Islamic University Journal of Applied Sciences (IUJAS)

https://journals.iu.edu.sa/jesc/Main/Issues

Volume VII, Issue II, December 2025, Pages 23-38

Advanced Threat Detection Using Structural Features and Graph Neural Networks for Malware Analysis

Nasser Alsharif*
Department of Sciences and Technology, Ranyah University College, Taif University
Taif 21944, Saudi Arabia

* Corresponding author: (Nasser Alsharif), Email: n.alsharif@tu.edu.sa

Abstract

Malware enriched with polymorphism, and obfuscation, has surpassed traditional signature and heuristic-based detection approaches. Machine learning and deep learning methods such as Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) have enhanced malware classification performance by utilizing static and sequential input as features. Nevertheless, the effectiveness of these approaches is limited due to their inability to model structural dependencies, which are crucial for identifying threats. This study, we propose a malware detection framework utilizing Graph Neural Networks (GNNs) to identify structural relationships within malware samples. The structural elements among the malware samples are incorporated within nodes/ and edges that apply to nodes, thereby allowing us to extract behavioral semantics that were not captured in previous models. The framework is evaluated using the EMBER dataset, which has 2,381 static and dynamic malware features; features are selected using Chi-square tests. We analyse advanced GNNs: Graph Convolutional Networks (GCNs); and Graph Attention Networks (GATs). Our findings demonstrate that the GNN-based malware detection framework outperforms classical detection methods (e.g., SVM, Random Forest, CNN, and RNN) consistently across multiple instances. This study establishes GNNs as a scalable, interpretable, and accurate approach for next-generation malware detection, and as a method that is resilient to adversarial evasion and structurally aware of malware behaviors.

Keywords: Malware Detection; Graph Neural Networks; Structural Features; GNN Explainer; EMBER Dataset.

https://doi.org/10.63070/jesc.2025.0.19
Received 27 May 2025; Revised 02 August 2025; Accepted 06 September 2025.
Available online 10 September 2025.

1. Introduction:

The pace of development for cybersecurity threats is now unprecedented, ranging from small to difficult malware attacks that are hard to detect. Traditional methods of preventing malware attacks, such as signatures and heuristic types, have long been the basis for cyber defense [1]. Nonetheless, such approaches have grave limitations, particularly in the detection of new and unknown variants of malware and even polymorphism. For example, bypassing traditional defense mechanisms, attackers will commonly employ evasion techniques, including methods for obfuscation, encryption, and adversarial adjustments [2]. This has led to a new demand for intelligent and adaptable malware detectors that are capable of evaluating and categorizing malware more effectively. The rapid strides in artificial intelligence (AI) and deep learning have yielded promising breakthroughs in recent years, and also with malware detection. Machine learning models, particularly deep learning-based methods such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), have achieved major advances in both malware classification as well as anomaly detection [3]. However, these models typically focus on static and behavioural features taken from samples of malware, treating each sample as if it were a sequence-dependent structure itself. As a result, this approach often overlooks extremely significant relational and structural dependences among malware, elements that typically provide the core of its behaviour patterns, such as execution traces for virus-related activity. Malware operates through a complex set of interactions involving API calls, system processes, and control flow relationships. Understanding these intricate dependence relations can offer a much more comprehensive perspective of how a malware instance functions and spreads [4]. Traditional machine learning models often miss out on these underlying structures, and thus their ability to detect complex malware threats is restricted [5]. In order to overcome these limitations, Graph Neural Networks (GNNs) have emerged as a powerful new choice for malware analysis and threat detection [6]. Unlike traditional deep learning methods, GNNs are explicitly designed to work with graph-structured data, allowing them to capture the complex relationships within malware samples [7].

In GNN, components of a malicious program, such as system calls, functions, and execution paths, can be represented by nodes in the graph, where interactions and dependencies between them are represented as edges [8]. This graph structure makes a much more detailed analysis of how malware behaves, which is better than traditional methods at detecting certain forms of malicious behaviour that might be difficult to find. In this regard, GNNs are able to learn from these graph representations and perceive patterns and dependencies that traditional models may overlook. Given its ability to understand structural graph patterns, a GNN-based malware detection system can improve classification accuracy, enhance generalization to different malware families, and provide increased resistance to evasion strategies used by attackers.

