Al-Rafidain Journal of Engineering Sciences

Journal homepage https://rjes.iq/index.php/rjes

ISSN 3005-3153 (Online)

PPO-Based Neural Architecture Search for Efficient and Hardware-Aware Windows Malware Detection in Cybersecurity

Muthana S. Mahdi

Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq

ARTICLE INFO

Article history:
Received 4 April 2026
Revised 4 April 2026,
Accepted 27 April 2026,
Available online 29 April 2026

Keywords:
Intelligent Agent
Windows Malware Detection
Reinforcement Learning
Neural Architecture Search
Proximal Policy Optimization

ABSTRACT

The rapid growth of malicious software targeting Windows operating systems has created significant challenges for modern cybersecurity systems. Traditional malware detection approaches often rely on manually designed models or signature-based techniques, which struggle to identify newly emerging and highly obfuscated malware variants. Although recent deep learning methods have improved detection accuracy, many of these models depend on fixed neural architectures and require extensive manual tuning, limiting their adaptability and efficiency. To address these challenges, this study proposes a reinforcement learning–driven Intelligent Agent framework for automated neural architecture search in Windows malware detection. The proposed system employs a reinforcement learning agent based on Proximal Policy Optimization to explore a simplified architecture search space and automatically construct an effective deep learning model. The approach was evaluated using the EMBER Dataset, a widely used benchmark dataset for malware classification. Experimental results demonstrate that the architecture discovered by the proposed Intelligent Agent achieves strong detection performance, reaching an accuracy of 97.3 while also outperforming several baseline approaches in terms of Precision, Recall, F1-Score, and AUC metrics. These findings indicate that integrating reinforcement learning with neural architecture search can significantly enhance malware detection performance while reducing the need for manual model design. In addition to achieving high detection accuracy, the proposed approach considers computational efficiency by enabling the discovery of hardware-aware neural architectures suitable for real-world deployment. The proposed framework highlights the potential of intelligent agent–based systems in developing adaptive and efficient cybersecurity solutions capable of addressing evolving malware threats.

1. Introduction

The high rate of digital technologies development and the increase in the number of networked systems have enhanced the quantity and variety of malicious programs that attack contemporary computing systems at a large scale [1]. Specifically, Windows-based systems have remained among the most commonly attacked operating systems because of their wide usage in both personal computers, enterprise infrastructures, and cloud systems [2]. Consequently, malicious Windows executable detection has become the pressing issue of cybersecurity studies. In the current scenario, malware tends to utilize advanced tools to bypass the conventional detection tools, code obfuscation, polymorphism, and anti-analysis to improve the malware's evasion. Such features complicate the process of detecting malware and demand more

Corresponding author E-mail address: muthanasalih@uomustansiriyah.edu.iq
https://doi.org/10.61268/p3hq7v10

sophisticated and dynamic analytical approaches [3].

Traditional methods of malware detection have been based on signature-based methods where a database of known malicious code patterns is present and used to detect threats that have been previously encountered [4]. Even though signature-based systems perform well when used to detect known malware families, they do not work well at recognizing newly emerging malware variants and zero-day attacks [5]. As a result, the cybersecurity profession has been slowly moving towards machine learning and deep learning algorithms that can automatically discover discriminative behaviors on large volumes of malware files [6].

Malware detection by machine learning processes features that have been extracted from executable files, either through a static or dynamic analysis methodology [7]. Without running the program, a static analysis analyzes file headers, libraries, and collections of bytes and metadata, which makes it efficient when used on a large scale [8]. By comparison, dynamic analysis monitors runtime behavior, such as system calls and API interactions, and offers more detailed information, but at the cost of more specialized environments and increased computational cost. Hence, numerous current works use deep learning models to learn meaningful representations with the use of static features without sacrificing the real-world detection performance [9].

Although there has been an advancement in learning-based malware detection, there are still a number of challenges. Most of the current literature makes use of manually constructed neural network architecture designs in which the researcher constructs the model by trial and error and by experiment. It is highly skillful and time-consuming, and the resulting architecture might not be the most efficient for a particular dataset, particularly as malware datasets are becoming larger and more complicated [10]. Moreover, other contemporary methods enhance the accuracy of detection with the application of ensemble learning, attention, or other transformer-based models; however, the methods are associated with high computational costs and require large-scale training, which may restrict their applicability to real-world cybersecurity systems [11].

It is based on these issues that this paper presents an agent-based intelligent neural architecture search model to detect Windows malware. The method adopts a reinforcement learning agent with the Proximal Policy Optimization (PPO) algorithm, which can simply search through a simplified architecture space and find the effective structures of neural models. Reinforcement learning allows the agent to engage with the search space, examine candidate architectures, and, over time, refine its decisions on the basis of performance feedback. The agent, through this process, gets to learn to choose model configurations that can get good malware classification performance, while maintaining computational efficiency and considering hardware-related constraints.

The primary goal of this study is to come up with an automated framework that can be able to find appropriate deep learning structures that will be able to detect malware with a minimum number of manual efforts. The intelligent agent also directs the search for architecture and builds neural networks that find significant patterns in malware feature representations, instead of using traditional trial-and-error modeling of the design of models.

