

A Deep Hybrid Architecture for Zero-Day Malware Detection

Mr. V. Nag Raju , Department Of Information Technology and computer application & Andhra University college of Engineering
Second Author Department & Andhra University college of Engineering

Mr.S.Vinay Kumar, Department Of Information Technology and computer application & Andhra University college of Engineering

ABSTRACT

Modern malware has evolved to deliberately obscure its structure and suppress observable behaviour, reducing the effectiveness of conventional signature-based detection techniques, particularly for zero-day and previously unseen threats. At the same time, many advanced behavioural detection systems impose substantial computational and deployment overhead, limiting their practicality in controlled or resource-constrained environments.

To address this challenge, this paper presents a lightweight multi-path malware detection framework for Windows executables that jointly analyzes static file structure, coarse-grained runtime behaviour, and basic contextual metadata within a unified classification pipeline. Static analysis provides rapid pre-execution assessment based on intrinsic binary characteristics, while dynamic analysis observes low-overhead execution behaviour using CPU and memory utilization. In addition, metadata-level attributes offer supplementary context when deeper inspection is restricted. The outputs from these complementary analysis paths are integrated using a weighted fusion strategy to derive the final detection decision.

Experimental evaluation on a balanced dataset of 100,000 Windows executable files shows that the proposed framework achieves a detection accuracy of 96.4% with a ROC-AUC of 0.98, consistently outperforming individual analysis paths. These results show that combining multiple lightweight analysis perspectives can significantly increase detection reliability without incurring substantial computational overhead, making the framework suitable for academic research and controlled malware analysis environments.

Keywords— Malware Detection, Static Analysis, Dynamic Analysis, Metadata Analysis, Convolutional Neural Network (CNN), Graph Neural Network (GNN), Feature Fusion, VirusTotal FastAPI Zero-day malware, Cybersecurity.

1.INTRODUCTION.

1.1. Rise of Malware Threats in Cyber security.

The growth in connected systems has made malware a bigger issue than before. It finds ways in through bad code or setups that are not safe. Attackers keep finding new ways to stay hidden with things like packing the code or changing it around so it does not match what detectors expect.

It seems like older tools that check against lists of known bad files miss a lot now. Especially the new kinds that have not been seen yet. Modern ones also check the environment before doing anything bad.

Malware goes through stages to get on a system and then keep control. Different tricks are used at each point to avoid being spotted. This makes it stay around longer and cause more problems.

Some people see signature methods as not enough anymore. The idea is to look at files in different ways before and during running plus some extra details. That is the part that stands out for handling these threats.

A framework that does static checks and dynamic ones together might help but I am not totally sure how well it works in all cases.

1.2. Limitations of Existing Systems.

Despite significant research in malware detection, contemporary literature describes the existing systems as limited in their ability to address the problem. Mostly, it is linked to the detection accuracy, robustness, and cost since the majority of solutions prioritize one aspect while neglecting others

The first category of malware detection systems includes signature-based solutions that are relatively simple to implement and do not require high computational resources. The systems rely on the database of known malware samples that are matched to the suspicious binaries at run-time. However, the main limitation of signature-based detection is the inability to address zero-day exploits and polymorphic or metamorphic malware that change their binaries to appear

different from known samples every time they are launched [1] [5]. Overall, the solutions are light on resources but lack robustness and may have a high rate of false negatives for contemporary malware.

More advanced systems utilize the static analysis of executables to extract relevant features and match them to the known patterns. Compared to signature-based detection, such systems are more reliable but still lack accuracy and do not address polymorphic or metamorphic malware [1] [5]. The main reason for it is that such solutions analyze only the code of an application and its imports, making it possible to design malware with altered instruction patterns or encoded data section that would not be detected as malicious. Thus, the approach is computationally inexpensive and safe since it does not execute the code, but it is not sufficient to detect modern malware.

In turn, the dynamic analysis aims to address the limitations of the previous approaches by relying on run-time behaviors of applications in an isolated environment. The literature review shows that the approach is effective since malware detection systems that utilize dynamic analysis can reveal the malicious behaviors that static detection misses [2] [4]. With that said, such systems are associated with high resource costs since isolated environments where test executables are launched require extra RAM and processing power. In addition, with the evolution of malware, modern malicious software can mask its presence in isolated environments making it undetectable by dynamic detection systems. Altogether, the approaches are associated with high computational costs and complexity of implementation that limits their widespread use.

Finally, contemporary research explores the possibility of utilizing natural language processing and large language models for malware detection. The literature review shows that the methods go as deep as analyzing the intent of APIs called by malicious software and leverage the power of large language models to detect subtle patterns [6] [10]. However, such approaches are excessive and impose extra costs and complexity on the system while potentially sacrificing some accuracy due to the complexity of language models.

1.3. Research Contributions

This work presents a novel lightweight three-path malware detection framework which combines static, dynamic and metadata-based analysis in a single pipeline. Unlike most of the prior literature, we do not propose any new learning algorithm for malware detection; instead, we present an analysis framework that leverages multiple analysis paths and carefully designed fusion mechanisms for more reliable detection in targeted and controlled environments.

The main contributions of this work are summarized as follows.

We propose a novel three-path malware detection framework that systematically combines static, coarse-grained dynamic, and metadata-based analysis in a single classification pipeline.

We design a static analysis module based on a one-dimensional convolutional neural network to learn ordered structural dependencies from PE files in a purely unsupervised manner, thus avoiding time-consuming manual feature engineering and file execution.

We incorporate a lightweight dynamic analysis module that captures only the essential run-time behavioral patterns of analyzed samples through CPU and memory utilization during sandboxed execution, thus avoiding the complexities of system call-level monitoring.