Despite these advances, AI-enabled malware detection also presents a number of challenges. Many of the current deep learning-based models struggle to effectively handle real-world data sets of large scale, which are the genuine source of malware data [9]. Moreover, adversarial attacks on AI-based security models cause people to wonder if these systems can be trusted and whether such attacks will work in practice against them. Despite some exploration of graph-based malware detection techniques, it has often resulted in methods with poor scalability and efficiency or that are easily manipulated by new attack strategies [10]. This study aims to fill this gap with a new kind of GNN-based malware detection system that not only increases detection accuracy but also improves model robustness and interpretability against adversarial manipulations.

To achieve this, the study focuses on designing and manufacturing a malware detection system that links graph neural networks with structural malware features. The method advanced involves transforming malware samples and their features into graphs using such as control flow graphs, dependency graphs, and system call sequences. This research adds to the current endeavours in AI-driven cybersecurity, demonstrating the potential of Graph Neural Networks for malware analysis. By leveraging relationships among malware nodes through structured data representation, GNN-based models provide an entirely new way of looking at threat recognition that is effective. This is an invaluable resource with which to enhance cybersecurity defenses in a landscape increasingly rife with complexity and digital risk. This research opens up new possibilities for the application of graph-based deep learning in malware detection, and offers an opportunity to further explore improving security solutions even upgrading them to the next level.

The remainder of this paper is structured as follows. Section 2 reviews related work on graph-based malware detection and GNNs. Section 3 outlines the proposed methodology, including the dataset, feature processing, and GNN model design. Section 4 presents the experimental setup and results. Section 5 covers model validation and explainability. Section 6 concludes the paper with key findings and future directions.

2. Related work

The use of graph-based methodologies, including GNNs, has garnered a considerable amount of attention toward malware detection and analysis. A study [11] were the first to employ GNNs for cross-architecture IoT malware detection by mapping binary files to function call graphs (FCGs) as higher-level, cross-architecture invariant features. This shows the value of using graph representations to capture complex program behaviours in a manner that is not sensitive to differences in various hardware configurations. Inspired by graph representations, authors [12] carried out an extensive review of the strength of graph-based data structures in intrusion detection systems. They suggest that graph-based models provide a higher level of abstraction or resistance to system activities and have less to evade by attackers. Similarly, a study by [13] showed that graph-based semantic analysis is very effective at decoding obfuscated code, and that GNNs are capable of distinguishing fine-grain semantic differences in binary code, which is essential for malware analysis. When applied to malware classification, the incorporation of structural features from CFGs has been the subject of much attention. Authors [14] surveyed state-of-the-art methods using CFGs with machine learning methods and summarized how diverse feature extraction and classification mechanisms can improve the accuracy of detection. This is consistent with the general tendency to use graph models to encode the intrinsic behavior of malicious software. Besides, advanced signal processing methods for graph signals emerged to possibly derive useful features to perform detection. A study by [15] proposed the graph frequency cepstral coefficient (GFCC), a new feature based on the graph Fourier analysis, to characterize the spectrum of a graph signal structured from the system data. These features can also be used in combination with structural graph features, thus yielding richer representations for malware detection. More recent studies also consider the fusion of multiple neural network architectures for detection performance improvement. A study by [16] developed financial fraud detection and showed the combination of GNN, CNN, and LSTM networks in one framework, which can learn deep information patterns for financial transactions. While being financial data-oriented, this approach demonstrates the capability of hybrid models that could be extended to malware analysis by acquiring various behavioural signs. In general, the literature suggests an increasing acknowledgement that structural features and GNNs perform and are complementary to each other in malware detection. Such approaches exploit the graph nature of program behaviours, call sequences, and system logs to improve detection decision results. GNNs can abstract the system behaviours into graph representations, meanwhile taking advantage of powerful feature extracting methods such as GSP-based feature [17], which makes them promising candidates for future malware analysis frameworks.