Three main contributions are made in this work. To begin with, it proposes an intelligent agent, which is based on reinforcement learning and is automatically trained to search neural architectures adapted to Windows malware detection tasks. Second, the framework has a simplified architecture search space that makes computational complexity simpler and flexible enough to effectively discover models. Third, the model that is obtained exhibits high detection abilities but does not have the unwarranted sophistication that is commonly attributed to manually developed deep learning systems. On the whole, the combination of intelligent agents

with automated architecture search helps to create more adaptive and effective malware detection systems that should respond to the changing cybersecurity threats.

2. Related Work

Research on Windows malware detection has progressed considerably during the past few years, with different machine learning and deep learning techniques proposed to improve the identification of malicious executable files. Older methods were mostly based on a static analysis of features, whereas newer methods have tried to add behavioral analysis, deep neural networks, and higher-order optimization methods. In spite of these developments, a number of current methods continue to have limitations associated with computational complexity, lack of adaptability, or reliance on pre-built model structures.

Huang et al. suggested a deep learning-based method of identifying Windows malware by analyzing their static features based on executable files [12]. In their research, they showed that neural network models are capable of detecting malicious patterns in PE files. The primary advantage of this work is that it can be used to realize high detection accuracy on the basis of the static analysis, which is computationally efficient in comparison with dynamic techniques. Nevertheless, the approach is based on a predefined neural structure that is not adaptable to varying feature distributions and changing malware trends.

Later, the method of detection proposed by Amer and Zelinka was introduced on the basis of API call sequences analysis [13]. It is based on dynamic behavioral information and, in this way, it can be utilized to enhance the detection of malware families that have not been observed before. Whereas the behavioral sequence methodology can improve detection power, the methodology assumes the execution of samples under controlled conditions, which imposes even greater computational demands and prevents scalability to large-scale analysis because of the methodological constraint.

Catak et al. created a framework of behavioral analysis that keeps track of Windows API call sequences to identify malware [14]. The technique is aimed at recording runtime behaviors that define malicious behaviors. Although this method is more effective at detecting new threats, it is susceptible to advanced techniques of obfuscation and needs complex pipelines to extract features.

Wang et al. suggested a hybrid framework of malware detection with the combination of both static and dynamic analysis in 2021 [15]. An advantage of this approach is its combination of several sources of features, making it more robust in detection. The model, however, has been primarily tested on malware of IoT, and its generalizability to the Windows environment is limited. Azeez et al. proposed an ensemble learning model involving the integration of several neural network models in detecting Windows PE malware [16]. Even though the ensemble approach enhances the precision, multiple models markedly raise the computation expense and training time.

Diener et al. considered the application of the memory analysis data as a malware detection tool in the context of a big-data framework [17]. The strategy has the advantage of elaborate runtime data acquired through system memory. However, the need to have a huge data infrastructure limits its practical implementation. Ravi and Alazab proposed an attention-based convolutional neural network to improve feature presentation in the classification of malware [18]. Compared to the model, the attention mechanism increases the computational complexity of training, although its classification performance is better.

Later, Bayesian hyperparameter optimization was used to improve malware detection models by another study by Algorain and Clark [19]. The method increases the efficiency of parameter tuning but is still reliant on the architecture that is set up in advance of the neural network. In the same way, Kudrekar and Rani defined a Q-learning-based strategy of malware detection through classification [20]. Despite the fact that reinforcement

learning facilitates adaptive decision making, the framework proposed is quite complex and difficult to implement and tune in the real-world setting. Kim et al. studied methods of anti-analysis applied by real-life malware samples [21]. Their work enlightens us on the malware evasion techniques; however, it does not suggest a model for detecting malware.

Kocak et al. [22] compared a number of machine learning models with benchmark datasets to detect Windows PE malware. Their findings indicate that conventional machine learning algorithms may perform well. Nevertheless, these models are usually based on features that are designed manually and have no capability to learn hierarchy representations. In a similar manner, Maulani et al. explored the malware detection methods with the aid of artificial intelligence methods [23]. In spite of the effectiveness of AI-based methods mentioned in the study, the research was chiefly concerned with the analysis of the techniques at rest, and the design of automated architecture was not even discussed. Moreover, the researchers, Du et al., suggested a deep neural network that is able to model the behavior patterns that are process-aware [24]. Although effective, the model is costly in both the extent of training data it needs and the changes in runtime behaviors.

Later Syeda and Asghar developed a dynamic malware classification method relying on the classification of API behaviors in Windows PE files [25]. This approach is good at real-time detection but remains a challenge to sophisticated methods of obfuscation. Besides this, Zada et al. performed a comparative analysis of various supervised learning algorithms in the detection of malware [26]. Although the study gives useful benchmarking findings, it does not suggest a new detection architecture. On the same note, Gururaja et al. showed the effectiveness of the ensemble models in malware detection in Windows environments [27]. The strategy, however, might not be able to generalize to newer versions of operating systems. Imran et al. also published another recent work that investigated adversarial vulnerabilities of machine learning-based malware detectors [28]. Their results emphasize the role of robustness, but the research is primarily based on adversarial analysis instead of suggesting better detection structures. Lastly, Khan and Nauman came up with a multi-head transformer architecture for detecting malicious code in Windows executables [29]. The model enhances better representation of complicated correlations among features. However, transformer-based architectures are computationally costly and might not be applicable in resource-constrained environments.

