Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.063046

ARTICLE

OMD-RAS: Optimizing Malware Detection through Comprehensive Approach to Real-Time and Adaptive Security

Farah Mohammad[1,2,*], **Saad Al-Ahmadi**[1,3] **and Jalal Al-Muhtadi**[1,3]

[1]Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, 11543, Saudi Arabia
[2]Department of Computer Science, and Technology, Arab East Colleges, Riyadh, 11583, Saudi Arabia
[3]College of Computer & Information Sciences, King Saud University, Riyadh, 11543, Saudi Arabia

*Corresponding Author: Farah Mohammad. Email: fnazar@ieee.org

Received: 03 January 2025; Accepted: 25 March 2025; Published: 30 July 2025

ABSTRACT: Malware continues to pose a significant threat to cybersecurity, with new advanced infections that go beyond traditional detection. Limitations in existing systems include high false-positive rates, slow system response times, and inability to respond quickly to new malware forms. To overcome these challenges, this paper proposes OMD-RAS: Implementing Malware Detection in an Optimized Way through Real-Time and Adaptive Security as an extensive approach, hoping to get good results towards better malware threat detection and remediation. The significant steps in the model are data collection followed by comprehensive preprocessing consisting of feature engineering and normalization. Static analysis, along with dynamic analysis, is done to capture the whole spectrum of malware behavior for the feature extraction process. The extracted processed features are given with a continuous learning mechanism to the Extreme Learning Machine model of real-time detection. This OMD-RAS trains quickly and has great accuracy, providing elite, advanced real-time detection capabilities. This approach uses continuous learning to adapt to new threats—ensuring the effectiveness of detection even as strategies used by malware may change over time. The experimental results showed that OMD-RAS performs better than the traditional approaches. For instance, the OMD-RAS model has been able to achieve an accuracy of 96.23% and massively reduce the rate of false positives across all datasets while eliciting a consistently high rate of precision and recall. The model's adaptive learning reflected enhancements on other performance measures—for example, Matthews Correlation Coefficients and Log Loss.

KEYWORDS: Malware; adaptive security; feature engineering; ELM; Kafka

1 Introduction

Malware refers to malicious software, which means all kinds of malware are intentionally designed to cause damage, disrupt operations, or gain unauthorized access to a computer system. Viruses have been in use since the oldest times and were replicated to move from one system to another [1]. Ransomware encrypts files on a victim's device and demands a ransom for its release. Trojan viruses are malware masquerading as genuine software, tricking users into installing the virus in their systems. Worms, on the other hand, are stand-alone programs that may self-replicate moving from one system to another. Spyware is used to track a user's clandestine activities and pass this information on to another party. Adware displays undesired advertisements and slows down systems. Fileless malware resides in memory; hence, it leaves no trace on the disk. Cryptojacking seizes a computer's resources to mine cryptocurrencies without the user's knowledge or consent [2]. Wiper malware aims at irreversibly deleting or wiping data, definitively causing a disruption to the owner, while rootkits grant continuous access to a system, concealing their existence from it. These various kinds of malware pose very clear and different threats; protection against each must be comprehensive.

Such malware can be very devastating because it results in massive financial losses and data breaches, mainly leading to operational disruptions [3]. It exposes businesses to the high risk of losing very vital data, including info on customers and intellectual property, thus leading to damaged reputation and potential legal implications. Ransomware attacks grind operations to a halt, forcing organizations to pay out huge ransoms or result in the very expansive process of recovering data. It can result in unauthorized and undue access to systems, which might allow cyber criminals to pilfer funds or manipulate data and also launch an attack further [4]. Besides spyware and adware can leak information about user activities, slacken system performance, and increase additional IT costs for security upgrades and cleaning up a system. It can further compound the financial and operational damage due to malware by some of its indirect costs in the form of lost productivity and customer trust.

Common traditional malware detection techniques include signature-based detection and heuristic-based detection. Signature-based detection involves the presentation of known patterns of the malware, known as 'signatures'. This is achieved by comparing the code of files or a program against the database for holding the signature of known malware [5]. While it was acknowledged as being very effective for known threats, it had limitations in detecting new, evolving malware that does not have the features or "fingerprints" for which it was watching. It requires continuous updating of the signature database and does not support zero-day attacks, in case the malware has just been released to the public and the signatures remain updated [6]. Heuristic-based detection makes an attempt to detect a malware by the behaviour or structure of a program and therefore is not akin to any known signatures.

The methodology paves the base to recognize unknown malware by identifying suspicious activities such as uncommon file modifications or system calls. On the other hand, heuristic analysis can produce false positives—exposing normal software as PC malware—thereby unnecessarily creating disruptions. Both methods also struggle to detect advanced threats, such as fileless malware working in memory and leaving no traces on the disk, or polymorphic malware that modifies its own code numerous times to avoid detection [7]. Among other weaknesses of traditional methods is that they usually consume most of the resources from the system being scanned, significantly slowing down performance and hence making the detection process less efficient. These limitations have raised the need for further sophisticated methods of detection, such as behavioral analysis, machine learning, and artificial intelligence, in enhancing the fight against these modern malware threats [8].

Computer-based techniques are those using software tools or programs to detect, prevent, and delete malicious software from a computer system. Normally, these include antivirus programs, firewalls, and intrusion detection systems [9]. It means that antivirus programs search through files, applications, and system activities in a hunt for malware that might supposedly have a known signature or behave suspiciously. Firewalls monitor incoming and outgoing network traffic and block those of them that reveal hostile connections. Intrusion detection systems analyze network or system activities for any signs of malicious activities or violations of security policies [10].

However, the malware detection methods, which are computer-based, have lots of limitations. Firstly, most of them are signature-based, meaning they can detect only known malware. Consequently, new malware, the unknown ones, or those that change very fast—zero-day threats, for instance—are hard to detect since they have yet to have a known signature assigned to them [11]. Modern techniques of malware, such as polymorphism, whereby the malware changes its code frequently to avoid detection, are fully capable of evading traditional approaches to detection entirely based on the computer. Another limitation is related to the potential false positive and negative detection. A 'false positive' means unwarranted alerts or actions against righteous software or activities that are misinterpreted as bad. Conversely, 'false negatives' are cases of actual malware that go undetected and, hence, are allowed to continue their operations unabated. However, all of these detection techniques are resource-intensive, using high processing power and slowing down the system in the case of full scans or large data volumes [12]. Deep learning-based malware detection tools use state-of-the-art neural networks to analyze large streams of data for patterns indicative of malicious behavior. Deep learning models, as opposed to those using traditional approaches that rely on predefined signatures or simple heuristics, can automatically learn complex features from raw data itself, be it source code, network traffic, or system logs. In particular, they are good at detecting previously unknown or evolving malware by identifying subtle, non-obvious correlations and patterns that might otherwise go undetected using traditional approaches [13].