3. Proposed Approach

The proposed approach leverages GNNs to effectively analyze malware by exploiting structural features intrinsic to malicious software. The algorithm for the proposed approach is given in Algorithm 1. The methodology comprises the following detailed steps:

3.1 Dataset and Preprocessing

In this study, we use the EMBER dataset [18] containing 900000 training samples, a reasoning that is based on the rich, publicly available, and provides an exact description for each malicious and benign Windows executable file. It involves a diverse set of static features like PE header metadata, byte histograms, string data, imported/exported APIs, and section information. This dataset is very helpful for extracting useful patterns related to malware analysis. The dataset was pre-processed, and all the missed, noisy values were removed. The distribution of various benign sample categories and multiple malware families is distributed inversely in the training and testing subsets to improve the generalization and robustness of our model.

Algorithm 1 Adversarially-Robust Deep Learning for Intrusion Detection

```
Require: Network flow dataset $D$, perturbation magnitude $\varepsilon$
Ensure: Trained and adversarially robust model $\mathcal{M}$
 1: Preprocessing Phase:
 2:     Remove redundant features from $D$
 3:     Impute missing values using median values
 4:     One-Hot encode categorical features
 5:     Normalize all numerical features using Min-Max scaling
 6: Feature Selection:
 7:     Apply Recursive Feature Elimination (RFE) to $D$
 8:     Select top-$k$ features to obtain reduced dataset $D'$
 9: Adversarial Sample Generation (FGSM):
10: for each $(x, y) \in D'$ do
11:     Compute gradient $g_x \leftarrow \nabla_x J(\theta, x, y)$
12:     Generate adversarial sample: $x_{adv} \leftarrow x + \varepsilon \cdot sign(g_x)$
13: end for
14: Combine clean and adversarial samples:
15:     $D_{train} \leftarrow D' \cup D'_{adv}$
16: Model Training:
17:     Train hybrid 1D-CNN + LSTM model $\mathcal{M}$ on $D_{train}$
18: Model Evaluation:
19:     Evaluate $\mathcal{M}$ on clean test data $D_{clean}$
20:     Evaluate $\mathcal{M}$ on adversarial test data $D_{adv}$
21: return $\mathcal{M}$
```

3.2 Feature Extraction

In this study, feature extraction was done using both static and dynamic analysis to get more comprehensive information about malware samples. From the static analysis, both byte and structural features, such as n-gram series of opcodes, frequency of imported API calls, sizes of PE sections, entropies, and control flow graphs (CFGs) generated from disassembly binary files were extracted.

These structure and control flow-based features represent the foundation and work logic of the files in the malware. Through dynamic analysis, sandbox-execution traces, behavioural features, system call sequences, registry and file access patterns, and network communications are extracted. These runtime behaviours provide additional information, in particular when it comes to stealthy or packed malware, which could escape a static analysis approach alone. The hybrid of static and dynamic features makes the malware characteristics more robust, subsequently improving the accuracy of detection and classification.

3.3 Feature Selection via Chi-square Test

The Chi-square (χ^2) test is used as a statistical feature selection method to determine the most important features for malware classification [19]. This method is especially powerful for categorical and count features like API use counts or the presence of opcode n-grams. The feature-target (malicious vs. benign) class label association strength under the independence assumption is taken into consideration by applying the Chi-square test as given in Eq. [1]. Features with high Chi-square values are informative and class-dependent, indicating that the feature is highly associated with the classification target.

The feature selection process involves the following steps:

1. For each feature f_i, compute the Chi-square statistic

$$\chi^2 \sum \frac{(O-E)^2}{E} \qquad (1)$$

where O is the observed frequency and E is the expected frequency assuming independence between the feature and class label.

2. Rank all features based on their $\chi2$ scores.
3. Select the top-k features that exceed a predefined significance threshold (e.g., $p < 0.05$) or retain the top N% percentile features for further processing.

This feature selection step reduces dimensionality, lowers computational overhead, and enhances the signal-to-noise ratio by removing irrelevant or redundant features [20]. Ultimately, it improves the performance and efficiency of the Graph Neural Network by ensuring the model focuses on the most discriminative and statistically significant inputs.