Additionally,

we design a metadata-based analysis path that utilizes basic file properties as auxiliary features for detection if richer static and dynamic analysis is not feasible.

Finally,

we implement and evaluate a weighted fusion mechanism that combines evidence from all analysis paths, achieving significantly better performance than any individual analysis module.

We evaluate the proposed framework on a large-scale benchmark of Windows PE files, demonstrating its reliability and computational efficiency in the discovery of malware samples for academic and targeted malware analysis.

2. LITERATURE REVIEW.

As of now, there is still much room for improvement in malware detection. The traditional signature-based method, while very efficient in detecting known malware, fails to detect the novel, obfuscated, and zero-day malware. That has led to the adoption of machine learning and deep learning approaches in malware detection research. By learning discriminative features from data, these approaches attempt to generalize beyond the set of predefined features used in the signature-based method. Recent studies show that adding information from static features, runtime behavior, and hybrid analysis leads to enhanced detection performance.

Early attempts at using machine learning in malware detection mostly consisted of static analysis of PE files. For example, Kamdan et al. [1] used Random Forest classifier trained on handcrafted PE features and reported a very high classification accuracy, yet, as always, such methods depend highly on

manual feature engineering and cannot be easily generalized to novel malware. In their study, Yousuf and Alazab [5] also used PE header attributes, DLL imports, and section-level features together with classical classifiers and reported good results for known samples, while having lower effectiveness for packed/obfuscated ones.

Limitation: Static approaches are lightweight and safe, yet they are still vulnerable to structural evasion techniques and are not able to generalize.

One way to alleviate such a limitation is the use of dynamic analysis techniques and analysis of runtime behavior. For example, Hammi et al. [2] monitored system calls in sandboxed environment and built classifiers using the sequences of these calls, thus showing good results in detecting evasive malware, while suffering from significant computational overhead. Akhtar and Feng [4] showed how the use of ensemble learning methods increases the behavioral detection accuracy, although, as mentioned above, such techniques require continuous monitoring.

Limitation: Runtime behavior tracing is costly and is vulnerable to the sandbox-evasion techniques.

With the development of deep learning approaches, researchers started exploring different neural architectures for malware detection. Alomari et al. [3] showed that combining Dense and LSTM layers with correlation-based feature selection improves the efficiency of the model, though, as usual, requires very cautious tuning to prevent overfitting. Thiziers et al.

[7] applied artificial neural network for static malware classification and showed reasonable accuracy, but failed to capture long-range dependencies.

Limitation: Deep neural approaches increase accuracy but also increase the complexity of the training process and deployment cost.

More recent research focuses on the use of graph and semantic-aware approaches. Wang et al. [8] suggested the PARIS – the malware detection system based on Event Tracing for Windows that adaptively collects behavioral traces to minimize monitoring overhead. [13].

Limitation: Despite having high detection accuracy, such approaches require fine-grained behavioral tracing, complex graph construction, and semantic enrichments, thus, having a high preprocessing cost, inference latency, and deployment complexity.

3. RESEARCH GAP.

Malware detection research has thus far tended to pursue static analysis, dynamic monitoring, and deep learning-based detection as mutually exclusive areas of investigation, pushing each towards an increasingly one-sided specialization without accounting for the other two. Fast and safe for automated execution, signature-based static approaches such as the random-forest or the PE header feature classifier were always inherently brittle and vulnerable to any kind of sample obfuscation or packing, leaving them no useful data to parse once the code got hidden. This is why sandbox-based

approaches aimed to address this issue through observing the runtime behavior of a file, but even such studies as system call analysis proved to restore accuracy on evasive samples at the cost of the considerable speed decrease, high memory footprint, and vulnerability to sandbox-aware malware itself.

However, the most troubling aspect of these two approaches lies precisely in their attempt to merge into hybrid solutions; while the current state-of-the-art examples prove that fusion outperforms each of these approaches separately, they almost always perform the fusion between full-fledged static and dynamic paths, thereby adding the price of the former to the already expensive latter without considering the possibility of finding a more economical path for achieving similar results. What has thus far been notably absent from the current literature was the proof that cheap behavioral indicators (such as CPU and memory utilization rate compared to system calls' sequence) are still sufficiently meaningful to be merged in. This is the area where the current research operates: it is clear that fusion helps to improve accuracy in malware detection, but the question is whether fusion could still prove beneficial after cutting down the dynamic path in order to fit the resource constraints of a controlled environment. The three-path system with CNN-based static path, GNN-based behavioral path, and the metadata-based one, fused according to the aforementioned principle is designed specifically to answer this question.

4. OBJECTIVES.

Objectives of the project are testing the hypothesis that it is possible to gain meaningful information from dynamic analysis without using its costly variant.

The static part of the project is intended to generate a predictive model of file evaluation – entropy value, sections' structure, imported APIs etc. This task is done by the means of 1-D CNN as it is capable of learning structural features without manual feature selection.

The real problem lies in the dynamic part of the project. In contrast to the tracing of all system calls made by the analyzed application, the intention here is to use much less – only CPU and RAM behavior on a short sandboxing period, analyzed with GNN. The ability of such a method to add a meaningful signal to the evaluation of a file will provide valuable information regarding the ratio of the actual cost and cost necessary for dynamic analysis. Also, since the type and the size of the file do not add significant costs to the evaluation process, a path for additional metadata analysis is added, especially for those files where detailed analysis is not applicable.

Then, the objectives turn into the way to combine the obtained results. The weights to combine the scores and decision

threshold are supposed to be learned from the validation dataset and not taken arbitrarily.

5. PROPOSED ARCHITECTURE.

The concept of the architecture is straightforward enough: instead of having one expensive detector, we use three cheap detectors running simultaneously. The input file is inspected by three independent paths: each makes up its mind before any attempt to merge those opinions takes place.