Deep learning architectures present notable advantages in malware detection, particularly their capacity to process extensive datasets and exhibit continuous performance optimization through incremental exposure to new samples. These systems demonstrate robust capabilities in identifying diverse malware variants, including fileless, polymorphic, and zero-day threats, through comprehensive behavioral analysis and characteristic pattern recognition, transcending traditional signature-based detection methodologies [14]. Nevertheless, deep learning-based detection systems face several inherent challenges. A primary constraint lies in their dependency on large-scale, accurately labeled training datasets, the acquisition and annotation of which demands substantial resources and expertise. Moreover, the inherent opacity of deep learning models—often characterized as 'black box' systems—poses significant challenges in decision interpretability. This lack of algorithmic transparency impedes security practitioners' ability to comprehend detection rationales and false positive occurrences, subsequently raising concerns regarding system reliability and compliance with regulatory frameworks [15]. In conclusion, OMD-RAS presents a significant advancement in the field of cybersecurity by integrating comprehensive data preprocessing, robust feature extraction, and the powerful Extreme Learning Machine (ELM) for real-time detection. By combining static and dynamic analysis, OMD-RAS captures a wide range of malware behaviors, ensuring thorough detection and mitigation of threats. Its continuous learning mechanisms enable the model to adapt to emerging threats, maintaining high accuracy and relevance over time. The rapid training capabilities of ELM further enhance the model's efficiency, making OMD-RAS a superior solution for combating modern malware threats in real-time, setting a new standard in adaptive security measures.

Research Contribution

The key contributions of the paper are as follows:

- Integration of comprehensive data preprocessing and dual-mode feature extraction (static and dynamic analysis) to capture a broad spectrum of malware behaviors.
- Implementation of the Extreme Learning Machine (ELM) for real-time detection, enabling rapid training and high accuracy in identifying malware threats.
- Continuous learning mechanisms that allow the model to adapt to emerging threats, ensuring sustained accuracy and effectiveness in a dynamic cybersecurity landscape.

The rest of the paper is organized as follows. In Section 2, we provide an overview of related work in the field of malware detection. Section 3 describes the proposed approach in detail. Section 4 presents the experimental results and evaluates the effectiveness of the proposed approach. Finally, Section 5 concludes the paper and discusses future research directions.

2 Related Work

The detection of malware, as described in Table 1, has been an active research area in the field of cybersecurity for many years. Researchers have explored a wide range of approaches to detect malware, including signature-based systems, behavior-based systems, and machine learning-based systems.

Shaukat et al. proposed a deep learning-based framework that enhances malware detection by transforming Windows PE files into colored images, eliminating the need for manual feature extraction [16]. A fine-tuned deep model extracts deep features, reducing training costs, followed by a powerful feature selection algorithm. The selected features are fed into a one-class classifier, which constructs a boundary around benign data, detecting anomalies as malware. This approach improves the detection of unseen, polymorphic, and zero-day attacks while mitigating overfitting. The framework's effectiveness is validated against state-of-the-art deep learning and conventional methods. Puneeth et al. proposed that a Robust Malware Detection Network (RMDNet) improves malware detection and classification using depth-wise convolution and concatenation operations [17]. It is evaluated on a dataset of 48,240 malware images (RGB) and multi-class malware and dumpware-10 datasets (grayscale), demonstrating high accuracy. By effectively learning distinguishing patterns, RMDNet outperforms recent deep learning models in detecting and classifying malware. Its superior performance highlights its potential for enhancing cybersecurity by identifying emerging threats with improved precision.

Similarly, Liu et al. [18] proposed another GCN-based framework for malware detection and classifying them into families based on behavioral characteristics. They extracted the system call graph of malware, which represents the relations between system calls during its execution. In this graph, the nodes are the system calls; the edges represent their dependencies. SCG is generated in a dynamic analysis using malware execution traces, while it gives information about runtime behavior. GCN operates on the topological structure of SCG to learn node-related features, characterizing specific system calls, and graph-related features, capturing the overall behavior of the malware. The performance of their approach was tested against two datasets that contain samples of malware, which showed a high accuracy in both detection and classification. It outperformed the other baseline methods, including the SVM approach and the CNN approach.

Another study analyzes the effectiveness of AutoML for both static and online malware detection [19]. For static detection, experiments on SOREL-20M and EMBER-2018 datasets assess performance on large and challenging datasets, optimizing NAS parameters for improved model selection. For online detection, AutoML is evaluated using CNNs in cloud IaaS environments and compared to six state-of-the-art CNNs on a newly generated dataset. Results show that AutoML outperforms traditional CNNs with minimal overhead in architecture optimization.

Table 1: Summarized literature review

Ref.	Methodology	Dataset	Limitations
[20]	CNN	Drebin dataset (215 features)	• High computational cost for training CNNs. • Potential overfitting due to high model accuracy.
[21]	FeSAD, with CNN and LSTM	EMBER-2018	• May not generalize well to non-ransomware malware. • Requires frequent retraining to handle concept drift effectively.

(Continued)

Table 1 (continued)

Ref.	Methodology	Dataset	Limitations
[22]	AutoML, PCA	SOREL-20M, EMBER-2018	• High computational overhead for AutoML processes. • May not always find the optimal architecture for all datasets.
[23]	CNN-BiLSTM and deep autoencoders	Network traffic data (NTD)	• Complexity in combining static and dynamic features. • Requires large amounts of labeled data for training.
[24]	Hybrid approach using CNN and LSTM with PCA	Public malware dataset, network traffic data	• Conversion of binaries to images may lose some details. • High computational requirements for hybrid models.
[25]	Deep learning with feature selection for malware detection	Two unspecified datasets	• Performance degradation with aggressive feature reduction. • High variability in results across different datasets.

The limitations identified in existing malware detection techniques highlight challenges such as high computational costs, overfitting, and performance degradation under varying conditions. For instance, deep learning models can be resource-intensive and prone to overfitting, while frameworks like FeSAD may struggle with generalizing to non-ransomware threats and require frequent updates. Similarly, AutoML techniques, while reducing manual effort, can involve substantial computational overhead and may not always optimize the model architecture effectively. In contrast, the proposed OMD-RAS model addresses these limitations with a robust and adaptable approach. By integrating comprehensive data collection, preprocessing, feature engineering, and both static and dynamic feature extraction, OMD-RAS ensures a thorough and nuanced understanding of malware behavior. The Extreme Learning Machine (ELM) component further enhances this by providing real-time detection capabilities with reduced computational overhead and high accuracy. OMD-RAS stands out for its ability to deliver superior performance while minimizing resource demands and adapting effectively to evolving threats, setting a new standard in malware detection and security.

3 Proposed Methodology

This section will cover the proposed OMD-RAS that has been depicted in Fig. 1 and is composed of data collection, followed by comprehensive preprocessing, which includes feature engineering and normalization. Feature extraction is performed using both static and dynamic analysis to capture the full spectrum of malware behavior. The processed features are then fed into the Extreme Learning Machine (ELM) model for real-time detection. The details of each phase have been described in the subsections below.