3.4 Structural Feature Encoding

The structural features are the basis of the graph-based malware detection approach used in this study. Structural features. Unlike flat statistical features, structural features can capture the intrinsic architectural and relation properties of the malware binaries. Important structural characteristics of both are encoded into directed graphs, including CFGs, opcode n-gram sequences, function/API call dependencies, and PE section placements. Nodes denote semantically relevant entities such as basic blocks or API calls, and edges indicate their control, calls, or data dependence. This graph representation enables GNN to capture rich patterns of malicious interactions that cross different components and take into account the execution logic, invocation sequences, and modular interactions. By encoding this structure explicitly, the model becomes more effective in detecting stealthy/ polymorphic malware that conducts simple changes on surface-level features but keeps important structures in the underlined architecture. The incorporation of this structural information not only enhances classification accuracy but also contributes to generalization to various malware families.

3.5. Graph Construction and Representation

To encode structural characteristics of malware in an interpretable graph representation, malware samples need to be modelled as a graph-based representation capturing their underlying program logic, behavioural characteristics, and control dependencies. In this approach, the internal structure of an executable is represented as a directed graph $G(V, E)$, where V is a set of nodes and E is a set of directed edges between them. This structured encoding allows the model to capture the topological patterns and context relationships underlying malicious binaries, which are often discarded in standard flat features-based models.

Graph nodes represent semantically meaningful components of a program. These are features based on functions present in the binary (custom-defined and system-defined), API calls that the malware uses to communicate with the operating system, and basic blocks in the CFGs, which are a sequence of instructions with a single entry and single exit point. Given a node $v_i \in V$, it has the associated feature vector x_i, that encodes opcode n-gram frequencies, PE section metadata, entropy scores, API type categories, and execution statistics. This feature vector has a compact but informative representation capturing how the program unit participates in the role and behaviour of the executable.

It contains edges that represent relations between nodes that correspond to program control and data flow. For example, an edge from the node $v_i$to the node v_j, represented by $e_{ij} \in E$, could be the edge for a function call, a control transfer, or usage of any common variables. These edges can be bound by a weight or type according to the type of relationship they indicate. For instance, control flow edges represent the flow of instruction execution; call graph edges denote static or dynamic calls; and data flow edges represent the flow, transformation, or correlation of the data between program elements. In graph-based malware analysis, edges represent control-flow, calls, or data dependencies, often weighted by metrics like call frequency. For dynamic analysis, temporal order captures execution sequence.

Nodes and edges are annotated with semantic metadata to assist subsequent learning. Node labels classify entities into several groupings like “network-related API”, “file access routine”, or “registry operation”, depending on behaviour or API family. Edge labels are used to specify the relationship, and they provide some guidance to the model to differentiate between control transitions and functional dependencies.

This graph G is the input to the GNN, which learns node embeddings through iterative message passing. At each layer l of the GNN, the embedding of the node v_i is updated as expressed in Eq. (2).

$$h_i^{(l)} = \sigma(\sum_{j \in \mathcal{N}(i)} f(h_i^{(l-1)}, h_j^{(l-1)}, a_{ij})) \qquad (2)$$

Where, $h_i^{(l)}$ is the node representation at layer l, $\mathcal{N}(i)$ is the set of neighbours of node i, $f(.)$ is a learnable aggregation function, σ is a non-linear activation function ReLU

For multiple rounds of such updates, node embeddings learn multi-hop neighbourhood information and structural dependencies. These node-level embeddings are pooled to obtain a graph-level representation of the overall graph. This final vector x_i passed through a classification layer to predict whether the sample is benign or a member of some malware family k. Transforming executable files into graphs and capturing their structural and behavioural correlations, the representation is conducive to detecting subtle malicious nuances in executables, even in their obfuscated or polymorphic form. It forms the basis for the construction of a strong, scalable , and interpretable GNN-based malware detection framework.

3.6 GNN-Based Detection Model

In order to develop an efficient and reliable malware detection system to classify malware samples based on structural properties, we construct a GNN-based malware detection model, which is able to capture both local and global features from graph-structured representations of executable codes. The model structure combines sophisticated graph embedding methods, multiple GNN layers for deep feature encoding, and an end-to-end supervised method for binary/multi-class classification tasks. The architecture of GNN is given in Fig. 1.