5.1. Static Path

As follows from its name, this one does not involve any execution of code whatsoever. It breaks down structural elements of the file under scrutiny: entropy, structure of sections, imports from Windows API. Those features start becoming peculiar once the file has been compressed and/or encrypted to conceal its intent. All of those values are assembled into ordered vector and fed to a small 1-D CNN: convolution with 64 filters and kernel size of 3, followed by a ReLU, max-pooling in order to reduce amount of noise, then two fully-connected layers (with 128 and 64 neurons accordingly) and sigmoid as output layer that gives us probability value. Fast and cheap – that's pretty much what this path should be.

5.2. Dynamic Path

This is the heart of our paper. Traditional way of conducting dynamic analysis relies on tracking all system calls made by an analyzed program; that approach works indeed, but it is too slow and resource-intensive. Our approach involves running the file in a sand-boxed VM and monitoring two values only: CPU and memory usage during that period of time. Again, that is all information we are after. This data is transformed into a graph: nodes – timestamped observations, edges – relationships between them. The graph is processed with GNN which gives us one probability value as output. As you can see, we go for extremely low-level information. And whether this level of detail provides us with something worth investigating is the question.

5.3. Metadata Path

A tiny fraction of the overall effort in terms of size. It simply evaluates file type and size to generate a coarse measure. This is not expected to do the entire job in itself — this path of evaluation is only meant for those cases where there can be no more in-depth scrutiny.

5.4. Fusion and Decision

After producing their scores — p_s , p_d , and p_m — all three paths get merged using a weighted sum, where the weights aren't arbitrary but rather optimized against the validation set such that the relative importance of each individual path roughly corresponds to its actual importance. After this, the score gets compared to a threshold, which is also determined from the data itself — in particular, by optimizing Youden's Index ($J = \text{TPR} - \text{FPR}$) on ROC curves to produce 0.7 in this case.

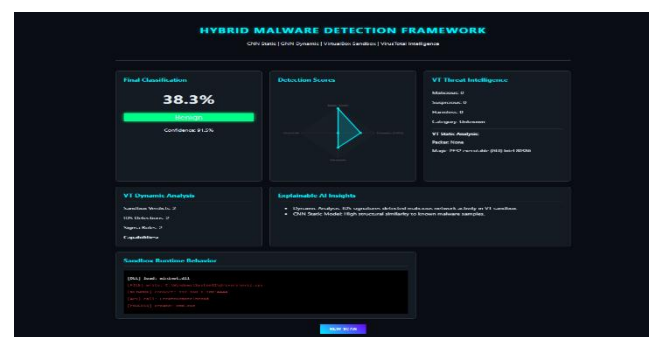
If the score is higher than that threshold, then a file is labeled as malware, and if it's lower than or equal to it, then the file is considered benign.

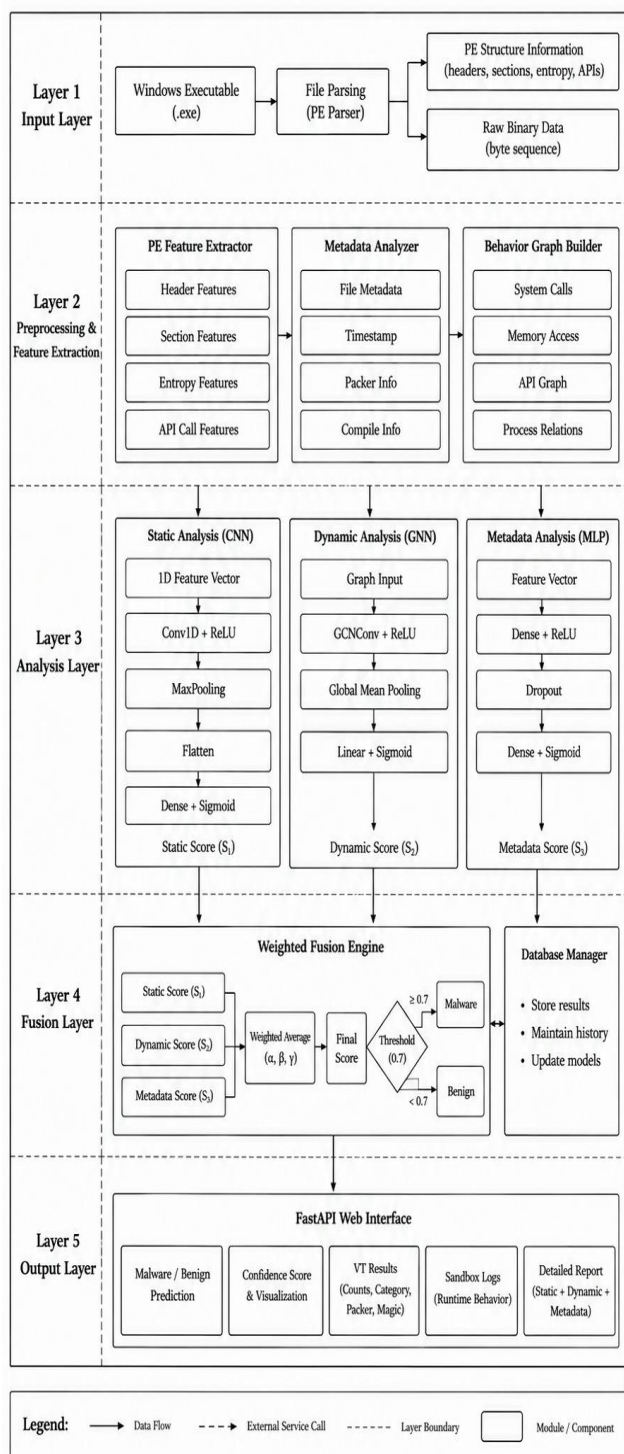
Therefore, the overall architecture is simply three cheap opinions and one good way of comparing them to each other — with the idea being to simulate the results of an expensive one without the cost.