Although major strides have been made in the domain of Windows malware detection, there are still a few weaknesses that can be observed in the current methods. Several conventional machine learning techniques use manually designed features and fixed model structures and, therefore, are not capable of adapting to the ever-changing nature of contemporary malware. Similarly, a number of deep learning models enhance the accuracy of detection by using complex models like attention, ensemble, or transformer models, but those methods tend to introduce large overhead and demand a huge amount of training.

Furthermore, the majority of current literature relies on handcrafted neural architectures, in which a successful model structure is determined by a significant amount of trial-and-error and manual manipulation. Even though methods like Bayesian optimization can be used to optimize hyperparameters, they fail to deal with the challenge of automatically finding an appropriate neural architecture.

This paper has proposed an intelligent agent based on reinforcement learning to address these drawbacks by utilizing a search space of neural architecture automatically. The proposed approach is going to use a PPO-based smart agent to find an effective model structure that can balance between the accuracy of detection and the efficiency of the computation, and minimize manual design tasks. Table 1 shows a summary of the related works with their strengths and weaknesses.

Table 1: Summary of related works

Ref.	Approach	Strength Points	Weak Points
[12]	Deep learning using static PE features	High detection accuracy with static analysis	Fixed architecture and limited adaptability
[13]	API call sequence analysis	Captures behavioral malware patterns	Requires dynamic execution environment
[14]	Behavioral API sequence modeling	Detects emerging malware behaviors	Sensitive to obfuscation techniques
[15]	Hybrid static–dynamic analysis	Robust detection using multiple feature sources	Limited evaluation of Windows malware
[16]	Ensemble neural networks	Improved classification accuracy	High computational cost
[17]	Memory analysis with big-data framework	Rich runtime information	Requires large infrastructure
[18]	Attention-based CNN	Enhanced feature representation	High training complexity
[19]	Bayesian hyperparameter optimization	Efficient parameter tuning	Depends on predefined architectures
[20]	Reinforcement learning classification	Adaptive decision making	Complex implementation
[21]	Analysis of anti-analysis techniques in malware	Provides insights into malware evasion strategies	Does not propose a detection model
[22]	Machine learning models for Windows PE malware detection	Strong baseline performance	Limited representation learning
[23]	AI-based malware detection techniques	Demonstrates AI potential	Focus mainly on static analysis
[24]	Process-aware deep neural networks	Captures runtime behavior patterns	Requires large training datasets
[25]	Dynamic API behavior classification	Improves real-time detection	Weak against advanced obfuscation
[26]	Supervised classifier comparison	Comprehensive performance evaluation	Does not propose a new detection model
[27]	Ensemble malware detection	High detection performance	Limited generalization
[28]	Adversarial vulnerability analysis	Highlights robustness issues	Not focused on detection architecture
[29]	Transformer-based malware detection	Captures complex feature relations	High computational complexity

3. Proposed Method

In this section, the paper outlines the method that would be used to build a working neural network model of Windows malware detection. Rather than having to manually design the network architecture by trial and error, the proposed approach uses an intelligent agent based on reinforcement learning to automatically search the space of a neural architecture. The Proximal Policy Optimization algorithm is used to train the agent by updating its decisions iteratively with candidate architecture evaluation and using the observed performance for detection using the algorithm.

The primary aim of the approach is to make the process of designing an architecture easier and yet arrive at a model that is able to achieve a high malware detection performance. The intelligent agent searches a small and well-designed set of search space which contains some of the possible combinations of convolutional layers, recurrent layers, and fully connected layers. The search process entails spending some time building candidate architectures, briefly training them using the training data, and testing their performance using validation data, and adjusting their policy in such a way that improved architectures have a

higher probability of being chosen in the subsequent iterations.

The proposed methodology operates with a benchmark dataset and a simplified architecture search space to maintain the methodology as simple and computationally efficient as possible. Once the search process has finished, the most optimal architecture that was identified by the PPO agent is chosen and thoroughly trained. The functionality of this last model is then measured, and the results are reported in Section 4.

3.1 Dataset Collection and Preparation

To simplify the experimental design in order to guarantee reproducibility, the study utilizes the EMBER Dataset [30] as the data source to construct and test the model proposed.

The EMBER dataset is employed as a common benchmark in malware detection studies and has a huge set of labeled Windows executable files modeled as static features obtained on Portable Executable files. Such attributes are metadata, header attributes, imported libraries, and statistics on the size of the bytes, among other structural features that one can use to differentiate malicious files and benign files.

Both malicious samples and benign samples are present in the dataset as fixed-length feature vectors. The dataset is also taken through a normal preprocessing pipeline before the neural network model is trained. To begin with, missing values are addressed, and the feature vectors are normalized to achieve consistent training behavior. The data is then split into 70% training, 15% validation, and 15% testing sets. The network parameters are learned with the help of the training subset, the validation subset is employed by the PPO agent to test the candidate architectures in the process of search, and the testing subset is kept to conduct the final testing of the chosen model.

This stage of preparation is necessary to make sure that the process of learning is regular and that the architecture search is informed by credible validation feedback.

3.2 Neural Architecture Construction Using a PPO-Based Intelligent Agent

The neural network architecture built with the help of an intelligent agent based on PPO is the main element of the offered approach. The agent does not use manual architecture design; instead, it learns how to build neural networks by searching through various combinations of layers and structural configurations within a fixed search space.