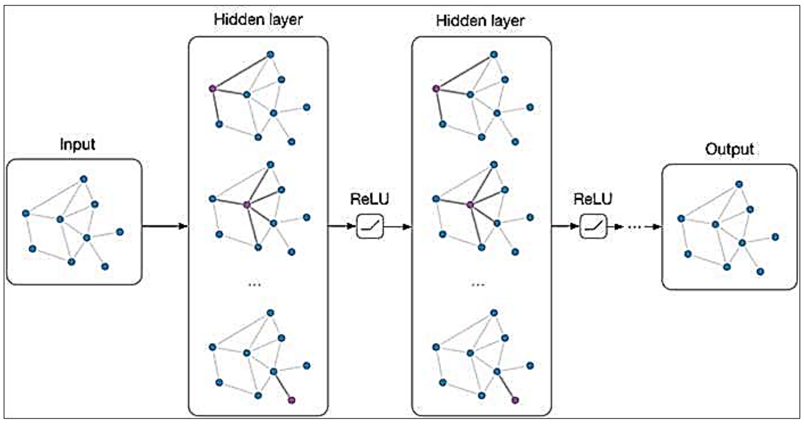

Fig: Graph Neural Network

In the first stage, graph embedding translates each node in the malware graph to a high-dimensional dense vector that captures the local context, structural role, and meaning of the node. For this, we experiment with three popular forms of GNNs: Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and GraphSAGE. GCNs summarise the information of the neighbours of a node via a convolution-like operation. Then the feature update rule of a GCN layer can be formulated as given in Eq. (3).

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} \tilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3}$$

Where, $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\widetilde{D}$ is the corresponding degree matrix, $H^{(l)}$ is the matrix of node features at layer l, $W^{(l)}$ is the learnable weight matrix, σ is an activation function ReLU.

GAT utilizes an attention mechanism that assigns different weights for the neighbour nodes during the aggregation process to enable the model to pay more attention to important structural correlation.

GraphSAGE, in turn, promotes scalability by sampling and aggregating features from a bounded number of neighbours. When node embeddings are extracted, a graph-level embedding is created through a readout or pooling operation on all nodes' representations. We use pooling techniques, including global mean pooling, global max pooling, and attention-based pooling to obtain a single vector $g \epsilon \mathbb{R}^{d}$ to summarize the entire graph as given in Eq. (4).

$$\mathbf{g} = \mathbf{POOL}\ (\{\mathbf{h_i} | \mathbf{v_i} \in \mathbf{V}\}) \tag{4}$$

This graph-based representation captures the holistic structural/behavioural fingerprints of the malware and is fed into the classification network. The whole model architecture consists of multiple layers of stacked GNNs, which achieve hierarchical representation learning and are capable of capturing the complex and high-order relations among graph nodes. To counter the vanishing of gradients and the degradation of features in deep models, we introduce residual connections across GNN layers, allowing for more effective gradient spreading as well as for the preservation of lower-layer information. The final layer of the GNN is followed by one or more fully connected (dense) layers, which play the role of the classifier. These layers interpret the learned graph-level embeddings and map them to the labels at the output layer, which can be binary (malicious or benign) or multi-class (malware family types). The final result is sent through a softmax or sigmoid activation function based on the classification change.

The model is trained in a supervised manner with the cross-entropy loss that measures the discrepancy between the predicted class probabilities and the ground truth labels, Eq. (5).

$$\mathcal{L} = \sum_{i=1}^{N} y_i log(\hat{y}_i) \tag{5}$$

Where $\boldsymbol{y_i}$ is the true label and $\boldsymbol{\hat{y}_i}$ is the predicted probability for sample $\boldsymbol{i}$.

For stable and fast optimization, we use the Adam optimizer, with the adaptive learning rate and momentum-based updates. We also use multiple regularizations to promote generalization and reduce overfitting, such as Dropout layers between dense layers. Weight decay to penalize large weights. Early stopping, monitoring validation loss to halt training when performance plateaus.

Hyperparameters (e.g., learning rate, batch size, the number of GNN layers, the embedding dimension, the dropout rate) are optimized either by grid search or Bayesian optimization to search for the most suitable model setting. This GNN-based design, and the capability it possesses to learn meaningful representations of graph-structured malware, forms a strong, scalable base for modern threat discovery and malware characterization.