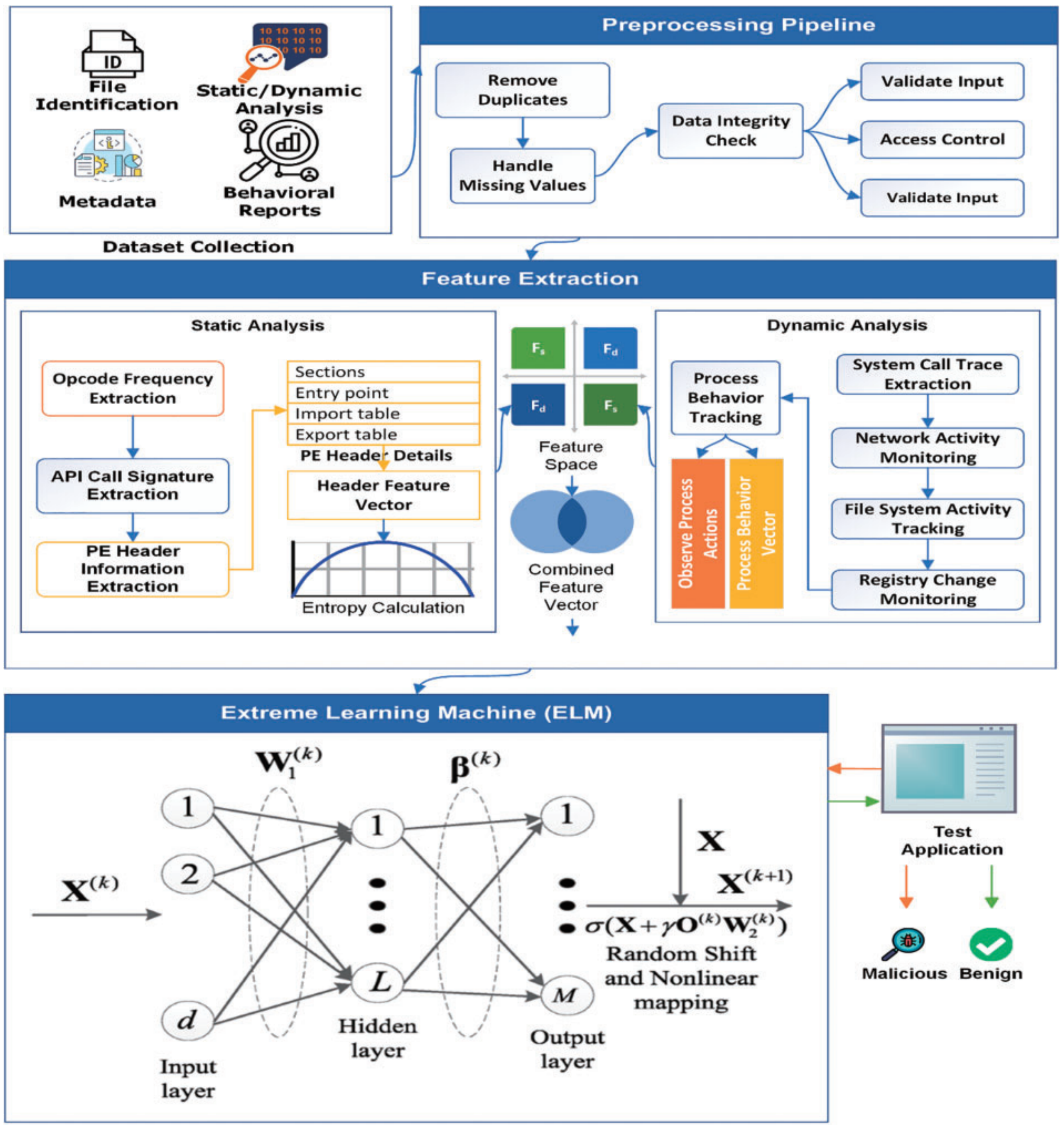


Figure 1: The architecture of OMD-RAS

3.1 Data Collection

The data collection process for malware detection involves gathering comprehensive datasets from various reputable sources, such as MalwareBazaar (MWD-I), VirusShare (MWD-II) and the Malicia Project (MWD-III). This allows for the acquisition of versatile and quite large sets of both benign and malicious software, which is perfect for training strong malware detection models. MalwareBazaar is a collaboration platform where contributors submit samples of malware research and security practitioners. This Malware-Bazaar dataset consists of completely diverse types of malware files, such as ransomware, Trojans, and worms. Each malware sample is thoroughly labeled with metadata, which consists mostly of file hash values, including MD5, SHA-1, and SHA-256, but also contains file size, first seen, and contextual tags of threat intelligence. This information is key in discerning the characteristics of each malware sample. The dataset description is given in Table 2.

Table 2: Dataset description

Category	Feature	Description	Source
File identification	File hashes	MD5, SHA-1, SHA-256 hashes	MalwareBazaar, VirusShare
	File size	Size of the file in bytes	
	File type	Type of the file (e.g., .exe, .dll, .apk)	
Metadata	First-seen date	Date the sample was first observed	MalwareBazaar, VirusShare
	Threat intelligence tags	Tags describing the malware	
Static analysis	Opcode frequencies	Frequencies of operation codes	All sources
	API call signatures	List and frequency of API calls	
	Entropy scores	Measure of randomness	
Dynamic analysis	System call traces	Sequences of system calls made during execution	Malicia project
	Network activities	Network behaviors (e.g., IP addresses, ports)	
Behavioral reports	Detailed logs	Logs of the infection process	
Labels	Malicious/Benign	Indicates if the sample is malicious or benign	All sources
	Malware family	Classification of the malware family	MalwareBazaar, VirusShare

VirusShare provides one of the biggest sets of malware samples available, with over 30 million unique files. Attributes of a file, like file hashes, timestamps, file types, and known signatures or indicators of compromise, are structured into the dataset by VirusShare. The size of this repository, along with the variety of samples covering a wide range of malicious activities, is invaluable for researchers working on the development and testing of malware detection algorithms. Malicia Project provides one of the datasets oriented at real malware samples collected from drive-by download attacks. Logs of the process of infection, dynamic traces of behavior, and network traffic data supplement the dataset. Each malware sample in the Malicia project is accompanied by hugely detailed behavioral reports, including, but not limited to, system calls, file-system modifications, registry changes, and network connections. It contains very fine-grained information about malware behaviors that enables in-depth analysis for understanding such behaviors, hence making it a very important resource for dynamic analysis and feature extraction.

3.2 Preprocessing and Feature Engineering

In the data preprocessing and future engineering phase for malware detection, several advanced techniques are employed to transform raw data into informative features that enhance the model's performance.

This phase involves dimensionality reduction, feature selection, feature transformation, data normalization, and data augmentation. The preprocessing and feature engineering steps are given in Algorithm 1.

Algorithm 1: Processing and feature engineering

```
                    Input: $X \in R^{n\times p}$ (Data matrix), y$\in \{0,1\}$(Target vector), $\lambda$ (Regularization parameter), k
                    (Number of principal components), m (Number of features to select), l (Number of latent
                    dimensions)
                    Output: X′ (Preprocessed data matrix) and y′ (Preprocessed target vector)
                    Dimensionality Reduction:
1                       $Compute\ covariance\ matrix : \sum = \frac{1}{n} X^T X$
2                       $Eigen\ decomposition : \Sigma = W \Lambda W^T$
3                       $Sort\ eigenvectors : W = [w_1, w_2, \ldots, w_p]$
4                       $Select\ top\ k\ eigenvectors : WPCA = [w_1, w_2, \ldots, w_k]$
5                       $Transform\ data : X_{PCA} = XW_{PCA}$
6                       $Compute\ scatter\ matrices : SB = \sum_{i=1}^{c} cni(\mu i - \mu)(\mu i - \mu)^T$T
7                   $Transform\ data : X_{LDA} = XW_{LDA}$
8                  $\boldsymbol{Initialize}\ features : F = \{1, 2, \ldots, p\}$
9                  $\boldsymbol{For}\ i = p\ to\ m$
                         $Train\ model\ and\ compute\ feature\ importance : I(f)$
                        $Remove\ feature\ with\ lowest\ importance : F = F \backslash \{argminI(f)\}$
                        $Select\ features\ with\ non-zero\ coefficients : F_{lasso} = \{J|\beta_j \neq 0\}$
                       $Compute\ mutual\ information : I(X_j; y)$
                    $Select\ top\ mmm\ features : FMI = \{j \mid I(X_j; y)\ is\ top\ m\}$
10                 End For
11                 For each $feature\ x_j$
                       Do
                           $x' = \frac{x - \min(x)}{max\,(x) - \min(x)}$
12                 End For
13                 Return $X'$ and $y'$
```

