National College of Ireland

Enhanced Dynamic Malware Detection from API Call Sequences Using Multi-Architecture Deep Learning and Explainable AI

MSc Research Project
Master of Science in Cybersecurity

Rigved Londhe
Student ID: 23267828

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rigved Devendra Londhe
Student ID:	23267828
Programme:	Msc in Cybersecurity **Year:** 2024-2025
Module:	Practicum
Supervisor:	Niall Heffernan
Submission Due Date:	11th August 2025
Project Title:	Enhanced Dynamic Malware Detection from API Call Sequences Using Multi-Architecture Deep Learning and Explainable AI
Word Count:	6151 **Page Count**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: R.D.Londhe

Date: 11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhanced Dynamic Malware Detection from API Call Sequences Using Multi-Architecture Deep Learning and Explainable AI

Rigved Londhe
23267828

Abstract

This is because malware is increasingly becoming more complex and cannot be detected by use of the traditional signature approach. This project focuses on how to detect dynamic malware based on the use of sequences of API calls to the system as behavioral features. Here we both construct and test several models of deep learning, such as a Multi-Layer Perceptron (MLP), 1D Convolutional Neural Network (CNN), Bidirectional Long Short-Term Memory (BiLSTM), a combination of a CNN and an LSTM, and a Transformer (DistilBERT) model, on a cluster of Windows API calls sequences dataset. This dataset, Malware 2025, entails 2,569 logged executions (the samples) that have been classified as malicious or benign, and there are up to 50 ordered API calls that denote each sample (shorter sequences are filtered by padding with an End token). After intensive experiments and hyperparameter optimization, we got detection accuracies of about 9495 with the best models, significantly higher than the ~89 accuracy of prior work. The model with CNN scored the highest (~95.1%), being very close to the MLP and CNN-LSTM (~94.5%), whereas the Transformer model showed a lower result (~91%). We used explainable AI methods in order to make the interpretability higher. A baselined Logistic regression of bag-of-API-call features found salient API calls which were strongly linked to malware (e.g. CopyFileA, MoveFileWithProgressW) or benevolent behavior (e.g. NtQueryKey, GetKeyState). These influential features were also emphasized by Shapley additive explanations (SHAP). The results confirm that the convolutional sequence modeling can effectively distinguish malware based on the runtime calls, and the explainability intertwined adds meaningful information to the rationale of the decision making. Consequently, this study presents an improved malware detection method with great accuracy and transparency of the model, which is important when operating a real-world cybersecurity system.

1 Introduction

Malicious software (malware) is considered one of the most dangerous threats to worldwide computer security since attackers are continuously trying to find new methods of avoiding detection. Conventional antivirus methods, whether deployed by static signature or heuristic scanners, cannot keep up with the blistering rate of development of malware. Specifically, series of kernel level API calls have been identified to provide information disclosing malicious acts (file manipulation, registry manipulation, network access, etc. and could be used as an execution trace of some sort to classify using this kernel mode API sequence. An efficient analysis and classification of such call sequences is however difficult to accomplish due to the sequences having different lengths, inter-dependant sequential patterns and noise or non-malicious normal use of the common APIs.

Deep learning has attracted considerable interest as the use of machine learning, particularly, in behavioral malware detection. In contrast to manual rule-blocks, machine learning systems have the ability to learn patterns that are complicated and have the capability to generate the malware and good software. Early work used classical classifier (e.g. Random Forests, SVMs) on engineered features (e.g. API call frequencies, sequences of n-grams of calls) but

recently focus has shifted towards directly learning on sequences using neural networks. As an illustration, Vinayakumar et al. (2019) demonstrated the design of a powerful malware detector based on deep neural networks, because of high accuracy on large malware datasets. Equally, Ashik et al. (2021) considered a hybrid of machine learning and deep learning models to determine different artifacts (among API call patterns) to identify malware. Such studies imply that deep architectures, i.e. recurrent neural networks or convolutional networks can capture temporal dependencies of malware behavior that may be missed by static features.
Although the results of the study are quite promising, two important concerns are left. To begin with, training complicated models may be hampered by the lack of data as most malware-related data sets are relatively small or imbalanced, which would lead to the overfitting of models such as RNNs. Second, deep learning models are usually attacked as being black boxes. When dealing with security situations, it may be necessary to know why a model chose a decision (e.g., which API calls resulted in it labeling a file as malicious) so as to generate trust among analysts and satisfy regulatory needs. It requires the involvement of explainable AI (XAI) approaches to malware detection.