4. Experimental Evaluation

We verify the efficiency and generalization of the proposed GNN-based malware detection method through large-scale experiments on the EMBER dataset. At first, the preprocessing on the dataset was performed, and the dataset was divided into three parts: train, validation, and test. We use a stratified 80/10/10 split that ensures each of the split subsets has a proportional number of the malicious as well as the benign samples. This stratification is important to prevent the class imbalance from affecting the performance metric and to improve the robustness of the model. The feature selection based on the Chi-square test is adopted before model training to keep only the most discriminative features. Using the Chi-square test, only 1000 features were used out of 2,381 features available in the dataset.

Table 1: Performance Evaluation

Evaluation Method	Formula	Description
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Measures the proportion of correctly predicted samples among the total predictions.
Precision	$\frac{TP}{TP+FP}$	Indicates how many predicted positive samples are truly positive.
Recall	$\frac{TP}{TP+FN}$	Measures how many actual positive samples were correctly predicted.
F1-Score	$\frac{2.Precision.Recall}{Precision+Recall}$	Harmonic Mean of precision and recall, balancing false positives and false negatives.

The GNN model and the baselines were implemented in Python 3.10, using PyTorch, and Geometric and Scikit-learn were used for building the models. All experiments are conducted on a machine with an NVIDIA RTX 3090 GPU (24 GB VRAM), an Intel Core i9 CPU, and 64 GB RAM and operating on Ubuntu 22.04 LTS. The use of GPU acceleration greatly facilitates the training speed of GNN layers, especially when graphs have a large number of nodes. The performance metrics used in the study are given in Table 1ensure a comprehensive and standardized evaluation of detection performance.

To validate the performance of our proposed model, we compare it against several state-of-the-art baseline classifiers used in the malware detection literature, such as Support Vector Machine (SVM), which is a discriminative margin-based classifier that is well-suited for high-dimensional data. Random Forest (RF), which is a simple bagging-based classier that is commonly used in the literature as a performance benchmark for malware detection problems; CNN a deep learning architecture typically applied on transformed feature vectors or image-based representations of malware, RNN which is a powerful class of neural networks that is designed to capture sequential dependencies, and hence performs well in analyzing opcode or system call sequences.

Table 2: Experimental Results

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**	**ROC-AUC (%)**
SVM	92.1	90.2	89.8	90.0	93.7
Random Forest	94.5	92.7	92.1	92.4	95.3
CNN	95.8	94.0	93.6	93.8	96.2
RNN	95.3	93.7	93.1	93.4	95.8
Proposed GNN	**99.1**	**98.4**	**97.9**	**98.2**	**99.7**

As proven by the results given in Table 2, the proposed GNN model performs better than all baselines in all metrics and attains an accuracy of 99.1%, a precision of 98.4%, and an ROC-AUC of 99.7%, indicating excellent identification accuracy of malicious versus benign samples. This gain is mainly because GNN can capture the hierarchical and relational features by message passing, both of which traditional models and sequence-based deep networks are not able to perform. Also, the higher values of Recall and F1-score of the GNN model demonstrate robustness in detecting more malware samples with a lower number of false negatives, which is important in cyber cybersecurity area where a missed detection can be of extreme consequences.