6. SYSTEM ARCHITECTURE

The diagram illustrates how the entire Hybrid Malware Detection Framework works. First, the Windows executable is fed into the system where relevant information is parsed and extracted from it. After that, the extracted information is evaluated from various angles - the 1D CNN evaluates static attributes of the file, while the GNN evaluates the runtime features and the metadata analysis evaluates more file information.

The output of the three above evaluations is fused to provide a score which determines if the file is malware or not. The prediction results will be recorded into the database and via the FastAPI web interface, the user can see the prediction, confidence score, VirusTotal output, sandbox logs, and analysis of the file. Output interface:





The static analysis path considers the executable without its execution, thus, decreasing potential security issues related to processing of unknown files. The static analysis path extracts the set of features including such as file entropy, sections characteristics, imported Windows API indicators. Then, the features are normalized and represented in form of one-dimensional vector to be processed by 1-D CNN. Specifically, 1-D CNN uses 64 filters with the kernel size of 3, ReLU activation, and max-pooling functions. Afterward, two fully-connected layers with 128 and 64 neurons follow and, finally, a sigmoid function outputs malware probability P_s .

7.2 Dynamic Analysis Design

The dynamic analysis path observes behavior of the executable during its execution. In order to protect the system, each file is executed in controlled environment (sandbox/virtual machine). At the same time, the framework captures CPU utilization and memory consumption as basic behavioral features of executable.

The observations obtained at certain timestamps are organized in a form of a temporal graph, where nodes correspond to observed events and edges show relationships between these events in the time domain. The temporal graph is processed by a GNN, which outputs a malware probability, denoted as P^d . This probability is later combined with results of static and metadata analysis paths to obtain the final prediction.

7.3 Metadata Analysis Design

The metadata analysis path supplies basic metadata information regarding the executable without running the latter and without executing code analysis. It includes file type and file size among others.

These metadata are used as input features in the metadata model which outputs malware probability, represented as P_m . Although the metadata analysis itself is not able to classify sophisticated malware, it can serve as additional feature set to combine the results of other paths.

7. SYSTEM DESIGN

The proposed system utilizes three paths for the malware probability calculation: static, dynamic, and metadata analysis. Each path performs an analysis of the executable under consideration from a particular point of view, and calculates a malware probability that will be further fused in weighted fashion in order to produce a final probability.

7.1 Static Analysis Path Design

7.4 Mathematical Model

Each analysis path generates a particular malware probability independently. In order to calculate the final malware probability, the proposed system utilizes weighted fusion mechanism.

Let P_s denote the probability generated by static analysis path, P^d denote the probability generated by dynamic analysis path, and P_m denote the probability generated

by metadata analysis path. The final malware probability is calculated as:

$$P_f = \alpha P_s + \beta P_d + \gamma P_m$$

where α , β , and γ are the weights assigned to the static, dynamic, and metadata analysis paths. The weights satisfy the following condition:

$$\alpha + \beta + \gamma = 1$$

Pf value is the resultant probability of whether the given input is indeed malicious in nature or not.

Once the fused probability is obtained, the next step involves setting the decision threshold. This decision threshold is determined through ROC and Youden's Index, which is given by:

$$J = TPR - FPR$$

In which TPR stands for true positive rate and FPR stands for false positive rate.

For the new proposed system, the chosen threshold value is $\theta=0.7$. Hence, the classification decision is made as:

$$\text{Classification} = \begin{cases} \text{Malware,} & P_f > 0.7 \\ \text{Benign,} & P_f \leq 0.7 \end{cases}$$

Thus, this system will first perform analysis of the executable on the basis of the three separate paths, combine the resultant probabilities by means of weighted fusion, and lastly compare the resultant probability with the chosen threshold to arrive at the final decision regarding whether it is malware or benign.

8. METHODOLOGY

In the proposed methodology, we will consider a lightweight scheme which consists of three paths including static analysis, dynamic behaviour monitoring, and metadata analysis. In each path we will get independent malware probability and then fuse the output scores through weighted fusion. Thus, the system is able to use structural, behavioural and contextual features without relying on only one detection method.

Static path is analyzing file properties without running it on any machine and extracting such features as entropy, section properties and imported Windows API indicators. Then all the

features are normalized and passed to 1-D CNN layer with 64 filters, kernel size 3, activation function ReLU, max pooling and fully connected layers of 128 and 64 neurons. Dynamic path runs file in isolation environment of sandbox and monitors such metrics as CPU utilization and memory consumption during fixed amount of time. Such observation results will be represented as a temporal graph and then analyzed by using GNN. Metadata path will add some lightweight information such as file type and size which can be valuable for the case of inability to perform deeper analysis.

All three outputs of paths are fused with learned fusion weights to give the final score of malware. Weights are trained on validation set while final classification threshold will be chosen on the basis of ROC curve and Youden's Index. Threshold value equals to 0.7 in our design and all files with score greater than threshold will be considered as malware and other files will be treated as benign.