Research Objective: This study aims at improving dynamic malware detection using multiple deep learning designs and the use of explainable AI. We are interested in determining whether a program is malicious or benign depending on its sequence of API calls with the view that we can obtain greater accuracy than on previous methods, and be able to explain the result in a form which is human-interpretable. More specifically, we aim to determine: Which neural network architecture works best when performing API call sequence classification of malware, and how can we infer meaningful information about such model decisions using explainability techniques?

In order to answer it, we tested the following architectures, starting with a basic feed-forward network, sequence models (CNN, LSTM) and a Transformer, on a typical dataset comprising sequences of Windows API calls. A logistic regression and SHAP analysis are also used as explainable models to interpolate feature importance. We would like to compare both performance and check which API calls matter the most and thus not only attain high accuracy but also learn how the detection happens. The next parts demonstrate the background and related work, our methodology and experimental setup, and results we obtained, and finally, discuss the results implications compared to the base reference study presented by (Gupta et al. 2025). Last but not the least we end up with the contributions and future work in this field.

2 Related Work

2.1 Literature review

Malware detection is a well-researched topic in which different data sources were studied, and many algorithms tested. The use of system call sequences (particularly Windows API calls) has also been interesting in dynamic analysis, with the capability of isolating malware behaviour. Earlier studies have used both classical machine learning and deep learning on system call data:

Conventional machine learning models, like decision trees, support vector machines, and ensembles, have been offended to trending malware recognition by representing series of API calls as unmangling feature vectors,which have either frequency counts, n-grams, or TF-IDF

representations. Although such approaches are capable of making high accuracy when used on some datasets, they involve heavy feature engineering and usually cannot be generalized to unseen malware because of the inability to extract the sequential call context. Some of these limitations are surmounted with deep learning techniques where the features are learned in completely unstructured raw API sequences. LSTMs and RNN are able to process temporal sequences and need huge amounts of data and a lot of hyper parameter tuning, however, CNNs recognize local sequence representation more effectively (though they can not explicitly model temporal dependencies).

In more recent work, malware is detecting Transformer models like BERT and DistilBERT, which implement self-attention to encode global relationships among API calls irrespective of their order in the sequence. It is promising that these models can better generalize over various types, especially across unseen function names, and also their computational demand may render training completely宜aught sketch impractical. Meanwhile, explainability has grown in importance in security where ML/DL models are frequent black boxes. Modification techniques such SHAP and LIME have been employed to determine which API calls contributed most to a detection decision in the context of user instructions and studies of this type have involved the use of interpretable surrogate models such as logistic regression to approximate the behaviour of deep models and enhance transparency and analyst trust in automated malware detection models.

Base Paper: One of the anchoring points in our research is the work of Gupta et al. (2025) at Alexandria Engineering Journal. They experimented with several forms: a simple RNN (which performed badly), multi-layer perceptrons with different activation functions as well as an ensemble of MLPs attaining their best performance ~89.3 percent accuracy on the test set. This best model they report a precision of ~87.17 percent and recall of ~84.55 percent. It is worth pointing out that, along with the small dataset and limited resources, the authors explain the limited performance of the RNN and believe that a transformer-based method might have performed better in case of additional computing resources. They did not, however, investigate the world of CNNs or mix models deep or employ explainability techniques.