To begin, dimensionality reduction was employed to decrease the number of features while preserving the most essential information. This study utilizes Principal Component Analysis (PCA) [26] to map the data onto a lower-dimensional space. PCA works by identifying the principal components—eigenvectors of the covariance matrix—that represent the directions of maximum variance in the data. Mathematically, given the data matrix X, the transformation is:

$$X' = XW \tag{1}$$

where W is the matrix of principal component vectors (eigenvectors). This reduces the dimensionality of X while retaining the most variance. Afterward, Linear Discriminant Analysis (LDA) [27] is used to maximize class separability by projecting the data onto a lower-dimensional space where the ratio of between-class variance (S_B) to within-class variance (S_W) is maximized. The transformation matrix W is obtained by solving:

$$W = argmax_w \frac{W^T S_B W}{W^T S_W W} \tag{2}$$

In selecting the most relevant features for the model, Recursive Feature Elimination (RFE) [28] has been applied to iteratively remove the least important features based on their weights in a given model. L1 Regularization (Lasso) is another method where less important feature coefficients are shrunk to zero during model training, effectively selecting a subset of features. The Lasso regression objective function is:

$$min_\beta \left(\frac{1}{2n} \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^{p} |B_j| \right) \tag{3}$$

where λ controls the degree of regularization. Mutual Information measures the dependency between each feature and the target variable, selecting features with the highest mutual information scores. Feature Transformation techniques further process the selected features to make them more informative. Autoencoders are neural networks that learn to encode input data into a lower-dimensional representation and decode it back. The encoding process can be represented as:

$$h = f(Wx + b) \tag{4}$$

where h is the hidden representation, W and b are the weights and biases, and f is the activation function. Variational Autoencoders (VAE) extend this by mapping input data to a distribution in the latent space, typically Gaussian, and the objective combines reconstruction loss and Kullback-Leibler divergence:

$$\mathcal{L} = E_{q(z|x)}[logp(x \mid z)] - D_{KL}(q(z \mid x) \parallel p(z)) \tag{5}$$

Sparse Coding represents data as sparse linear combinations of basis vectors, capturing the essential structure of the data. Given a data vector x, Sparse Coding finds a sparse representation a such that:

$$x \approx Dax \tag{6}$$

where D is the dictionary of basis vectors. Later on data normalization has been applied to scales the selected features to a common range, typically [0, 1], using min-max scaling:

$$x' = \frac{x - \min(x)}{max\,(x) - \min(x)} \tag{7}$$

This ensures that features contribute equally to the model.

In this phase, various techniques were carefully chosen to enhance model performance. Principal Component Analysis (PCA) was employed for dimensionality reduction as it effectively preserves variance while reducing feature space complexity. Linear Discriminant Analysis (LDA) was selected to maximize class separability, making it particularly useful for malware classification. Recursive Feature Elimination (RFE) was preferred for feature selection due to its iterative approach, which systematically removes less significant features, while Lasso regression further refined the selection by shrinking irrelevant feature coefficients to zero. These techniques were chosen over alternatives such as t-SNE or autoencoders, which, despite their advantages, may introduce interpretability challenges or higher computational overhead. Additionally, while OMD-RAS provides robust detection capabilities, certain challenges exist, including potential computational overhead in real-time adaptation, increased false positives in dynamic conditions, and the necessity for efficient resource allocation in large-scale deployments. Addressing these limitations remains an area for future optimization

3.3 Feature Extraction

In the feature extraction phase for malware detection, both static and dynamic analyses are performed to gather comprehensive features that can effectively differentiate between benign and malicious software. Static analysis involves examining the executable without executing it, focusing on extracting opcode frequencies, API call signatures, PE header information, and entropy scores. To extract opcode frequencies, the binary is disassembled, and the occurrences of each opcode are counted. Let O_i denote the frequency of the ith opcode in the binary, then the feature vector for opcode frequencies is $O = [O_1, O_2, \ldots, O_n]$, where n is the total number of distinct opcodes.

API call signatures are extracted by analyzing the import table of the binary to list the API functions called. If A_i represents the occurrence of the ith API call, the API call signature vector is $A = [A_1, A_2, \ldots, A_m]$, where m is the total number of distinct API calls. PE (Portable Executable) header information is another critical feature in static analysis. It includes details like the number of sections, entry point, import table, and export table. Let PE_i be the ith feature extracted from the PE header, then the PE header feature vector is $PE = [PE_1, PE_2, \ldots, PE_k]$, where k is the number of PE header features. Entropy scores measure the randomness in different sections of the binary, indicating potential packing or obfuscation. The entropy H for a section can be computed using the formula

$$H = -\sum_i p_i \log_2 p_i \tag{8}$$

where p_i is the probability of the ith byte value.

Dynamic analysis involves executing the malware in a controlled environment and monitoring its behavior to collect system call traces, network activities, file system activities, registry changes, and process behaviors. System call traces capture the sequence of system calls made by the executable during its execution. Let $SC = [sc_1, sc_2, \ldots, sc_l]$ represent the system call sequence, where l is the length of the trace. Network activities are recorded by monitoring the connections initiated by the executable, including IP addresses, ports, and protocols used. If N_i represents the ith network activity, the network activity vector is $N = [N_1, N_2, \ldots, N_p]$.

File system activities track operations such as file creation, deletion, and modification. Let F_i denote the ith file system activity, then the file system activity vector is $F = [F_1, F_2, \ldots, F_q]$. Registry changes capture modifications to the Windows registry, crucial for understanding the malware's persistence mechanisms. If R_i is the ith registry change, the registry change vector is $R = [R_1, R_2, \ldots, R_r]$. Process behaviors include actions like process creation, termination, and code injection. Let P_i denote the ith process behavior, then the process behavior vector is $P = [P_1, P_2, \ldots, P_s]$. By extracting these static and dynamic features, we obtain a comprehensive representation of the malware, enabling the detection model to effectively distinguish between benign and malicious software. The combined feature set X can be represented as $X = [O, A, PE, H, SC, N, F, R, P]$, providing a holistic view of the executable's characteristics and behaviors.

Table 3 shows the list of some of the extracted feature after the feature extraction phase for malware detection. Static analysis involves extracting opcode frequencies, such as '*mov*' (250 occurrences) and push (120 occurrences), which provide insights into the code patterns used by the executable. API call signatures, including '*CreateFile*' (15 calls) and '*ReadFile*' (20 calls), are identified by examining the import table, indicating the interactions the executable has with the system. PE header information, such as the number of sections (5) and entry point address (0 × 401,000), reveals structural details of the binary, while entropy scores like the code section entropy (6.5) measure the randomness within different sections, hinting at potential obfuscation. Dynamic analysis captures runtime behaviors, including system call traces (‘*NtCreateFile*’ called 12 times), network activities (5 connections initiated), file system activities (4 files created), registry changes (3 keys created), and process behaviors (2 processes created). These features collectively provide a detailed representation of the executable’s static characteristics and dynamic actions, crucial for effectively distinguishing between benign and malicious software.

