	Custom CNN	98.36
Gond et al. [18]	Dynamic	API name/arg n-grams with TF-IDF; hybrid selection	22,054	DT, SVM, kNN, RF, XGB/LightGBM	99.02
Proposed Method	**Static**	**Section-level features**	**13**	**Extra trees**	**95.5**

Table 4 compares the proposed framework with previous static and dynamic approaches. Many existing methods achieve high accuracy but rely on large feature sets (hundreds to thousands of dimensions), dynamic execution (sandbox, slow, evasion-prone), or complex deep learning pipelines. In contrast, our framework uses only 13 static section-level features, requires no execution, and attains 95.5% test accuracy. This balance of accuracy, efficiency, and interpretability makes it suitable for resource-constrained environments such as IoT devices and edge systems.

Feature counts and accuracies are reported as published in each study. Our results are on a balanced dataset of 1258 samples with diverse malware families and mixed benign sources, using five-fold CV and a 70/30 test split. Other studies use different datasets (e.g., VirusTotal, custom collections, SOREL, EMBER) and evaluation protocols (e.g., different CV folds, temporal splits, or test sets).

3.3. Discussion

The results demonstrate that a small subset of section-level attributes (13 features) can achieve competitive malware detection accuracy (95.5%) while using fewer features by an order of magnitude than most prior work. Unlike dynamic approaches, our framework requires no sandbox execution and is immune to environment-aware evasion. Compared to static methods that rely on hundreds or thousands of features (e.g., API imports, opcodes), our 79% dimensionality reduction improves interpretability and reduces computational overhead.

The use of diverse malware families and mixed benign source set avoids the structural homogeneity of a single source and therefore provides a more realistic and challenging evaluation. The current performance (95.5% test accuracy) reflects the genuine difficulty of distinguishing malware from legitimate software that may be packed, obfuscated, or structurally unusual. This directly addresses concerns about dataset bias and overfitting.

The RFECV procedure retained 13 section-level attributes: total_sections, entropy_.text, .data, entropy_.data, .rdata, entropy_.rdata, raw_size_.rsrc, entropy_.rsrc, .bss, .reloc, entropy_.reloc, .pdata, and entropy_.pdata. High entropy in code or data sections (e.g., entropy_.text, entropy_.data, entropy_.rdata, entropy_.rsrc, entropy_.reloc, entropy_.pdata) is a known indicator of packing or encryption. The presence flags (.data, .rdata, .bss, .reloc, .pdata) and total_sections capture structural anomalies, while raw_size_.rsrc reflects unusual resource section sizes. This alignment with security knowledge makes the model transparent and trustable for analysts.

Recent DL-based approaches (e.g., MalConv, CNN-on-opcodes, transformer-based models) achieve comparable or slightly higher accuracy (98–99%) but require GPU acceleration, large training sets (millions of samples), and complex feature engineering. Hybrid static–dynamic methods, such as EarlyMalDetect, also report high accuracy but rely on sandbox execution and are vulnerable to evasion. In contrast, our framework uses only 13 static features, trains in <0.5 s on a CPU, and infers in 0.2 ms, making it uniquely suitable for real-time, resource-constrained environments.

Our dataset (1258 samples) is modest relative to large benchmarks like SOREL-20M, which may limit generalization to unseen families. Static structural features remain vulnerable to advanced packing, obfuscation, and polymorphism. While extra trees performed well, robustness against zero-day threats requires further validation.

Future work will include evaluation on larger benchmarks (EMBER, SOREL-20M), hybrid static–dynamic detection, and adversarial robustness testing.

4. Conclusions

This study introduced a lightweight and competitive static malware detection framework that leverages section-level attributes of Windows PE files. By applying RFECV and grid search optimization using the extra-trees classifier, the feature space was reduced from 61 to 13 highly discriminative attributes, achieving 94.9% cross-validation accuracy and 95.5% test accuracy on a held-out 30% split, with an AUC of 0.987. These results confirm that compact feature sets (79% dimensionality reduction) can rival much larger and more complex models while significantly improving computational efficiency and interpretability.

Compared with traditional dynamic analysis and high-dimensional static approaches, the proposed method offers clear advantages: reduced overhead, faster inference (0.2 ms per sample), and better scalability. These characteristics make it well suited for deployment in resource-constrained environments such as IoT devices, endpoint protection systems, and cloud-based detection pipelines.