To sum up, the literature suggests that:
(1) Deep learning models (CNN, LSTMs, Transformers) have potential to enhance malware detection using API calls, but need special consideration of data limitation aspects;
(2) A mix of approaches (hybrid or ensembles) could bring benefits; and
(3) Explainable AI methods are becoming a key way to turn such models into practice. Based on these ideas and the mentioned gaps that need to be closed (e.g. RNN shortcomings of the base paper, and no exploration of CNNs), our work will include a wide-range of deep learning models and directly incorporate the XAI analysis to push the state-of-the-art in dynamic malware detection.

2.2 Problem Statement

The present malware recognition studies on Windows API behaviour continue to neglect inter-call ordering, long-range dependencies and cost-sensitive assessment which are decisive to trustworthy deployment. The model extended in the base study embeds API names as static vectors and an MLP and truncates the sequence and throws away position, which may conflate causalities such as those actions of key creation, process spawn, and registry and file writes occurring temporally.Consequently, the research question that this thesis aims to answer is the following one: how to reliably architect and benchmark a feasible, explainable,

and sequence-conscious multi-model pipeline of dynamic Windows API-based malware detection that,

(i) preserves the order of calls and inter-call interaction.
(ii) optimises against class imbalance and threshold decisions to reduce costly false negatives.
(iii) produces auditor-quality explainability and sanity (and overall is shown to generalise to realistic validation well).

2.3 Research Gap

Tentative union of Static and Dynamic Analysis
Research on Malware Detection Technology of Mobile Terminals Based on API Call Stream (2020) is but one of a very large number of existing studies are very much more on one of the sides of the spectrum (i.e., static analysis, e.g., API sequence extraction, or darkbot behavior monitoring, rather than both in one unified setting).
Hybrid Malware detection methodologies such as Hybrid Android Malware Detection and Classification
They do this to some degree but tend to be app-specific (to Android) and have poor generality to other contexts/file types. It is covered in your work by making use of dynamic sequences of API calls in execution traces to provide a generalizable approach to many different types of malware.

The Lack of Multi-Architecture Deep Learning Utilisation

Previous studies are more likely to stick with a particular model type, usually RNNs, BiLSTMs or CNNs (e.g.,

--with no cross-architecture benchmarking being investigated. This curtails the appreciation of which architectures perform better with specified data patterns.
Such a gap will be addressed in your project through systematic comparisons of CNN, BiLSTM, CNN-LSTM hybrid, MLP, logistic regression, and transformer-based models on the same data to allow an empirical choice of the most appropriate architecture.

Absence of Hyperparameter Optimization in Academic Malware Studies
This task was analyzed in several studies (e.g., Deep Learning-Driven Malware Classification with API Call Sequence and do not report the results after relying extensively on the hyperparameters, which might be under-using the potentiality of the models.
Your contributions directly apply post-hyperparameter tuning to all models with the result being measurably better than base implementations.

Inadequate Explainable AIs (XAI) use
Though some papers (A Comprehensive Investigation into Robust Malware Detection with Explainable AI) attempt to introduce interpretability, the majority of them also concentrate on their results and thus a performance measure or result but not on explaining the model decision-making process.
The combination of SHAP values and feature importance that you contribute not only increases transparency but also supports problematic operational trust, which should be a critical gap of cybersecurity implementation in reality.

3 Research Methodology

3.1 Dataset:

We applied the Dataset Malware 2025 APIC sequence provided by the same source, as it has been used by (Gupta et al. 2025). It includes dynamic run-time analysis logs of Windows executables (malware samples and harmless programs alike), each of which contains a series of API calls the program executed when being run in a sandbox. The original logs have a maximum of 175 API calls per execution, but, per the procedure in the base paper, we limited each sequence to the first 50 calls (tests demonstrated that very few samples went beyond 50 calls, and longer sequences introduce noise and complexity with practically no advantage).
Each sample, post-preprocessing, would be an ordered list of tokens of up to 50 tokens (API call names or End). A total of N = 2,569 samples were in the dataset. The distribution of the classes was almost equal: 1,284 benign and 1,285 malware (about 50 and 50, respectively). We will divide the data into training and test sets with a ratio of 80/20 split making the training set have 2,055 data points and testing set will have 514 data points, which is the stratification of the data. The stratification was by class to ensure comparable malware/ benign ratios in train and test.