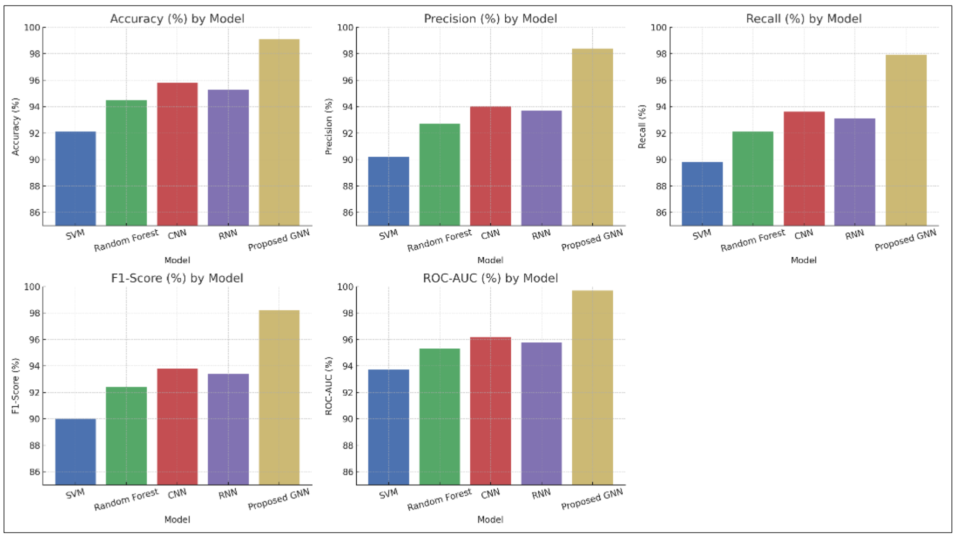

Fig. 2. Comparison of performance evaluation

The experimental results and comparison of various models, as given in Fig. 2, justify that returning to graph-based structural learning by using GNNs is very effective without requiring the prior knowledge of designing new classic machine learning models, as well as traditional deep learning models.

5. Model Validation

In order to make sure that the designed GNN-based malware detection model is both effective enough and reliable enough, as well as generally applicable, quantitative validation and qualitative explainability analysis were applied. These attempts offer an analysis of the decision-making process of the model, and confirm that the learned patterns are indeed semantically and functionally meaningful.

We initially use k-fold cross-validation ($k = 5$) to test the robustness and the generalization ability of the model on various subsets of the dataset. This provides a test set of 37 data points and ensures each data point goes through both training and testing. Performance is averaged across folds to reduce overfitting and the chance that an evaluation may be biased towards a specific data split. The standard deviation of accuracy, precision, and F1-score across the different folds is still less than 0.5%, which indicates that the model has a stable performance.

We employ explainability methods tailored for graph-based learning. In particular, we adopt GNNExplainer, the state-of-the-art explainable framework on GNNs that finds the most important subgraph and node features for a given prediction. GNNExplainer works by training a soft mask on the input graph that identifies a few nodes and edges with maximum influence on the model's output.

Given a sample classified as malicious, GNNExplainer finds the most influential paths (e.g., sequences of API calls or control flow transitions) used by the model to reach its conclusion. This helps to check that the model is not overfitting to random correlations. Extract signatures by identifying the behavioural factors of the malware. We further investigate overall trends by combining results over all samples, demonstrating that the GNN invariably attends to behaviour-rich parts (i.e., file system manipulation, network, and registry access sequences), which are identifiable characteristics of maliciousness. Conversely, the benign examples have naturally occurring graphs, with low interaction breadth and easily discernible structural motifs.

This interpretability analysis not only validates the semantic correspondence that exists between the model's attention and known threat patterns but also imparts trust for real-world deployment of GNN-based detection systems. It also provides a window for adding human-in-the-loop feedback mechanisms in which expert analysts can refine or validate the automatic predictions. The simultaneous application of cross-validation and graph explainability methods guarantees that the resulting model is not only accurate but also transparent, reliable, and auditable, all of which are essential to a model that can be deployed in modern cybersecurity environments. These experimental results show the effectiveness of utilizing graph representations to improve the detection, generalization, and evasion resistance.

6. Conclusion

In this work, we propose a novel, robust, and scalable malware detector driven by GNNs, and show substantial performance gain compared to traditional machine learning and deep learning approaches. The proposed GNN-based approach takes advantage of structural information such as control flow graphs, API call dependencies, and system call sequences to discover complex relational patterns that are ignored in flat or sequential input representations. Experiments on the EMBER dataset demonstrate that the performance of the updated model surpasses that of state-of-the-art works, with the testing accuracy and ROC-AUC scores reaching 99.1% and 99.7% respectively, and having an edge over traditional classifiers such as SVM, Random Forest, and deep learning models such as CNN and RNNs by all metrics. Additionally, incorporating the GNNExplainer module improves the model interpretability by introducing key subgraphs and node features, leading to classification results that are important for trust in real-world cybersecurity scenarios. In addition, the generalization and robustness of the proposed model are further verified by 5-fold cross-validation, and its performance variance is stable. These results demonstrate the effectiveness of using structural awareness from GNNs in malware detection, providing an effective line of defense against polymorphic and evasive threats. This study paves the way for future studies in explainable and adaptive graph-based malware analysis mechanisms, taking a step forward to more intelligent, interpretable, and robust cybersecurity approaches.