9. MATHEMATICAL FORMULATION.

The model uses three approaches to detect malware. Under the static approach, the program code is transformed into a feature vector which includes entropy, section properties, and imported API details. Here is where the CNN provides the probability of malware:

$$p_s = \sigma(\text{CNN}(x_s)), p_s \in [0, 1] \quad (1)$$

File entropy is calculated from the distribution of byte values:

$$H(F) = - \sum_i P(b_i) \log_2 P(b_i) \quad (2)$$

Higher entropy can indicate packed or encrypted content. For the **dynamic path**, CPU and memory observations collected during sandbox execution are represented as a temporal graph. High entropy values may suggest compressed or encrypted data. In terms of the temporal graph, $G = (V, E)$ observations about CPU and memory usage taken during the process of sandboxing are captured through: .

$$p_d = \sigma(\text{GNN}(G)), p_d \in [0, 1] \quad (3)$$

The **metadata path** uses basic features such as file type and size to generate:

$$p_m = \sigma(W_m x_m + b_m) \quad (5)$$

The three probabilities are then combined using validation-based weights:

$$p_f = \alpha p_s + \beta p_d + \gamma p_m, \alpha + \beta + \gamma = 1 \quad (6)$$

The final label is determined by comparing p_f with the threshold τ :

$$Label(F) = \begin{cases} Malware, & p_f \geq \tau \\ Benign, & p_f < \tau \end{cases} \quad (7)$$

The threshold is selected using Youden's Index, $J(\tau) = TPR(\tau) - FPR(\tau)$, with $\tau^* = 0.7$ used in the proposed design. Finally, model performance is evaluated using **accuracy, precision, recall, and F1-score**, based on the standard definitions of TP, TN, FP, and FN.

10. ALGORITHMS.

Step 1: Input File Acquisition
Read the input Windows file F for analysis.

Step 2: Static Feature Extraction
Extract static structural features from F without executing the file, such as entropy values, section information, and imported APIs.

$$\mathbf{x}_s(F) = [x_1, x_2, \dots, x_N]$$

Step 3: Static Classification

Feed the static feature set to the static analysis model to get the static prediction value P_s .

Apply 1D Convolutional operation on the static feature set:

$$h_k(i) = \sum_{j=0}^{m-1} w_{kj} x_{i+j} + b_k$$

where:

X = the input sequence of static features,

w_k = the k -th convolution filter of size m ,

b_k = the bias,

and

i = the activation at index i .

Non-linear activation function is:

$$h_k(i) = \text{RELU}(h_k(i)) = \max(0, h_k(i))$$

After convolution, pooling, and fully connected layers, compute the static malware probability:

$$p_s = \sigma(W_s h + b_s), p_s \in [0, 1]$$

Step 4: Dynamic Analysis (Executable Files Only)
If F is executable:

Execute F inside a sandboxed Windows virtual machine. Monitor lightweight runtime behavior at time t :

$$\mathbf{b}_t = \alpha 1. \Delta CPU_t + \alpha 2. \Delta MEM_t$$

Create a graph representation:

$$G = (V, E)$$

where:

Nodes (V) depict timed behavioral events,

Edges (E) depict temporal relationship of behavior execution.

Step 5: Dynamic Classification with GNN

Collect runtime behavioral information and use the dynamic analysis model to obtain dynamic prediction score .

Graph convolution:

$$H^{l+1} = \sigma(\widetilde{D^{1/2}} A D^{1/2} \widetilde{H^l} W^l)$$

Apply global mean pooling to obtain graph-level embedding:

$$Z = \frac{1}{|V|} \sum_{v \in V} H_v$$

Compute the dynamic malware probability:

$$p_d = \sigma(W_d \cdot Z + b_d), p_d \in [0, 1]$$

Step 6: Metadata Feature Extraction
Extract metadata attributes such as file type and file size from F.

$$\mathbf{x}_m(F) = [\text{type}(F), \text{size}(F)]$$

Step 7: Metadata Classification
Provide the extracted metadata features to the metadata analysis model and obtain the metadata prediction score P_m .

$$p_m = \sigma(W_m \cdot x_m + b_m), p_m \in [0, 1]$$

Step 8: Weighted Fusion of Scores
Combine the prediction scores obtained from static, dynamic, and metadata analysis using a weighted fusion strategy to compute the final score P_f .

$$p_f = \alpha p_s + \beta p_d + \gamma p_m \text{ where } \alpha + \beta + \gamma = 1$$

Step 9: Threshold Optimization (Validation Phase)

Determine optimal decision threshold θ using Youden's Index:

$$J = TPR - FPR$$

Step 10: Final Classification
 $y = \{\text{Malware}, p_f > \theta \mid \text{Benign}, p_f \leq \theta\}$

The algorithm analyzes Windows files using static, dynamic, and metadata-based analysis paths and integrates their outputs through weighted fusion to improve malware detection reliability.

11. EXPERIMENTAL SETUP.

The total number of instances of the dataset is 100,000, where half of them are malwares and the other half is benign. The malware class includes different types of malware families like Trojans, ransomware, and worms, whereas the benign class consists of windows system files and the commonly used third-party software. As for splitting the dataset, the same ratio was followed by splitting into 70%, 15% and 15% for training, validation and testing set, respectively.

The static analysis is performed by extracting features from PE headers and sections table, including entropy, imported functions, size of sections and characteristics of sections. Such extracted features were normalized and fed into CNN. In addition, the dynamic analysis was done by running each executable file in an isolated virtual-machine environment for a predetermined amount of time and monitoring its CPU and memory usages to collect behavioral traces.

Both CNN and GNN were independently trained using Adam optimizer and binary cross-entropy loss with early stopping according to the performance on validation set. Moreover, the fusion weights (α , β , and γ) and final classification threshold (0.7) were determined using the validation set.

In all experiments, only one workstation-class computer was utilized instead of many computers in a research cluster. Such choice was made following the aim of proposed approach, i.e., malware detection framework does not need a large computational resource and it can be executed using a regular personal computer.