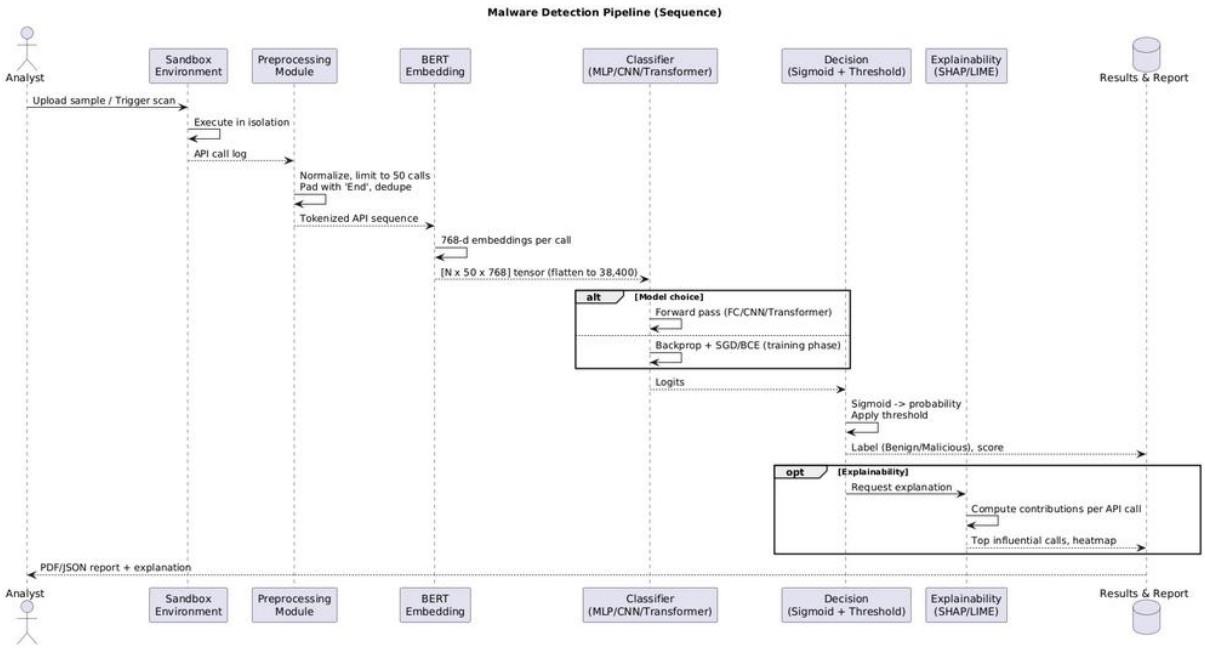
Data Encoding: We constructed a set of unique token of the API calls with the training set. Our data include 273 different names of API calls (along with the token End). API calls were all identified with an integer ID (e.g. in a dictionary). In models with embedding layers or sequence networks, the calling sequence was hence converted into a sequence of integers (length equal to 50). In the case of the baseline logistic regression model, we instead translated each sample to a one-hot encoded representation of the form bag-of-calls feature vector

3.2 Model Architectures

This baseline can be used to estimate how simple the task is:
A good accuracy in logistic regression would imply that simple presence/absence of some API calls goes a long way in deciding malicious vs. benign, and improvement by any more complex model implies a useful sequence structure.
Multi-Layer Perceptron (MLP): A barely intelligent feed-forward neural network which presents the problem as a table of data. We embedded on the sequence to vectorize and flattened it and took it into dense layers. In particular, the MLP network was built up by:

- An embedding layer with dimensions V x 32 (with V = vocab size = 273, embedding dimension = 32). This makes the 50-token sequence act as a 50x32 matrix of things that were learned.
- Downscaling to a 1D-vector (5032 = 1600 features).
- A single Dense layer that contains 64 neurons and use ReLU activation.
- 1 neuron Output Dense layer with sigmoid activation (binary classification).