Initially, in the process, the agent will produce a candidate neural architecture by choosing a few structural elements, including convolutional depth, the number of filters per layer, kernel size, activation function, pooling strategy, recurrent layer, fully connected layer, and dropout regularization. After generating the architecture, a temporary model is generated and trained during a small number of epochs on the training portion of the EMBER dataset. Following this brief training period, the model is tested using the validation data to determine its capacity to detect malware.

The intelligent agent is provided with feedback on the validation performance. Making better architectures with regard to detection performance is rewarded more, which motivates the agent to generate similar architectures in subsequent iterations. The agent learns over time, through repeated exposure to the training environment, what architectural patterns are more useful in the malware detection task.

In this study, the search space is deliberately narrowed down to a few useful architectural options to serve the purpose of making the computationally demanding much simpler. The agent has the option of 1D convolutional layers with one, two, or three layers. The layer of each convolutional block can either have 32, 64, or 128 filters with 3 or 5 kernel size. These layers use the EMBER feature vector to learn local features between groups of features.

Following the convolutional layers, the agent can follow this optionally with a recurrent layer, which is based on a Long Short-Term Memory unit. Dependency among the feature representations that have been extracted is captured by the LSTM layer. At last, the last classification is done with either one or two fully connected layers.

This reduces the size of the search space, enabling the agent to effectively investigate the various model configurations without incurring too much computational cost. Concurrently, it does not make it overly restrictive to the point of finding effective network structures to detect malware. In an attempt to further explain how the PPO-based agent makes decisions when constructing its neural architecture, Table 2 below gives a summary of the state space and the action options the agent has at every step of the architecture generation process.

Table 2: State–Action Space Used by the PPO Agent During Architecture Search

State (Architecture Decision Stage)	Description	Available Actions
Input Stage	Define the input representation used by the model	Use the EMBER feature vector
Convolution Layer Depth	Determine the number of 1D convolutional layers used for feature extraction	1 layer, 2 layers, or 3 layers
Convolution Filters	Define the number of filters in each convolutional layer	32, 64, or 128 filters
Kernel Size	Select kernel size for convolution operations	3 or 5
Activation Function	Nonlinearity used in convolutional and dense layers	ReLU or LeakyReLU
Pooling Layer	Decide whether pooling is applied after convolution layers	Apply pooling or Skip pooling
Pooling Type	If pooling applied, select type	Max Pooling or Average Pooling
Pooling Size	If pooling applied, select window	2 or 3
Recurrent Layer Inclusion	Whether to include a recurrent block after convolution blocks	Add LSTM layer or skip
LSTM Configuration	If LSTM added, select hidden size	64 or 128 units
Dense Layer Depth	Number of fully connected (dense) layers after recurrent block	1 layer or 2 layers
Dense Layer Size	Define number of neurons in the dense layer	128 or 256 neurons
Dropout Inclusion	Whether to apply dropout after dense layer(s)	Dropout or No Dropout
Dropout Rate	If dropout applied, select rate	0.2, 0.4, or 0.5
Output Layer	Final classifier specification	SoftMax layer for binary classification

After several iterations of architecture exploration and evaluation, the PPO agent identifies the configuration that consistently achieves the highest validation performance. This architecture is finally chosen as the final one and is once more trained with the entire training data. Section 4 gives the results of the final architecture and discusses them.

3.3 Final Selected Network Architecture

In the end, the search process of architecture was completed, and the PPO agent found a compact neural network architecture, which presented steady validation and efficient training behavior. The architecture chosen is two 1D convolutional layers, a Long Short-Term Memory layer, and a fully connected layer to do the final classification. The general layout of the chosen architecture is shown in Figure 1, which provides the final model that the intelligent agent found at the end of the architecture search process.

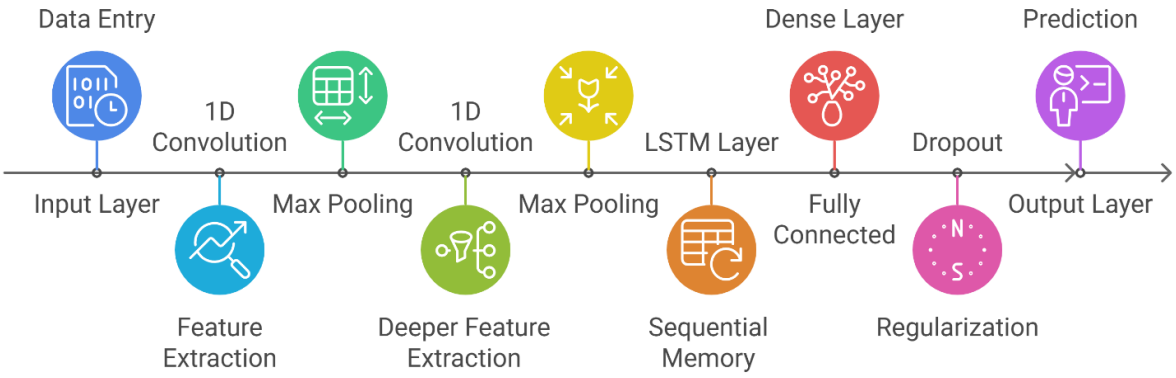

Figure 1. Architecture of the final model discovered by the intelligent agent.