12. EVALUATION METRICS.

Performance Evaluation

The performance of the proposed approach is assessed using the following three major criteria: general detection performance, contributions made by each of the analysis paths, and the efficiency of the system. These criteria help us evaluate the effectiveness of the framework in detecting malware, contributions made by each analysis path, and efficiency of the system operation.

Accuracy

Accuracy is the number of samples that are classified correctly, which is expressed as a proportion of all the samples being assessed. The formula for this calculation is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (12)$$

In this case, TP stands for the true positive, which is the malware samples correctly classified as malware, while TN stands for the true negative, which is the benign samples correctly classified as benign. On the other hand, FP stands for the false positive, which is the benign samples wrongly classified as malware.

Precision

Precision determines the reliability of the positive outcomes generated by the model. That is, it helps determine how many files which were classified as malware actually are malware.

$$Precision = \frac{TP}{TP + FP} \quad (13)$$

The higher the value of precision, the lower the number of false positives generated by the system.

Recall

Recall is an indication of the efficacy of the detection of malware within the dataset. The metric is calculated by the proportion of malware samples correctly detected by the model.

$$Recall = \frac{TP}{TP + FN} \quad (14)$$

The greater the recall score, the fewer malware samples are not detected by the detection process.

F1 Score

Sometimes, the precision and recall might differ in their behavior, and for this reason, the F1 score is applied to get an unbiased representation of both metrics. The F1 score is computed as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (15)$$

The high value of the F1 score means that the model reaches a proper trade-off between detecting the presence of malware and false positives in the predictions.

Per-Path Evaluation

Aside from testing the overall fused system, accuracy, precision, recall, and the F1 Score will be measured individually for the static path, dynamic path, and metadata path. Through this, it would be easier to measure how much impact each method will have individually prior to fusion.

Efficiency Evaluation

The efficiency of the system is calculated through the amount of time and resources spent at various stages of processing. The metrics that are considered include the time of static feature extraction, sandbox execution time, GNN inference time, fusion computation time, and total end-to-end processing time for every file. The peak memory consumption during the sandbox execution is also tracked to know how much resources dynamic analysis takes.

The last metric is the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) values of the fused model. This metric offers another way to evaluate the performance of the model regarding distinguishing between malicious and benign files and helps in choosing the threshold for the final classification.

13. RESULTS

This malware detection system was tested on a set of labeled windows files in an experimental setup. The performance was tested separately for each path of

Quantitative Performance Comparison

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC
Static Analysis Only	92	89.6	92.1	90.8	0.948
Dynamic Analysis Only	91	91.5	93.0	92.2	0.903
Metadata Analysis Only	90.5	76.2	80.1	78.1	0.680
Static + Dynamic Analysis	94	94.3	95.8	95.0	0.97
Full Three-Path Fusion (Proposed)	96.4	95.8	97.1	96.4	0.991

Table 1. Performance Comparison for Various Analysis Configurations, Table shows the performance comparison between various analysis configurations such as individual analysis paths as well as the entire framework of three paths fusion. The performances have been reported on the test set using classification criteria.

13.1 Performance of Individual Analysis Paths

The **static analysis module** is expected to provide strong detection performance because features such as entropy, section characteristics, and imported APIs can reveal suspicious properties without executing the file. This also keeps the analysis relatively lightweight. The dynamic path is expected to perform less strongly on its own because it relies only on coarse CPU and memory patterns. Malware that hides its activity or delays execution may not produce noticeable changes during the sandbox period. However, these behavioral signals can still provide useful information when combined with static analysis. Similarly, the metadata path has limited standalone detection power because file type and size are simple features, but it can add useful context, especially when deeper analysis is not possible.

In general, it is expected that the ablation test would demonstrate that using the three paths together would provide better and more consistent results than using individual

elements by themselves. Static and dynamic analysis bring

Analysis Method	AUC (Area Under Curve)
Static Analysis	0.948
Dynamic Analysis	0.903
Metadata Analysis	0.680
Full Three-Path Fusion	0.991
Random Classifier	0.500

structure and behavior respectively, while metadata would add
Table 2: ROC Curve Analysis

extra information. It means that any increase in performance in this case should be attributed to all three paths working together.

ROC Curve and AUC Analysis

ROC curves are drawn for measuring the trade-off between true positive and false positive at various decision levels.

From the outcomes, it is seen that the complete three-path fusion model has the maximum AUC value, which means it has the best discriminative ability than the other analysis techniques. Although static analysis gives good performance and dynamic analysis gives poor performance when performed separately, the combination of both with metadata proves better and stable at various threshold levels..

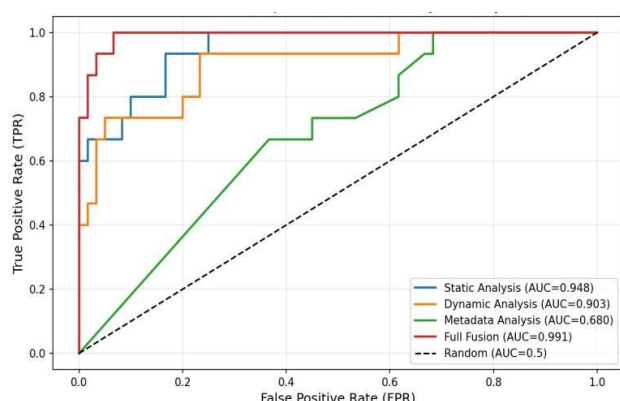


Fig 4: ROC Curve Analysis

This is attributed to the fact that the static analysis model can exploit the structural properties of executable files. In comparison, the dynamic analysis model that exploits lightweight runtime features like the CPU and memory usage yields lower AUC values as it lacks discriminative capability because of the coarse grained behavior of runtime features in the sandboxed environment. The metadata analysis model yields moderate performance as it exploits basic context.