Table 3: Some of the extracted features list after feature extraction process

Feature category	Feature name	Value of each feature
Opcode frequencies	‘mov’, ‘Push’, ‘call’, ‘Jmp’, ‘xor’	250, 120, 95, 45, 30
API call signatures	‘CreateFile’, ‘ReadFile’, ‘WriteFile’, ‘CloseHandle’, ‘GetProcAddress’	15, 20, 10, 25, 5
PE header information	Number of sections	5
	Entry point	0 × 401,000
	Size of code section	102,400 bytes
	Number of imported functions	50
	Number of exported functions	10
Entropy scores	Code section entropy	6.5
	Data section entropy	4.2
System call traces	NtCreateFile	12
	NtWriteFile	8
	NtClose	10
Network activities	Connections initiated	5
	IP addresses	3
	Ports	3
File system activities	Files created	4
	Files deleted	2
Registry changes	Keys created	3
	Keys modified	2
Process behaviors	Processes created	2
	Processes terminated	1
	Code injections	1

3.4 Final Malware Detection

The Extreme Learning Machine is a powerful and efficient algorithm specifically designed for SLFNs. In malware detection, ELM has striking advantages with regard to the fast training process and strong generalization ability of the model. Principally, ELM is randomly initialized with input weights and biases, and then it calculates the output weights analytically to significantly reduce training time compared with traditional neural networks.

In the case of malware detection, the feature extraction phase provides a set of features that serve as inputs to the ELM. These features, denoted as $X \in R^{n \times p}$ where n is the number of samples and p is the number of features, are fed into the input layer of the ELM. The hidden layer of ELM consists of L neurons with randomly assigned weights $W \in R^{p \times l}$and biases $b \in R^{L}$. The hidden layer output H is computed using an activation function $g(\cdot)$, which can be a sigmoid, hyperbolic tangent, or any other non-linear function. The hidden layer output is given by $H = g(XW + b)$ where $g(XW + b)$ denotes the elementwise application of the activation function. The objective of ELM is to find the output weights $\beta \in R^{L \times m}$ that minimize the error between the predicted outputs $H\beta$ and the actual target values $T \in R^{n \times m}$, where mmm is the number of output classes (e.g., benign or malicious). The output weights are determined by solving the following equation:

$$\beta = H^{\dagger} T \tag{9}$$

where $H^{\dagger}$ is the Moore-Penrose pseudoinverse of the hidden layer output matrix H. In the context of malware detection, once the ELM is trained, it can be used to classify new samples by computing their hidden layer output and applying the learned output weights. Given a new feature vector x_{new}, the classification decision is made by

$$y' = argmax(H_{new}\beta) \tag{10}$$

where $H_{new} = g(x_{new}W + b)$ is the hidden layer output for the new sample, and y' is the predicted class label. The use of ELM in malware detection leverages its ability to quickly learn from high-dimensional feature spaces, making it well-suited for processing the complex feature sets derived from static and dynamic analysis of malware samples. The random assignment of hidden layer parameters reduces the computational burden, allowing the model to scale efficiently with large datasets. The analytical solution for the output weights ensures that the model training is both fast and robust, providing reliable detection performance in distinguishing between benign and malicious software. After the Extreme Learning Machine (ELM) phase in malware detection, the output typically consists of the predicted labels for the test dataset, along with performance metrics that evaluate the effectiveness of the model. The network structure diagram of extreme learning machine is shown in Fig. 2. Table 4 shows the output that might look like.

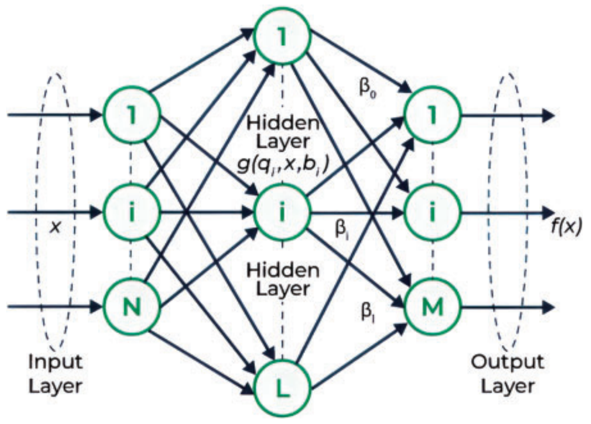


Figure 2: Network structure diagram of extreme learning machine

Table 4: Predicted labels for data 1

Sample ID	Predicted label
001	Malicious
002	Benign
003	Malicious
004	Benign
005	Malicious

4 Experiment and Results Analysis

This section discusses the experimental evaluation and result of the proposed work. The comprehensive detail of each has been presented in below sub section.

4.1 Environment for Malware Detection System Deployment

To deploy a robust malware detection system using the Extreme Learning Machine (ELM) model, a comprehensive environment is set up that integrates model deployment, real-time detection, alert and response mechanisms, and continuous learning. This environment ensures that the system remains scalable, adaptive, and capable of providing real-time protection against emerging threats.

- ***Model Deployment***

  Model deployment begins by containerizing the Extreme Learning Machine (ELM) model using Docker to ensure a consistent environment across different platforms. The Docker image is built and pushed to a container registry, and Kubernetes is used to manage the deployment and scaling of these containers on a cloud platform, such as AWS using AWS Elastic Kubernetes Service (EKS). The details are given in Table 5.

Table 5: Deployment environment

Component	Description
Docker	Containerizes the ELM model and its dependencies for consistent deployment.
Kubernetes	Manages the deployment, scaling, and operations of the Docker containers.
Cloud platform	AWS EKS used for scalable and reliable deployment.

- ***Real-Time Detection***

  In the real-time detection phase, Apache Kafka handles the real-time ingestion of data streams, such as network logs and system activities. Apache Flink processes these streams in real-time and applies the ELM model to detect malware. The results are accessible via RESTful APIs, allowing other systems to query the model for real-time detection. The details are given in Table 6.

- ***Alert and Response***

  The detection system is integrated with a Security Information and Event Management (SIEM) system like Splunk or the ELK Stack for aggregating, analyzing, and visualizing security data. A Security Orchestration, Automation, and Response (SOAR) platform that are Palo Alto Networks Cortex XSOAR is used to automate response actions, such as isolating infected systems or blocking malicious IPs. The details are given in Table 7.

Table 6: Real-time detection environment

Component	Description
Apache Kafka	Ingests real-time data streams for processing.
Apache Flink	Processes data streams in real-time using the ELM model.
FastAPI	Exposes the model via RESTful APIs for real-time queries.

Table 7: Alert and response environment

Component	Description
SIEM	Aggregates and visualizes detection logs and alerts.
SOAR	Automates response actions based on detection alerts.

- ***Continuous Learning and Improvement***

  Continuous learning involves setting up a pipeline to periodically retrain the ELM model using new data. Reinforcement learning techniques help in continuously improving the model based on detection feedback. Online learning algorithms ensure the model adapts in real-time as new data becomes available. The details are given in Table 8.