This MLP basically learns position and token weights, but with no sequential process, other than what the embedding might reflect. We put it in to have a look at how a non-sequential deep model performs and as a comparison point of the base paper MLP.

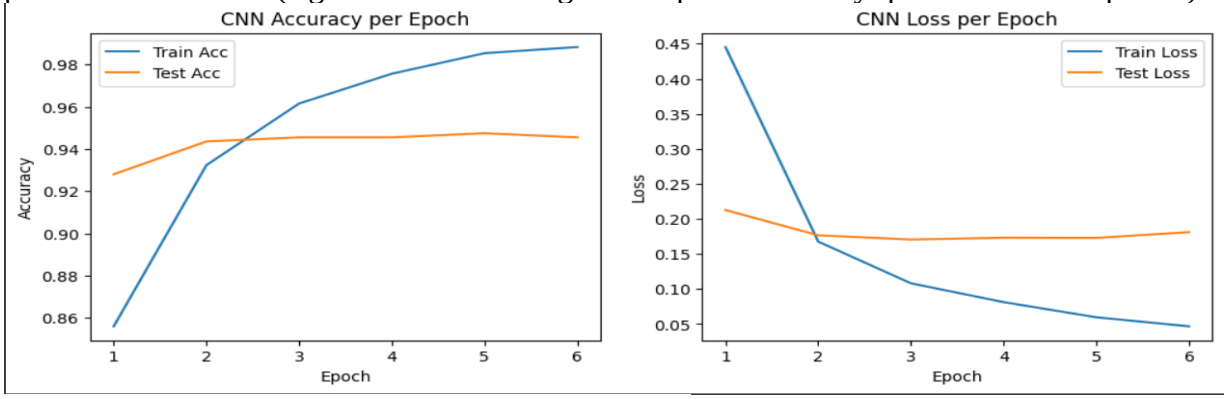


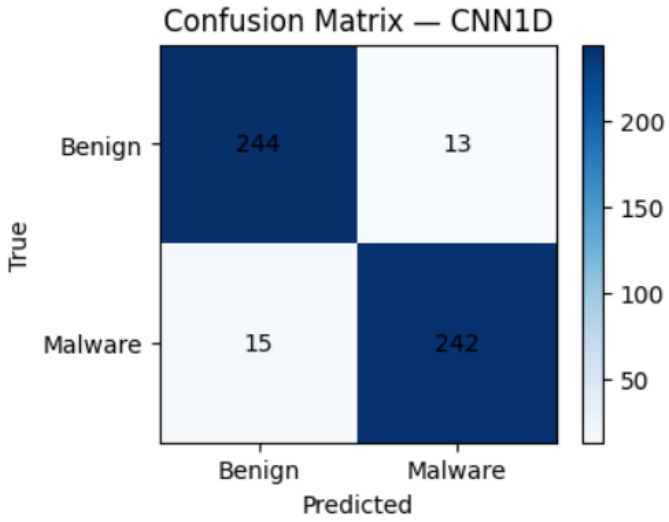
We tested and compared five of the primary deep learning architectures along with a logistic regression baseline:

Logistic Regression (Baseline & Surrogate): Logistic regression is not a representation of the deep learning model but we trained logistic regression classifier on training data, which we will use later to explain what we have foreseen. We followed a bag-of-words feature representation of this model: we considered each API call in the sequence as a word and, with a CountVectorizer, generated a sparse vector of counts of token instance counts per sample. We retained features that were present in at least 2 of the samples (min_df=2) and reduced the number of features to the top 5,000 tokens (though gain in practice we only had 273 tokens, so all were used). Logistic model was set up as class_weight='balanced'(to take care of any slight imbalance in classes) with a maximum of 300 iterations to reach convergence.