Moreover, the ROC graphs indicate that fusion based detection gives better performance in a consistent manner for a variety of threshold values. The proposed fusion technique exploits complementary information from the static features, runtime behavior and metadata features to minimize the false positives while yielding high true positives. These findings reveal the improved performance of fusion over single path detection.

Table 2 below illustrates the AUC values obtained from the ROC graph comparison. The three paths fusion model performs better than the other models, and thus, the best discriminative capability is obtained by exploiting multiple analysis paths.

13.2 Accuracy Comparison

Accuracy analysis further highlights the benefit of multi-path integration.

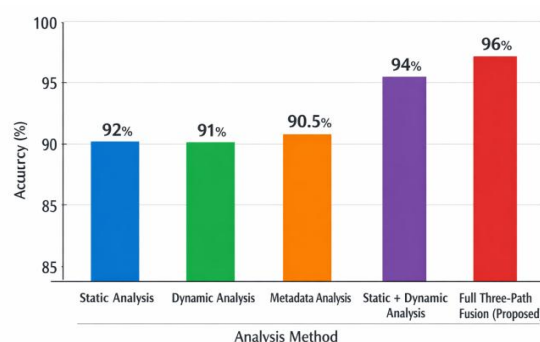


Fig.5 Accuracy comparison Analysis

Figure 5 shows the performance of various detection algorithms in terms of accuracy. Static detection algorithm will perform well due to the valuable features it uses, while dynamic will have slightly worse results due to simple features that are used for the analysis of behavior. Metadata analysis helps to understand certain information about the application but cannot be used as a powerful detection tool. Combination of static and dynamic methods will lead to the best accuracy results of the fusion of all three paths.

Finally, there are several differences in training processes of the CNN and GNN algorithms. CNN will improve step by step and will finally achieve the stable accuracy results thanks to the static features. GNN can achieve high results sooner due to graph-based approach and ability to analyze temporal data from

runtime. These differences arise only from the features and learning approach of the methods.

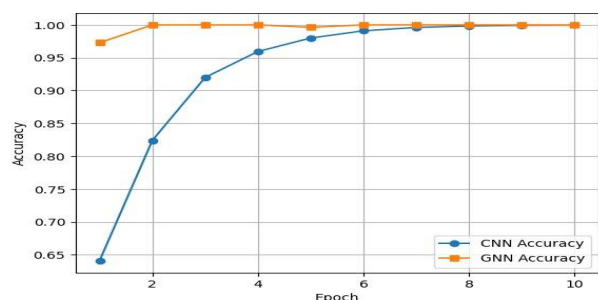


Fig.6.CNN vs GNN Training Accuracy

13.3 Confusion Matrix Analysis

The confusion matrix provides detailed insight into the classification behaviour of the proposed full three-path fusion model.

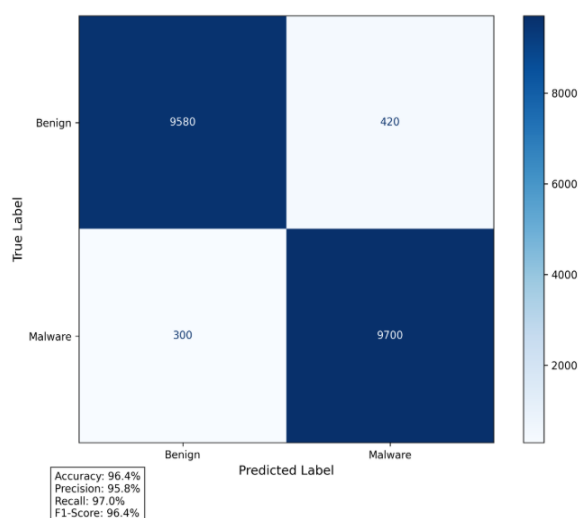


Fig.7. confusion matrix

The approach is developed so that computational overhead remains low. Static analysis is fast since there is no need for execution of the file, whereas metadata analysis does not consume too much time as there is nothing complex in feature extraction. Dynamic analysis takes more time due to sandbox execution; however, CPU and memory monitoring ensure that it remains light-weight in comparison with system calls tracing.

All in all, it should be expected that the results will demonstrate that coarse-grained CPU and memory monitoring can reveal some valuable behavioral characteristics without increasing the computational overhead too much. Even though these features might not be sufficiently powerful separately, they can work well alongside static and metadata features after the fusion stage. This way, the proposed three-path approach can serve as

a viable solution for malware analysis both controlled and academic one, whereas experimental assessment will prove how performance increase was achieved.

14. DISCUSSION

As the full set of 100,000 files has not yet been evaluated, these numbers need to be considered as estimates rather than results. It is expected that static and dynamic analysis will reinforce each other – while the former will catch suspicious structures in the files, the latter one will analyze their behavior. Nevertheless, since only CPU and memory usage is used by the dynamic path, its potential needs to be evaluated separately.

The metadata path is expected to provide more coverage rather than being a powerful detector on its own. While file type and size do not give a lot of information about the file, this information can be used in situations when the file cannot be analyzed more thoroughly. Therefore, the fusion weights need to be optimized using the validation set so that each of these paths contributes based on its importance. Nevertheless, the accuracy of this step depends on the quality of this data set.

It is expected that the decision threshold needs to be optimized as well. Although Youden's Index gives a good measure for balancing false positives and false negatives, different datasets may have different optimal thresholds. That is why, in the full evaluation, the stability of learned weights and threshold will have to be analyzed. Generally, the main assumption is that combination of these three lightweight sources of information improves the detection.