Table 8: Continuous learning environment

Component	Description
Data pipeline	Collects and preprocesses new data for retraining.
Reinforcement learning	Improves the model based on detection feedback.
Online learning algorithms	Incrementally updates the model with new data.

4.2 Baseline Approaches

Following baseline approaches are carefully selected to compare the proposed model results with these approaches:

- Baseline 1 [29]: The proposed methodology involves applying AutoML techniques for both static and online malware detection by optimizing hyperparameters and neural architecture search (NAS).
- Baseline 2 [30]: Introduced a novel feature extraction technique utilizing an autoencoder architecture, named DroidEncoder, specifically designed for the classification of Android malware applications.
- Baseline 3 [31]: Developed a machine learning-based approach for malware detection, termed MalwD&C, aimed at enabling the secure installation of Programmable Executable (PE) file.
- Baseline 4 [32]: Proposed a MobiPCR, which is strict, precise, and efficient mobile-oriented malware detection system.

4.3 Result

The performance of the Optimizing Malware Detection through Real-Time and Adaptive Security (OMD-RAS) model was rigorously evaluated across three distinct datasets: MWD-I, MWD-II, and MWD-III. The results demonstrated the model's effectiveness, with the accuracy rates ranging from 95.20% to

96.32%. MWD-II exhibited the highest accuracy at 96.32%, while MWD-I and MWD-III followed closely with 95.64% and 95.20%, respectively. In terms of precision, the model consistently maintained high levels, with MWD-II achieving the top precision of 94.88%, and MWD-I and MWD-III achieving 93.12% and 94.08%, respectively. The recall values also reflected the model's robust capability in identifying malware instances, with MWD-II again leading with a recall of 95.66%, followed by MWD-III at 94.67% and MWD-I at 93.97%. These results indicate that OMD-RAS not only delivers high accuracy in malware detection but also excels in minimizing false positives and effectively identifying malware, demonstrating its adaptability and reliability in addressing the challenges posed by evolving malware threats. Fig. 3 shows experimental results on MWD-I, MWD-II and MWD-III.

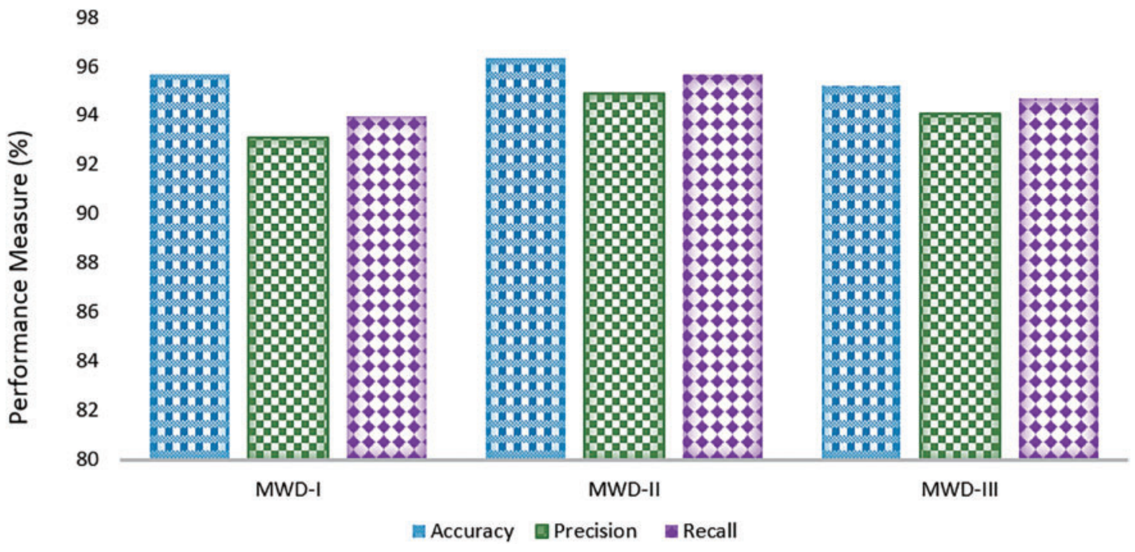
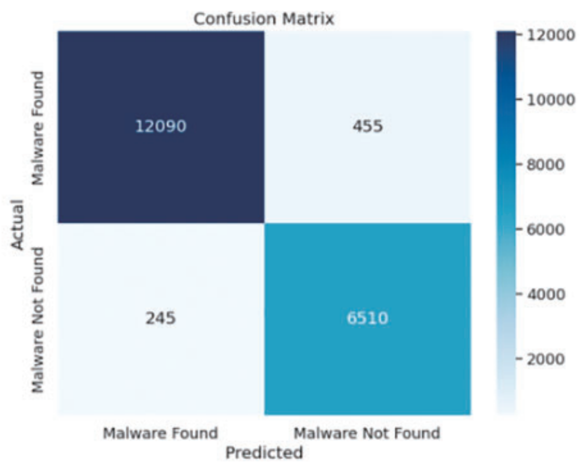


Figure 3: Experimental results on MWD-I, MWD-II and MWD-III

In another experiment, confusion matrices were employed to evaluate the effectiveness of the proposed approach in distinguishing between normal and fraud instances, as illustrated in Fig. 4. The model demonstrated a notable average accuracy of 95.12% across all datasets, indicating a high true positive rate while maintaining a low false positive rate across various classification thresholds.

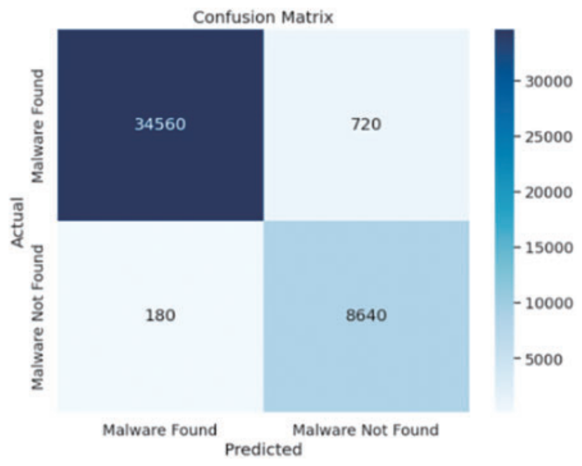
Results indicate a very outstanding improvement over all baselines across all key performance metrics by the proposed model. It recorded an accuracy of 95.72%, way above Baseline 1 at 87.23%, Baseline 2 at 84.15%, and Baseline 3 at 92.49%. Improvements in accuracy state that the proposed model is reliable and consistent in correctly identifying malware. It can also be noticed that the proposed model results in very good precision of 94.02%, outperforming Baseline 1 with its precision level at 85.18%, Baseline 2 at 83.62%, and Baseline 3 at 91.05%. This probably means that it is more effective in reducing false positives and hence accurately detects actual malware examples. It also brings out the recall metric, which measures the ability of the model to detect actual malware cases, at 94.7%. This outperforms Baseline 1 at 86.21%, Baseline 2 at 84.2%, and Baseline 3 at 91.94%. This improved recall may mean the proposed model is better at detecting malware threats without missing possible risks. Fig. 5 shows comparison with baseline approaches in terms of Accuracy, Precision and Recall.