The initial convolutional layer has 64 3-kernel-sized filters and a ReLU activation of the feature to identify low-level feature interactions in the EMBER feature representation. This is followed by a max pooling operation in order to decrease the size of the extracted feature maps.

The second convolutional layer has 128 filters whose kernel size and activation function are the same. This layer allows the network to acquire more complex feature patterns that are linked to malicious actions. This convolution is followed by another layer of max pooling aimed at further minimizing the dimensions of the features.

The resultant output features of the convolutional layers are subsequently fed to the LSTM layer in the form of a sequence. This modification enables the LSTM component to provide the modeling of relationships among feature groupings.

The LSTM layer has 128 hidden units, and its duty is to extract dependencies between the representations of the extracted features. Despite the LSTM layer being static, as the EMBER feature operates, the feature subsets are able to form structural relationships, and this enhances the overall detection capacity of the model.

The final part of the network consists of a fully connected layer. The dense layer has 256 neurons and uses the ReLU activation function to smooth the representation that has been learned. This is followed by a dropout mechanism to minimize overfitting when training. The last layer involves the use of the SoftMax activation function to generate a likelihood that a given executable file is malicious or benign. The fine structure of the chosen final model is demonstrated in Table 3. The hyperparameter values are indicated in Table 4.

Table 3: Detailed Architecture of the Final Selected Model

Layer No.	Layer Type	Configuration	Description
1	Input Layer	EMBER feature vector	Fixed-length feature representation extracted from PE files
2	1D Convolution	64 filters, kernel size = 3, ReLU activation	Extracts local feature relationships
3	Max Pooling	Pool size = 2	Reduces feature dimensionality
4	1D Convolution	128 filters, kernel size = 3, ReLU activation	Learns higher-level feature patterns
5	Max Pooling	Pool size = 2	Further dimensionality reduction
6	LSTM Layer	128 hidden units	Captures dependencies between feature groups
7	Fully Connected	256 neurons, ReLU activation	Refines learned representation
8	Dropout	Rate = 0.4	Reduces overfitting
9	Output Layer	SoftMax (2 neurons)	Binary classification (malware / benign)

Table 4: Hyperparameter Values Used

Hyperparameter	Value
Optimizer	Adam
Initial Learning Rate	0.001
Batch Size	64
Training Epochs	40
Dropout Rate	0.4
PPO Learning Rate	0.0003
PPO Clip Range	0.2

This architecture was chosen, as it received the optimal combination of detection accuracy/computational efficiency in the search process. This was made possible by the use of convolutional feature extraction and sequential modeling, which enabled the model to capture both local and global interactions among the features within the EMBER dataset. This architecture has specific performance outcomes, which are detailed in the next section.

3.4 Hardware-Aware Reward Design

To ensure that the proposed approach is suitable for real-world deployment, a hardware-aware reward mechanism was integrated into the reinforcement learning process. In practical cybersecurity systems, malware detection models must operate under constraints such as limited memory, computational capacity, and real-time processing requirements.

In this work, the PPO-based intelligent agent was trained using a reward function that explicitly combines detection performance with computational efficiency. The reward is defined as:

$$\text{Reward} = \text{Accuracy} - \alpha \times \text{Computational\_Cost} - \beta \times \text{Model\_Size}$$

where:

- Accuracy: Validation accuracy achieved by the candidate neural architecture.
- Model_Size: Total number of trainable parameters in the neural network.
- Computational_Cost: A relative computational complexity score defined as:

  $$\text{Computational\_Cost} = \text{Number\_of\_Layers} + \text{Model\_Size}$$

- where Number_of_Layers represents the total depth of the network.
- α: Weighting coefficient controlling the impact of computational cost on the reward.
- β: Weighting coefficient controlling the impact of model size on the reward.

In this study, the coefficients were set to $\alpha = 0.01$ and $\beta = 0.001$ to maintain detection accuracy as the primary objective while penalizing excessive model complexity.

This reward formulation guides the intelligent agent to prioritize neural architectures that achieve high detection accuracy while maintaining low computational complexity and compact model size. As a result, the generated architectures are both effective in malware detection and efficient for deployment in resource-constrained environments.

By directly incorporating computational efficiency into the reward function, the proposed framework performs hardware-aware neural architecture design as an integral part of the optimization process.

4. Results and Analysis

The section provides an experimental analysis of the suggested PPO-based neural architecture search model in Windows malware detection. The trials were performed with the use of the EMBER Dataset, which is generally regarded as a standard dataset to test machine learning models in malware detection. The evaluation aims at determining the effectiveness of the proposed architecture that has been found by the reinforcement learning agent and comparing its performance with various related methods that have been presented in the literature.

4.1 Experimental Setup

The experiments were all done in the Python programming language and the deep learning framework PyTorch. The neural architecture search space intelligent agent implemented based on the PPO algorithm was done using the Stable-Baselines3 library. Common scientific computation libraries such as NumPy and scikit-learn were used to process the data and provide normality of features.

The training experiments were executed on a workstation equipped with an Intel Core i7-10700K processor, 32 GB RAM, and an NVIDIA RTX 3060 graphics card. GPU acceleration was utilized to speed up the training process of both the candidate architectures generated during the search phase and the final selected model. The neural network model was trained using the Adam optimizer with an initial learning rate of 0.001 and a batch size of 64. The training process was conducted for 40 epochs, while early validation feedback was used by the PPO agent to guide the architecture search procedure.