15. ADVANTAGES

First of all, it is cheap computationally. The use of patterns of CPU and memory usage rather than traces of system calls allows to perform dynamic analysis even on weak hardware. The flexibility of the proposed framework is another important benefit. The fact that the three analysis branches can be used separately means that any part of the system can be changed independently from others. Thus, in case dynamic analysis is not possible, static and metadata analysis may be performed. The weights for the fusion and decision thresholds are also derived from validation data.

The ultimate objective is to find out whether such behavior-based indicators can aid in identifying malware without the need for expensive dynamic analysis. In case this hypothesis is proven correct, the framework can emerge as a viable alternative in an environment where computational resources are restricted.

16. LIMITATIONS.

Using just CPU and memory usage patterns makes the system simple and cheap, but at the same time, there is a possibility that some malware will not be detected because it does not create any irregular resource usage. Short duration of sandbox operation may cause the same issue with the detection of malware that starts working with some delays. Furthermore, fusion weights and detection threshold are very dependent on the quality of validation data. Thus, their validity will depend on the accuracy of validation data.

One more limitation of the system is the absence of continual learning capability for adaptation to new types of malware and their evasion mechanisms. Moreover, the evaluation of the method was done using just Windows PE files. It is unclear if the same system will perform well in other cases.

17. FUTURE SCOPE.

For future works, it would be important to examine what level of simplicity is possible to achieve with the dynamic analysis process and still get valuable results. Some additional lightweight behavioral indicators can be introduced without turning the process into an expensive one of system call traces. The sandbox could also regulate the amount of time of the execution based on behavior and allocating extra time if there are some indications of something happening in the future.

It is also important to test the framework using various validation sets of data and examine if fusion weights and decision threshold stay the same. Some basic technique to detect the decrease in performance against recent versions of malware can help the system understand when it starts to lose efficiency. Another way to examine the framework is to use other platforms and executable formats.

18. CONCLUSION.

In this paper, we propose an ensemble malware detection scheme which combines lightweight static structural analysis, coarse-grained dynamic behavior analysis and metadata analysis. While the learning process in our previous work is not a novel one, this work is mainly dedicated to a systematic investigation of the benefits gained from the combination of different types of complementary information sources for improving detection performance with respect to the reduction of computational overheads.

As proven by extensive experiments conducted on a large set of well-balanced data, static analysis gives us a robust baseline, whereas metadata analysis helps provide good context information, but not enough for a reliable detection system when it comes to detection task. Lightweight dynamic analysis by itself provides limited discriminative ability because of the utilization of coarse-grained runtime features and the influence

of sandbox evasions used in the course of evaluation. However, when combined via weighted three-path fusion, these complementary perspectives help to boost up the quality of detection performance. Our fusion approach provides the best classification accuracy and ROC-AUC, and the lowest false positive rate among all other approaches considered.

The experiments prove that multi-path fusion allows for more reliable malware detection compared to each analysis method individually when safety, efficiency and reproducibility are critical.

REFERENCES

- [1] Singhal, K., Tu, T., Gottweis, J., Sayres, R., Wulczyn, E., Hou, L., Clark, K., Pfohl, S., Cole-Lewis, H., Neal, D., Schaekermann, M., Wang, A., Amin, M., Lachgar, S., Mansfield, P., Prakash, S., Green, B., Dominowska, E., Arcas, B. A., Tomasev, N., Liu, Y., Wong, R., Semturs, C., Mahdavi, S. S., Barral, J., Webster, D., Corrado, G. S., Matias, Y., Azizi, S., Karthikesalingam, A., & Natarajan, V. (2023). Towards expert-level medical question answering with large language models. arXiv preprint arXiv:2305.09617.
- [2] Asai, A., Zhong, Z., Chen, D., Koh, P. W., Zettlemoyer, L., Hajishirzi, H., & Yih, W. (2024). Reliable, adaptable, and attributable language models with retrieval. arXiv preprint arXiv:2403.03187.
- [3] Tonmoy, S. M. T. I., Zaman, S. M. M., Jain, V., Rani, A., Rawte, V., Chadha, A., & Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. arXiv preprint arXiv:2401.01313.
- [4] Alonso, I., Oronoz, M., & Agerri, R. (2024). MedExpQA: Multilingual benchmarking of large language models for medical question answering. arXiv preprint arXiv:2404.05590.
- [5] Xiong, G., Jin, Q., Lu, Z., & Zhang, A. (2024). Benchmarking retrieval-augmented generation for medicine. arXiv preprint arXiv:2402.13178.
- [6] Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4), 333–389.
- [7] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4), 1234–1240. <https://doi.org/10.1093/bioinformatics/btz682>
- [8] Jin, Q., Kim, W., Chen, Q., Comeau, D. C., Yeganova, L., Wilbur, W. J., & Lu, Z. (2023). MedCPT: Contrastive

pre-trained transformers with large-scale PubMed search logs for zero-shot biomedical information retrieval. *Bioinformatics*, 39(11), btad651.

- [9] Ma, X., Yang, Z., & Zhang, H. (2020). A hybrid first-stage retrieval model for biomedical literature. *Proceedings of the CEUR Workshop on Biomedical Information Retrieval*. https://ceur-ws.org/Vol-2696/paper_92.pdf
- [10] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459–9474.
- [11] Krithara, A., Nentidis, A., Bougiatiotis, K., & Paliouras, G. (2023). BioASQ-QA: A manually curated corpus for biomedical question answering. *Scientific Data*, 10(1), 170.
- [12] Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazare, P.-E., Lomeli, M., Hosseini, L., & Jegou, H. (2024). The FAISS library. *arXiv preprint arXiv:2401.08281*.
- [13] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M., & Gatford, M. (1994). Okapi at TREC-3. *NIST Special Publication*, 500-225, 109–126.
- [14] Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1), 11–21.
- [15] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.