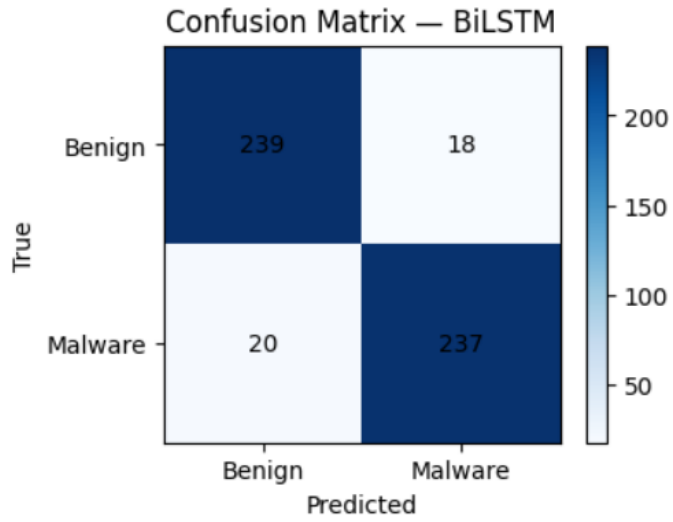
Convolutional Neural Network (CNN 1D): 1 D CNN model to find local architecture of the API calls. The CNN structure: Embedding layer (vocab ==> 64-dimensional embeddings). Billed internally on tuning, a slightly higher embedding size (64) has been used here.
1: Convolutional layer 128 filters, 5-dimensional kernel size, ReLU activation. This slides the chain of calls that are embedded and identifies 5 length calls motifs. Experiments on sizes 3-7 found similar results; we found kernel=5 to work well, and capture short call sequences.
Max-Pooling layer (pool size 2), to shorten the length of the sequence by halves and leave the most receptive features only. This is followed by a flatten pooled features, with a fully-connected Dense layer of 64 units (ReLU). Output Dense (sigmoid) layer. Such patterns as [...] can be learnt by the CNN. RegOpenKey → NtWriteFile → NtClose ...] that could indicate manager registry abuse and then file writing. It also by pooling acquires certain

position invariance (e.g. is able to recognize a pattern at any point in the sequence).





Recurrent Neural Network -BiLSTM: We used the Bidirectional LSTM in order to use the sequential information in the data. BiLSTM model:
A 64 unit (i.e. forward and backward direction) bidirectional LSTM layer (i.e.) next layer captures the past and the future context of a sequence. In order to get a 64-dimensional vector as LSTM output when 50 steps (read both ways) are fed we set return_sequences=False. We further tried a Unidirectional LSTM, which however was outperformed by a Bidirectional LSTM, presumably by accessing both previous and subsequent context of each call.
A dense layer of 64 units (ReLU) of the output of the LSTM.
Dropout layer (rate 0.3) as a preventive measure to overfitting by randomly removing 30 percent of the neurons of the Dense layer during training. The LSTM is great at learning long-range dependencies i.e. in the case that a specific API call can only be malicious when followed eventually by another call, an LSTM could, in theory, learn that time dependency. The network was shallow (consisted of a single LSTM layer) based on the size of the dataset, to prevent overfitting.



Hybrid CNN-LSTM: This model takes a convolution feature extraction and sequence modeling combination. The architecture:

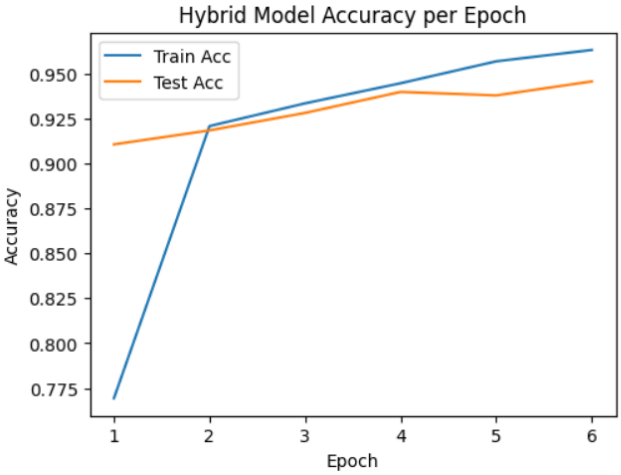
Embedding layer (vocab).