4.2 Performance Evaluation

The results of the chosen architecture were measured across a variety of popular classification metrics, such as accuracy, precision, recall, F1-score, and the Area Under the ROC Curve (AUC) with equations (1-5) [31].

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

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

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

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}} \quad (4)$$

$$AUC = \sum_{i=1}^{n-1} (FPR_{i+1} - FPR_i) \times \frac{TPR_{i+1}+TPR_i}{2} \quad (5)$$

TP denotes the true positive response, TN denotes the true negative response, FP denotes the false positive response, FN denotes the false negative response, TPR denotes the true positive rate, and FPR denotes the false positive rate.

These measures give a clear picture of the detection potential of the proposed model, especially when it comes to malware classification, where false positives and false negatives are of great importance. Table 5 presents a summary of the Performance Metrics on the EMBER Dataset.

Table 5: Performance Metrics on EMBER Dataset

Metric	Value
Accuracy	97.3
Precision	97.1
Recall	97.5
F1-Score	97.3
AUC	98.1

As the results of Table 5 show, the proposed model shows a high percentage of detection accuracy on the EMBER dataset. The high precision value reveals that the model generates a small number of false positive detections, which is valuable to implement in malware detection systems in practice. On the same note, the recall value indicates that the model can be used to detect a good percentage of malicious samples accurately. The F1-score validates the fact that the model has a good balance between precision and recall. Moreover, the AUC is high, indicating that the classifier is capable of appropriately differentiating between the good and bad executable files at varying levels of decision. Figure 2 represents the performance metrics in a graphical manner, which, in turn, highlights the effectiveness of the model in the parameters considered.

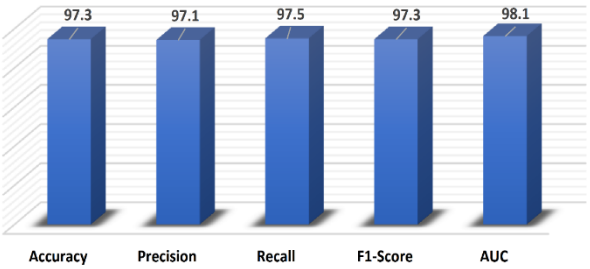

Figure 2. Performance evaluation metrics of the proposed PPO-NAS model using the EMBER dataset.

In order to conduct more analysis on the proposed model learning behavior, training, and validation error curves over epochs are shown in Figure 3. The curves also show consistent convergence and low overfitting, which proves the effectiveness of the architecture selected by the PPO agent.

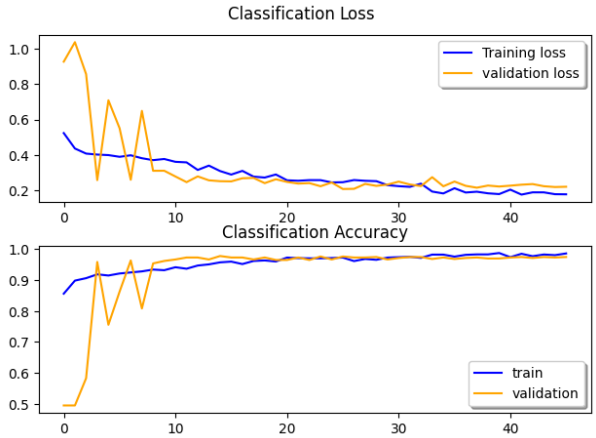

Figure 3. The training and validation loss and accuracy curves of the proposed model

4.3 Comparison with Existing Methods

In order to further evaluate the performance of the proposed approach, the performance of the model was compared to several related malware detection approaches, which are discussed in the related work section. These approaches involve traditional machine learning models as well as deep learning–based architectures that have been evaluated on the same dataset. Table 6 indicates the comparison against Existing Methods on the EMBER dataset.

Table 6: Comparison with Existing Methods

Approach	Accuracy	Precision	Recall	F1-Score
[19]	92.9	92.8	93.2	92.9
[23]	94.2	93.9	94.5	94.1
[25]	96.0	96.2	95.9	96.0
Proposed PPO-NAS Model	97.3	97.1	97.5	97.3

As Table 6 illustrates, the proposed PPO-based architecture search methodology yields improved performance over various popular malware detector models. The comparison of the proposed PPO-NAS model with the related works is also graphically demonstrated in Figure 4, which depicts that its performance is better in all the measures of evaluation.

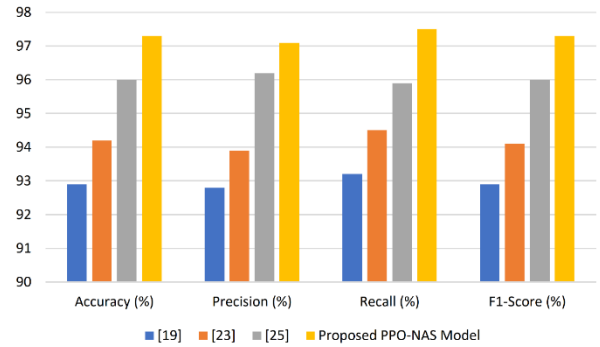

Figure 4. A graphical comparison of the proposed PPO-NAS model against related works.