(a) Confusion Matrix of MWD-I



(b) Confusion Matrix of

MWD-II



(c) Confusion Matrix of

MWD-III

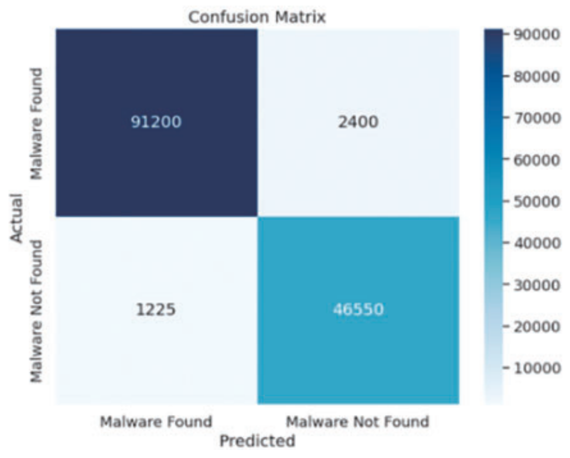


Figure 4: Confusion matrix of MWD-I, MWD-II and MWD-III

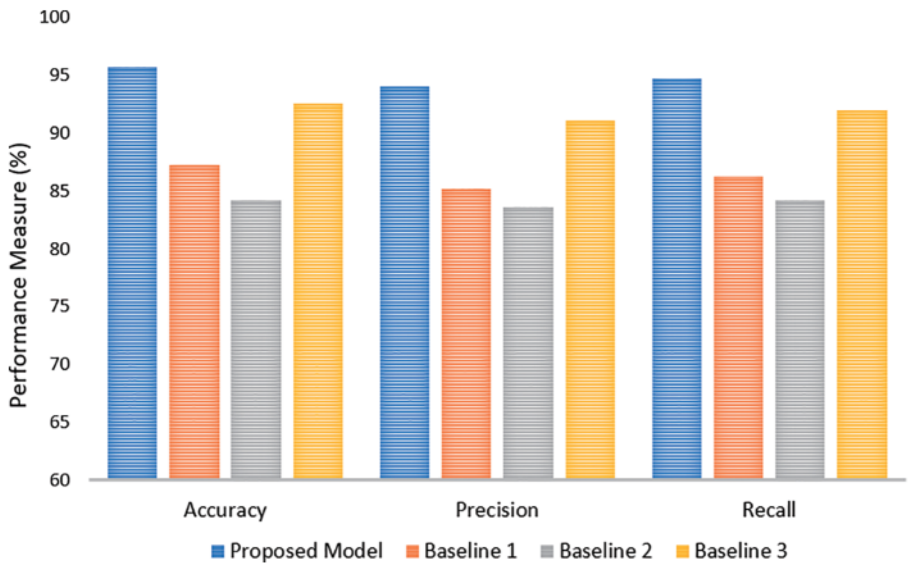


Figure 5: Comparison with baseline approaches in terms of Accuracy, Precision and Recall

The proposed model has the following advantages over baseline models: First, on accuracy, hence it gives more reliable malware detection with lower chances of false negatives, thus improving the overall security of the system. Second, improved precision avoids a large number of false positives, which is crucial for operational efficiency. Thirdly, the model's improved recall rate makes it more potent at identifying actual malware threats, thus improving general security posture. Last but not least, this model provides a good balance among high accuracy, precision, and recall, which makes it an all-rounded solution for malware detection in that it articulates and distances a wide variety of malware threats with strong performance. These benefits underscore the proposed model as both robust and reliable for modern solutions in the evolving challenges addressed by sophisticated malware attacks.

The log loss metric quantifies the accuracy of the model's predicted probabilities. It happens to be an integral component of evaluating the effectiveness of classification models, since in most cases it is related to the detection of malware, its construction based upon lower values of log loss, producing therefore more accurate probability predictions and implicitly reducing classification errors. In Table 9, the log loss value of the proposed OMD-RAS model is compared with three baseline models for three respective datasets: MWD-I, MWD-II, and MWD-III. The proposed model, OMD-RAS, had a log loss of 0.340 for the MWD-I dataset, far contrasted and lower as compared to Baseline 1, log loss: 0.560; Baseline 2: 0.430; and Baseline 3: 0.390. Here, the major reduction in log loss reveals the effectiveness of the proposed OMD-RAS model in furnishing accurate probability estimates and, therefore, reducing classification errors. The trend in better performance from the just-mentioned performance of the MWD-II dataset was exhibited in the developed OMD-RAS model that culminated in a log loss of 0.375 when compared to Baseline 1 (0.570), Baseline 2 (0.435), and Baseline 3 (0.400). The most impressive improvement can be seen in the MWD-III dataset, inferred as the lowest log loss, where OMD-RAS gives a value of 0.321, while Baseline 1 is 0.555, Baseline 2 is 0.425, and Baseline 3 is 0.395. Without much doubt, these results indicate that the OMD-RAS model is really better characterized by highly precise and reliable probability estimations on all datasets, representing preferable, advanced, and more exact tools for malware detection than baseline ones.

Table 9: Log loss comparison of proposed model with baselines

Dataset	Baseline 1	Baseline 2	Baseline 2	ETD-BTM (Proposed model)
MWD-I	0.560	0.430	0.390	0.340
MWD-II	0.570	0.435	0.400	0.375
MWD-III	0.555	0.425	0.395	0.321

5 Conclusion and Future Work

The OMD-RAS framework represents a significant advancement in malware detection, addressing critical limitations of traditional methods. By integrating comprehensive preprocessing, feature extraction through both static and dynamic analysis, and leveraging the Extreme Learning Machine (ELM) for real-time detection, OMD-RAS enhances detection accuracy while reducing false positives. Its continuous learning capability allows for adaptation to emerging threats, ensuring high performance across various datasets. With an accuracy of 96.23%, OMD-RAS demonstrates its superiority in detecting and mitigating malware threats, providing a robust and scalable solution for modern cybersecurity challenges. For future work, further enhancements will focus on improving the interpretability of OMD-RAS by incorporating approaches from Explainable AI (XAI), allowing security analysts to better understand and trust model predictions. Additionally, expanding the framework by integrating anomaly detection methods, such as GAN-based models, could strengthen its resilience against evolving malware tactics, particularly for detecting zero-day attacks. Another key direction will involve optimizing the continuous learning mechanisms to enable faster and more efficient adaptation to emerging threats while minimizing computational overhead. Finally, efforts will be made to evaluate OMD-RAS in more diverse and complex environments, including real-world, large-scale cybersecurity infrastructures, ensuring its practical deployment and effectiveness in industry settings.

Acknowledgement: This research was supported by a grant from the Center of Excellence in Information Assurance (CoEIA), King Saud University (KSU).

Funding Statement: This research work is funded by a grant from the Center of Excellence in Information Assurance (CoEIA), King Saud University (KSU).

Author Contributions: The authors confrm their contribution to the paper as follows: study conception and design: Farah Mohammad; data collection: Saad Al-Ahmadi; analysis and interpretation of results: Jalal Al-Muhtadi; draft manuscript preparation: Farah Mohammad, Saad Al-Ahmadi and Jalal Al-Muhtadi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All data generated or analyzed during this study are included in this published article.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Ahmad I, Bakar AA, Jan R, Yussof S. Dynamic behaviors of a modified computer virus model: insights into parameters and network attributes. Alex Eng J. 2024;103(7):266–77. doi:10.1016/j.aej.2024.06.009.