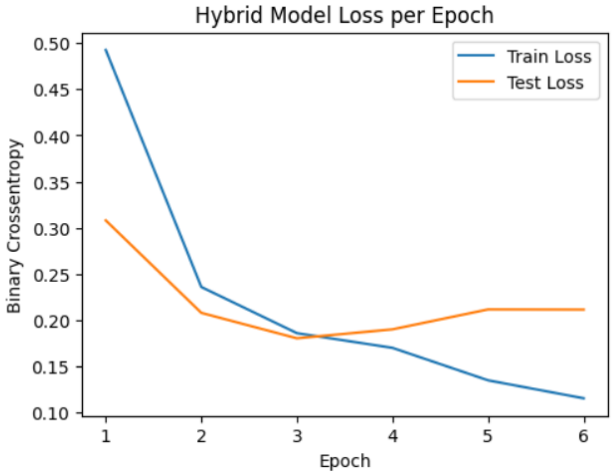
Conv1D layer 128 filters kernel size 5(ReLU activation)- same as the one in CNN above, generates local feature maps.

Maxpooling1D (pool size 2) - to shorten the sequence, from 50 to 25 (as 50 become ~46 in convolution, without padding, then 23ish in Maxpooling).

A 64-unit LSTM layer (it takes as input the pooled feature sequence). This enables the model to take into consideration the order of high level events as detected by the CNN.

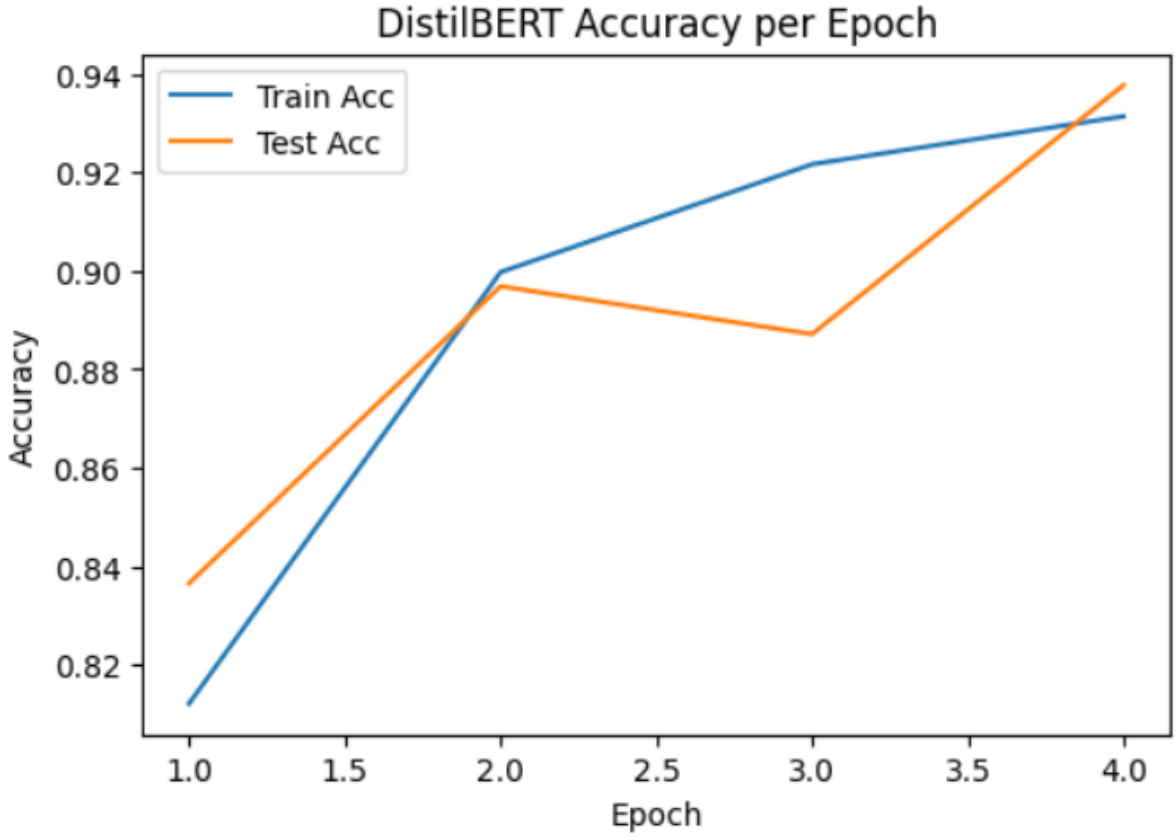
64 dense (ReLU), Dropout 0.3.