Some of the baseline strategies mentioned in the literature are able to record moderate detection on the EMBER dataset. But their architectures are normally pre-configured, and they are held constant throughout the training process. This reduces their capability to actively experiment with different structural arrangements that can better represent, in the feature space, the more intricate relationships. Consequently, the performance gains that can be attained by systematic exploration of architecture are limited.

The given model is also superior in detection performance as it is able to automatically find an effective neural architecture with the assistance of reinforcement learning. The PPO agent tries various architectural setups and picks a structure that balances feature extraction ability with model complexity. Consequently, the last architecture will have better classification performance than the baseline methods.

Generally, the experimental evidence supports the hypothesis that a combination of reinforcement learning agent-based neural architecture search and deep learning can greatly improve the performance of malware detectors without resorting to a significantly large and inefficient model design.

4.4 Computational Efficiency and Hardware Awareness

In addition to detection performance, computational efficiency is a critical factor in practical cybersecurity systems. Models deployed in real-world environments must operate under constraints such as limited memory, processing power, and real-time requirements.

The final architecture generated by the PPO-based intelligent agent is relatively compact, consisting of two convolutional layers, one LSTM layer, and one fully connected layer. This structured design reduces unnecessary architectural complexity while maintaining high detection capability.

The resulting model contains approximately 0.2 million trainable parameters, indicating a lightweight architecture compared to typical deep neural network models used in malware detection tasks. This compact size contributes to reduced memory consumption and improved computational efficiency.

Furthermore, the reduced architectural depth and parameter count enable faster inference, making the model suitable for deployment in resource-constrained and real-time environments.

These results demonstrate that the proposed PPO-based framework not only achieves high detection performance but also produces computationally efficient and hardware-aware neural architectures, which is a key requirement in computer engineering applications.

5. Conclusions

The growing complexity of contemporary malware is a constant challenge to cybersecurity mechanisms, especially in areas where Windows-based platforms have predominant computing infrastructures. More classical methods of malware detection, such as signature-based systems and manually crafted machine learning models, tend to be unable to keep pace with fast-changing malware variants and the more sophisticated obfuscation methods. Though deep learning methods have enhanced the detection powers, the success of the models highly relies on the manual designing of the neural structures, which are more complex to create and tune with professional knowledge. The study proposed a reinforcement learning-based framework in which an Intelligent Agent is used to automate the system of designing neural architecture in detecting Windows malware. Through the use of the Proximal Policy Optimization algorithm, the Intelligent Agent suggested in the paper searches a simplified architecture space and sequentially finds model configurations that offer a better detection performance. The proposed framework, compared to conventional methods, which require paying a lot of attention to manual architecture design, enables the agent to improve on evaluation feedback and build an efficient neural network architecture by itself. The suggested method was tested on the EMBER Dataset, which represents a malware detection study benchmark at scale. It was shown that the architecture identified by the Intelligent Agent achieved an accuracy of 97.3, and also had good performance in other evaluation measures, which include Precision, Recall, F1-Score, and AUC. These findings affirm that the automated neural architecture search that employs reinforcement learning can be successfully used to enhance the performance of a malware detection system with respect to the relatively efficient model structure. On the whole, the results indicate the possibility of using intelligent agent-based systems to develop automated cybersecurity solutions. The proposed framework will help to create more scalable and efficient malware detection mechanisms because it will reduce the dependence on the manual model design and allow the discovery of an adaptive architecture. Future work will focus on fully integrating hardware-aware reward mechanisms to explicitly optimize the trade-off between detection accuracy and computational efficiency. Further studies can

build upon this study by considering larger spaces of architecture search, incorporating dynamic behavioral aspects, and exploring how adaptable intelligent agent-based models are to adversarial malware attacks.

Acknowledgments

The Authors would like to thank Mustansiriyah University (uomustansiriyah.edu.iq) in Baghdad, Iraq, for supporting this work.

References

[1] Y. M. Mohialden, M. T. Younis, S. A. Salman, and E. A. W. Hachim, "Challenges and advancements in quantum cryptography: Standardization, security risks, and practical implementations," Proceedings of the International Conference on Applied Innovations in IT (ICAIIT), vol. 13, no. 4, pp. 307–314, 2025.

[2] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam, "A survey on ML techniques for multi-platform malware detection: Securing PC, mobile devices, IoT, and cloud environments," Sensors, vol. 25, no. 4, p. 1153, 2025.

[3] M. Tanha and S. Kafaie, "A review of explainable machine learning for Android malware detection and analysis," IEEE Access, 2025.

[4] A. Basim, I. M. Ali, R. A. Lateef, and Z. L. Ali, "Enhanced ransomware traffic detection using a hybrid ResNeSt101 and Isolation Forest framework," in Third International Conference on Emerging Trends in AI and Computational Technologies (ICONEST 2025), Proc. SPIE, vol. 14141, pp. 87–96, 2026.

[5] Y. Dehfouli and A. Habibi Lashkari, "Memory analysis for malware detection: A comprehensive survey using the OSCAR methodology," ACM Computing Surveys, vol. 58, no. 4, pp. 1–58, 2025.

[6] M. Alshoulie and A. Mehmood, "Deep learning approaches for malware detection: A comprehensive review of techniques, challenges, and future directions," IEEE Access, 2025.