2. Koch L, Begoli E. Adversarial binaries: AI-guided instrumentation methods for malware detection evasion. ACM Comput Surv. 2025;57(5):1–36. doi:10.1145/3706573.
3. Riggs H, Tufail S, Parvez I, Tariq M, Khan MA, Amir A, et al. Impact, vulnerabilities, and mitigation strategies for cyber-secure critical infrastructure. Sensors. 2023;23(8):4060. doi:10.3390/s23084060.
4. Johny JA, Asmitha KA, Vinod P, Radhamani G. Deep learning fusion for effective malware detection: leveraging visual features. Clust Comput. 2025;28(2):135. doi:10.1007/s10586-024-04723-w.
5. Sangher KS, Singh A, Pandey HM. Signature based ransomware detection based on optimizations approaches using RandomClassifier and CNN algorithms. Int J Syst Assur Eng Manag. 2024;15(5):1687–703. doi:10.1007/s13198-023-02017-9.
6. Deldar F, Abadi M. Deep learning for zero-day malware detection and classification: a survey. ACM Comput Surv. 2023;56(2):1–37. doi:10.1145/3605775.
7. Feng J, Shen L, Chen Z, Lei Y, Li H. HGDetector: a hybrid Android malware detection method using network traffic and function call graph. Alex Eng J. 2025;114(1):30–45. doi:10.1016/j.aej.2024.11.068.
8. Djenna A, Bouridane A, Rubab S, Marou IM. Artificial intelligence-based malware detection, analysis, and mitigation. Symmetry. 2023;15(3):677. doi:10.3390/sym15030677.
9. Wang F. Design of computer network security intrusion prevention strategy and evaluation algorithm analysis technology. Procedia Comput Sci. 2023;228:1270–6. doi:10.1016/j.procs.2023.11.093.
10. Sharma YK, Tomar DS, Pateriya RK, Bhandari S. MOSDroid: Obfuscation-resilient Android malware detection using multisets of encoded opcode sequences. Comput Secur. 2025;5(1):104379. doi:10.1016/j.cose.2025.104379.
11. Liu Y, Li W, Dong X, Ren Z. Resilient formation tracking for networked swarm systems under malicious data deception attacks. Int J Robust Nonlinear Control. 2025;35(6):2043–52. doi:10.1002/rnc.7777.
12. Lyu M, Gharakheili HH, Sivaraman V. A survey on enterprise network security: asset behavioral monitoring and distributed attack detection. IEEE Access. 2024;12(1):89363–89. doi:10.1109/ACCESS.2024.3419068.
13. Yan S, Wang S, Duan Y, Hong H, Lee K, Kim D, et al. An LLM-assisted easy-to-trigger backdoor attack on code completion models: injecting disguised vulnerabilities against strong detection. arXiv:2406.06822. 2024.
14. Kara I. Fileless malware threats: recent advances, analysis approach through memory forensics and research challenges. Expert Syst Appl. 2023;214(1):119133. doi:10.1016/j.eswa.2022.119133.
15. Krishna B, Krishnan S, Sebastian MP. Understanding the process of building institutional trust among digital payment users through national cybersecurity commitment trustworthiness cues: a critical realist perspective. Inf Technol People. 2023;81(2):9435.
16. Shaukat K, Luo S, Varadharajan V. A novel machine learning approach for detecting first-time-appeared malware. Eng Appl Artif Intell. 2024;131(1):107801. doi:10.1016/j.engappai.2023.107801.
17. Puneeth S, Lal S, Singh MP, Raghavendra BS. Rmdnet-deep learning paradigms for effective malware detection and classification. IEEE Access. 2024;81(1):98345. doi:10.1109/ACCESS.2024.3403458.
18. Liu L, Wang BS, Yu B, Zhong QX. Automatic malware classification and new malware detection using machine learning. Front Inf Technol Electron Eng. 2017;18(9):1336–47. doi:10.1631/FITEE.1601325.
19. Rigaki M, Garcia S. The power of meme: adversarial malware creation with model-based reinforcement learning. In: 28th European Symposium on Research in Computer Security; 2023 Sep 25–29; The Hague, The Netherlands. p. 44–64.
20. Aamir M, Iqbal MW, Nosheen M, Ashraf MU, Shaf A, Almarhabi KA, et al. AMDDLmodel: android smartphones malware detection using deep learning model. PLoS One. 2024;19(1):e0296722. doi:10.1371/journal.pone.0296722.
21. Fernando DW, Komninos N. FeSAD ransomware detection framework with machine learning using adaption to concept drift. Comput Secur. 2024;137(1):103629. doi:10.1016/j.cose.2023.103629.
22. Hasan R, Biswas B, Samiun M, Saleh MA, Prabha M, Akter J, et al. Enhancing malware detection with feature selection and scaling techniques using machine learning models. Sci Rep. 2025;15(1):9122. doi:10.1038/s41598-025-93447-x.
23. Nasser AR, Hasan AM, Humaidi AJ. DL-AMDet: deep learning-based malware detector for Android. Intell Syst Appl. 2024;21(1):200318. doi:10.1016/j.iswa.2023.200318.

24. Thakur P, Kansal V, Rishiwal V. Hybrid deep learning approach based on LSTM and CNN for malware detection. Wirel Pers Commun. 2024;136(3):1879–901. doi:10.1007/s11277-024-11366-y.
25. Alomari ES, Nuiaa RR, Alyasseri ZAA, Mohammed HJ, Sani NS, Esa MI, et al. Malware detection using deep learning and correlation-based feature selection. Symmetry. 2023;15(1):123. doi:10.3390/sym15010123.
26. Bharadiya JP. A tutorial on principal component analysis for dimensionality reduction in machine learning. Int J Innov Sci Res Technol. 2023;8(5):2028–32. doi:10.5281/zenodo.8002436.
27. Qu L, Pei Y. A comprehensive review on discriminant analysis for addressing challenges of class-level limitations, small sample size, and robustness. Processes. 2024;12(7):1382. doi:10.3390/pr12071382.
28. Awad M, Fraihat S. Recursive feature elimination with cross-validation with decision tree: feature selection method for machine learning-based intrusion detection systems. J Sens Actuator Netw. 2023;12(5):67. doi:10.3390/jsan12050067.
29. Brown A, Gupta M, Abdelsalam M. Automated machine learning for deep learning-based malware detection. Comput Secur. 2024;137(2):103582. doi:10.1016/j.cose.2023.103582.
30. Bakır H, Bakır R. DroidEncoder: malware detection using auto-encoder based feature extractor and machine learning algorithms. Comput Electr Eng. 2023;110(1):108804. doi:10.1016/j.compeleceng.2023.108804.
31. Buriro A, Buriro AB, Ahmad T, Buriro S, Ullah S. MalwD&C: a quick and accurate machine learning-based approach for malware detection and categorization. Appl Sci. 2023;13(4):2508. doi:10.3390/app13042508.
32. Liu C, Lu J, Feng W, Du E, Di L, Song Z. MOBIPCR: efficient, accurate, and strict ML-based mobile malware detection. Future Gener Comput Syst. 2023;144(1):140–50. doi:10.1016/j.future.2023.02.014.