Nonetheless, limitations remain. The framework relies exclusively on static features, which may be less effective against sophisticated obfuscation, polymorphism, and packing techniques. Furthermore, the dataset, while balanced and diverse (including both system binaries and user-space applications), is relatively modest in scale compared with large repositories such as SOREL-20M, and broader validation is necessary to confirm generalization to emerging malware families.

Future research should therefore explore hybrid static–dynamic detection strategies, investigate advanced DL architectures for richer feature representations, and evaluate performance on large-scale benchmark datasets such as EMBER and SOREL-20M. Enhancing

https://doi.org/10.3390/engproc2026168006

adversarial robustness and integrating real-time threat intelligence are also promising directions to ensure adaptability against evolving attack vectors.

In conclusion, the findings demonstrate that accurate and efficient malware detection can be achieved with a minimal set of structural PE features (13 attributes), highlighting the potential of the proposed framework as a deployable, scalable, and real-time cybersecurity solution.

Author Contributions: Conceptualization, M.K., J.-E.M. and Y.M.; methodology, M.K., J.-E.M. and Y.M.; software, M.K.; validation, Y.M. and K.E.G.; formal analysis, M.K.; investigation, M.K.; resources, M.K.; data curation, M.K. and J.-E.M.; writing—original draft preparation, M.K., J.-E.M. and Y.M.; writing—review and editing, M.K., J.-E.M., Y.M. and K.E.G.; visualization, M.K.; supervision, Y.M.; project administration, Y.M.; funding acquisition, Y.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset generated and analyzed during this study is not publicly available but is available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Falowo, O.I.; Ozer, M.; Li, C.; Abdo, J.B. Evolving malware and DDoS attacks: Decadal longitudinal study. *IEEE Access* **2024**, *12*, 39221–39237. [CrossRef]
2. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-Rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware detection issues, challenges, and future directions: A survey. *Appl. Sci.* **2022**, *12*, 8482. [CrossRef]
3. Faruki, P.; Bhan, R.; Jain, V.; Bhatia, S.; El Madhoun, N.; Pamula, R. A survey and evaluation of android-based malware evasion techniques and detection frameworks. *Information* **2023**, *14*, 374. [CrossRef]
4. Catalano, C.; Chezzi, A.; Angelelli, M.; Tommasi, F. Deceiving AI-based malware detection through polymorphic attacks. *Comput. Ind.* **2022**, *143*, 103751. [CrossRef]
5. Galloro, N.; Polino, M.; Carminati, M.; Continella, A.; Zanero, S. A systematical and longitudinal study of evasive behaviors in windows malware. *Comput. Secur.* **2022**, *113*, 102550. [CrossRef]
6. Muzaffar, A.; Hassen, H.R.; Zantout, H.; Lones, M.A. Reassessing feature-based Android malware detection in a contemporary context. *PLoS ONE* **2026**, *21*, e0341013. [CrossRef] [PubMed]
7. Sahay, S.K.; Sharma, A.; Rathore, H. Evolution of Malware and Its Detection Techniques. In *Information and Communication Technology for Sustainable Development*; Tuba, M., Akashe, S., Joshi, A., Eds.; Springer: Singapore, 2020; Volume 933, pp. 139–150. [CrossRef]
8. Yan, S.; Ren, J.; Wang, W.; Sun, L.; Zhang, W.; Yu, Q. A Survey of Adversarial Attack and Defense Methods for Malware Classification in Cyber Security. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 467–496. [CrossRef]
9. Akhtar, M.S.; Feng, T. Evaluation of machine learning algorithms for malware detection. *Sensors* **2023**, *23*, 946. [CrossRef] [PubMed]
10. Syeda, D.Z.; Asghar, M.N. Dynamic malware classification and API categorisation of windows portable executable files using machine learning. *Appl. Sci.* **2024**, *14*, 1015. [CrossRef]
11. Kim, M.; Kim, H. A dynamic analysis data preprocessing technique for malicious code detection with TF-IDF and sliding windows. *Electronics* **2024**, *13*, 963. [CrossRef]
12. Shahnawaz, M.; Gond, B.P.; Mohapatra, D.P. Dynamic malware classification of windows PE files using CNNs and greyscale images derived from runtime API call argument conversion. *arXiv* **2025**, arXiv:2505.24231. [CrossRef]
13. Gond, B.P.; Rajneekant; Kishore, P.; Mohapatra, D.P. Malware classification leveraging NLP & machine learning for enhanced accuracy. *arXiv* **2025**, arXiv:2506.16224. [CrossRef]
14. Nunes, M.; Burnap, P.; Reinecke, P.; Lloyd, K. Bane or boon: Measuring the effect of evasive malware on system call classifiers. *J. Inf. Secur. Appl.* **2022**, *67*, 103202. [CrossRef]
15. Kim, M.; Cho, H.; Yi, J.H. Large-Scale Analysis on Anti-Analysis Techniques in Real-World Malware. *IEEE Access* **2022**, *10*, 75802–75815. [CrossRef]