[7] I. K. Abbas, M. S. Mahdi, A. Basim, and K. I. Abbas, "An agent-based reinforcement learning framework for dynamic cryptographic security," Dijlah Journal of Engineering Sciences (DJES), vol. 3, no. 1, pp. 377–391, Mar. 2026.

[8] S. Alzahrani, Y. Xiao, S. Asiri, J. Zheng, and T. Li, "A survey of ransomware detection methods," IEEE Access, 2025.

[9] S. F. Ali, M. R. Abdulrazzaq, and M. T. Gaata, "Learning techniques-based malware detection: A comprehensive review," Mesopotamian Journal of CyberSecurity, vol. 5, no. 1, pp. 273–300, 2025.

[10] W. Almobaideen, O. Abu Alghanam, M. Abdullah, S. B. Hussain, and U. Alam, "Comprehensive review on machine learning and deep learning techniques for malware detection in Android and IoT devices," International Journal of Information Security, vol. 24, no. 3, p. 110, 2025.

[11] M. S. Mahdi, "A deep learning–based hybrid neural network model for malware detection," Journal of Techniques, vol. 8, no. 1, pp. 62–70, 2026.

[12] X. Huang, L. Ma, W. Yang, and Y. Zhong, "A method for Windows malware detection based on deep learning," Journal of Signal Processing Systems, vol. 93, no. 2–3, pp. 265–273, 2020.

[13] E. Amer and I. Zelinka, "A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence," Computers & Security, vol. 92, p. 101760, 2020.

[14] F. Çatak, A. Yazı, O. Elezaj, and J. Ahmed, "Deep learning based sequential model for malware analysis using Windows exe API calls," PeerJ Computer Science, vol. 6, p. e285, 2020.

[15] C. Wang, Z. Zhao, F. Wang, and Q. Li, "A novel malware detection and family classification scheme for IoT based on DEAM and DenseNet," Security and Communication Networks, vol. 2021, pp. 1–16, 2021.

[16] N. Azeez, O. Odufuwa, S. Misra, J. Oluranti, and R. Damaševičius, "Windows PE malware detection using ensemble learning," Informatics, vol. 8, no. 1, p. 10, 2021.

[17] M. Dener, G. Ok, and A. Orman, "Malware detection using memory analysis data in big-data environment," Applied Sciences, vol. 12, no. 17, p. 8604, 2022.

[18] V. Ravi and M. Alazab, "Attention-based convolutional neural network deep learning approach for robust malware classification," Computational Intelligence, vol. 39, no. 1, pp. 145–168, 2022.

[19] F. ALGorain and J. Clark, "Bayesian hyper-parameter optimisation for malware detection," Electronics, vol. 11, no. 10, p. 1640, 2022.

[20] S. Kudrekar and U. Rani, "Classification of malware using multinomial linked latent modular double Q learning," Indonesian Journal of Electrical Engineering and Computer Science, vol. 28, no. 1, p. 577, 2022.

[21] M. Kim, H. Cho, and J. Yi, "Large-scale analysis on anti-analysis techniques in real-world malware," IEEE Access, vol. 10, pp. 75802–75815, 2022.

[22] A. Koçak, E. Söğüt, M. Alkan, and O. Erdem, "Detection of different Windows PE malware using machine learning methods," Politeknik Dergisi, vol. 26, no. 3, pp. 1185–1197, 2023.

[23] I. E. Maulani, T. Herdianto, M. O. Laksana, F. Syawaludin, and K. Komarudin, "Exploring the effectiveness of artificial intelligence in detecting malware and improving cyber security in computer

networks," Eduvest: Journal of Universal Studies, vol. 3, no. 4, 2023.

[24] C. Du et al., "Toward detecting malware based on process-aware behaviors," Security and Communication Networks, vol. 2023, pp. 1–16, 2023.

[25] D. Syeda and M. Asghar, "Dynamic malware classification and API categorisation of Windows portable executable files using machine learning," Applied Sciences, vol. 14, no. 3, p. 1015, 2024.

[26] I. Zada et al., "Fine-tuning cyber security defenses: Evaluating supervised machine learning classifiers for Windows malware detection," Computers, Materials & Continua, vol. 80, no. 2, pp. 2917–2939, 2024.

[27] H. Gururaja, N. Khandige, N. Nayak, R. Srivatsan, and N. S., "Ensemble learning for robust malware detection in the Windows 7 environment," International Research Journal of Advanced Engineering Hub, vol. 2, no. 2, pp. 261–270, 2024.

[28] M. Imran, A. Appice, and D. Malerba, "Evaluating realistic adversarial attacks against machine learning models for Windows PE malware detection," Future Internet, vol. 16, no. 5, p. 168, 2024.

[29] S. Khan and M. Nauman, "Interpretable detection of malicious behavior in Windows portable executables using multi-head 2D transformers," Big Data Mining and Analytics, vol. 7, no. 2, pp. 485–499, 2024.

[30] "EMBER: Dataset for training static PE malware machine learning models", Kaggle, 2026. [Online]. Available: https://www.kaggle.com/datasets/dhoogla/ember-2018-v2-features. [Accessed: Mar. 14, 2026].

[31] M. Adnan, M. O. Imam, M. F. Javed, and I. Murtza, "Improving spam email classification accuracy using ensemble techniques: A stacking approach," International Journal of Information Security, vol. 23, no. 1, pp. 505–517, 2024.