References:

[1] M. I. Malik, A. Ibrahim, P. Hannay, and L. F. Sikos, “Developing resilient cyber-physical systems: a review of state-of-the-art malware detection approaches, gaps, and future directions,” *Computers*, vol. 12, no. 4, p. 79, 2023.

[2] F. K. Alarfaj and N. A. Khan, “Enhancing the performance of SQL injection attack detection through probabilistic neural networks,” *Applied Sciences*, vol. 13, no. 7, p. 4365, 2023.

[3] A. Redhu, P. Choudhary, K. Srinivasan, and T. K. Das, “Deep learning-powered malware detection in cyberspace: a contemporary review,” *Frontiers in Physics*, vol. 12, p. 1349463, 2024.

[4] C. Wei, Q. Li, D. Guo, and X. Meng, “Toward identifying APT malware through API system calls,” *Security and Communication Networks*, vol. 2021, no. 1, p. 8077220, 2021.

[5] A. A. Alqarni, N. Alsharif, N. A. Khan, L. Georgieva, E. Pardade, and M. Y. Alzahrani, “MNN-XSS: Modular neural network based approach for XSS attack detection,” *Computers, Materials and Continua*, vol. 70, no. 2, pp. 4075–4085, 2022.

[6] L. Li, F. Qiang, and L. Ma, “Advancing Cybersecurity: Graph Neural Networks in Threat Intelligence Knowledge Graphs,” in *Proc. Int. Conf. Algorithms, Software Engineering, and Network Security*, Apr. 2024, pp. 737–741.

[7] H. Shokouhinejad et al., “Recent advances in malware detection: Graph learning and explainability,” *arXiv preprint arXiv:2502.10556*, 2025.

[8] D. Zapzalka, S. Salem, and D. Mohaisen, “Semantics-Preserving Node Injection Attacks Against GNN-Based ACFG Malware Classifiers,” *IEEE Transactions on Dependable and Secure Computing*, 2024.

[9] M. A. Hossain et al., “AI-enabled approach for enhancing obfuscated malware detection: a hybrid ensemble learning with combined feature selection techniques,” *International Journal of System Assurance Engineering and Management*, pp. 1–19, 2024.

[10] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, “A survey on malware detection with graph representation learning,” *ACM Computing Surveys*, vol. 56, no. 11, pp. 1–36, 2024.

[11] C. Li, G. Shen, and W. Sun, “Cross-architecture Internet-of-Things malware detection based on graph neural network,” in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, Jul. 2021, pp. 1–7.

[12] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, “Graph neural networks for intrusion detection: A survey,” *IEEE Access*, vol. 11, pp. 49114–49139, 2023.

[13] R. Cohen, R. David, F. Yger, and F. Rossi, “Identifying Obfuscated Code through Graph-Based Semantic Analysis of Binary Code,” in *Int. Conf. Complex Networks and Their Applications*, Dec. 2024, pp. 135–148. Cham: Springer Nature Switzerland.

[14] S. Mitra, S. A. Torri, and S. Mittal, “Survey of malware analysis through control flow graph using machine learning,” in *Proc. IEEE 22nd Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCom)*, Nov. 2023, pp. 1554–1561.

[15] L. Xu et al., “A Novel Feature Based on Graph Signal Processing for Detection of Physical Access Attacks,” in *Odyssey*, 2022, pp. 107–111.

[16] Y. Cheng et al., “Advanced financial fraud detection using GNN-CL model,” in *Proc. Int. Conf. Computers, Information Processing and Advanced Education (CIPAE)*, Aug. 2024, pp. 453–460.

[17] L. Xu et al., “A Novel Feature Based on Graph Signal Processing for Detection of Physical Access Attacks,” in *Odyssey*, 2022, pp. 107–111.

[18] “EMBER dataset,” Available: https://github.com/elastic/ember. Accessed: Apr. 10, 2025.

[19] I. S. Thaseen and C. A. Kumar, “Intrusion detection model using fusion of chi-square feature selection and multi class SVM,” *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017.

[20] N. A. Khan, M. Y. Alzaharani, and H. A. Kar, “Hybrid feature classification approach for malicious JavaScript attack detection using deep learning,” *International Journal of Computer Science and Information Security*, vol. 18, no. 5, 2020.