16. Muralidharan, T.; Cohen, A.; Gerson, N.; Nissim, N. File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements. *ACM Comput. Surv.* **2023**, *55*, 108. [CrossRef]
17. Maniriho, P.; Mahmood, A.N.; Chowdhury, M.J.M. EarlyMalDetect: A novel approach for early windows malware detection based on sequences of API calls. *arXiv* **2024**, arXiv:2407.13355. [CrossRef]
18. Venčkauskas, A.; Jusas, V.; Barisas, D. Zero-Day Ransomware Attack Detection Using Static Portable Executable Header Features. *Appl. Sci.* **2025**, *15*, 10576. [CrossRef]
19. Demirci, D.; Sahin, N.; Sirlancis, M.; Acartürk, C. Static Malware Detection Using Stacked BiLSTM and GPT-2. *IEEE Access* **2022**, *10*, 58488–58502. [CrossRef]
20. Ayub, M.A.; Siraj, A.; Filar, B.; Gupta, M. Static-RWArmor: A static analysis approach for prevention of cryptographic windows ransomware. In Proceedings of the 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Exeter, UK, 1–3 November 2023; pp. 1673–1680. [CrossRef]
21. Yousuf, M.I.; Anwer, I.; Riasat, A.; Zia, K.T.; Kim, S. Windows malware detection based on static analysis with multiple features. *PeerJ Comput. Sci.* **2023**, *9*, e1319. [CrossRef] [PubMed]
22. Baker Del Aguila, R.; Contreras Pérez, C.D.; Silva-Trujillo, A.G.; Cuevas-Tello, J.C.; Nunez-Varela, J. Static malware analysis using low-parameter machine learning models. *Computers* **2024**, *13*, 59. [CrossRef]
23. Miraoui, M.; Belgacem, M.B. Binary and multiclass malware classification of windows portable executable using classic machine learning and deep learning. *Front. Comput. Sci.* **2025**, *7*, 1539519. [CrossRef]
24. Han, J.; Kamber, M.; Pei, J. *Data Mining: Concepts and Techniques*, 3rd ed.; Elsevier: Amsterdam, The Netherlands, 2011.
25. Kuhn, M.; Johnson, K. *Applied Predictive Modeling*; Springer: New York, NY, USA, 2013.
26. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY 2016), New Orleans, LA, USA, 9–11 March 2016; pp. 183–194. [CrossRef]
27. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C.K. Malware Detection by Eating a Whole EXE. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018. Available online: https://cdn.aaai.org/ocs/ws/ws0432/16422-75958-1-PB.pdf (accessed on 12 November 2026).
28. Guyon, I.; Weston, J.; Barnhill, S.; Vapnik, V. Gene selection for cancer classification using support vector machines. *Mach. Learn.* **2002**, *46*, 389–422. [CrossRef]
29. Saeys, Y.; Inza, I.; Larrañaga, P. A review of feature selection techniques in bioinformatics. *Bioinformatics* **2007**, *23*, 2507–2517. [CrossRef] [PubMed]
30. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
31. Wong, T.-T. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognit.* **2015**, *48*, 2839–2846. [CrossRef]
32. Vabalas, A.; Gowen, E.; Poliakoff, E.; Casson, A.J. Machine learning algorithm validation with a limited sample size. *PLoS ONE* **2019**, *14*, e0224365. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.