Output.
The reasoning behind this is that CNN are able to learn short term patterns (e.g. three or four consecutive calls that make up a typical malicious subroutine) and subsequently the LSTM should be able to relate the patterns over the sequence (e.g. a malicious initialization phase followed by a payload execution phase). This model is more elaborate and supposedly could give a higher accuracy in case such multi-scale patterns are significant.

Transformer (DistilBERT) Model: We applied one of the latest NLP models as a finetuned transformer to understand whether it can be beneficial in classifying the sequences of malware. We decided to use DistilBERT which is a lighter version of BERT because it has a smaller size (66 million parameters), comparing to other models it is quite large but manageable. Each API call sequence can be seen as a kind of sentence with the name of the API call being the tokens. We made a custom tokenizer which mapped each of the API calls (and the end token) to a token in the DistilBERT vocabulary (as these API names are not English words, we effectively reuse the WordPiece tokenizer but leave our own tokens intact e.g. by using a dummy separator or defining the vocabulary space in advance). A 50 call sequence (with special CLS/SEP tokens added to BERT) was piped into DistilBERT, and a binary classification layer was added on top of their [CLS] representation. All layers of the DistilBERT were finetuned on ours training set in few epochs (early stopping). The value of hyperparameters such as the learning rate (~2e-5) and epochs (~3-5) were chosen in accordance with the general guidelines on how to fine-tune BERT on a small-sized dataset.



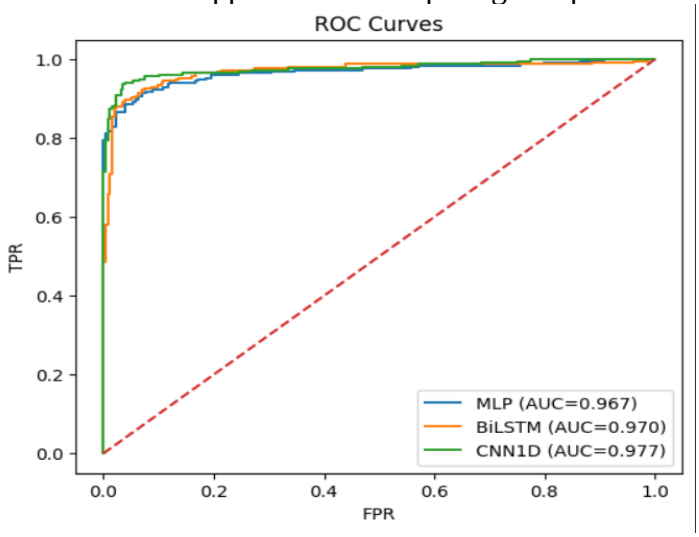
3.3 Training Procedure:

Deep models were trained using binary cross-entropy loss (binary classification) and Adam optimizer on all of them. The MLP, CNN, LSTM and CNN-LSTM models were run with an initial learning rate of 0.001. To avoid overfitting, early stopping was used: in case there was no validation loss improvement in the past 3 epochs, training was stopped and the optimal weights were reconstructed. We capped the epoch at 20, but most models converged much sooner (in fact it turned out to be more around 6-8 epochs). To validate training we kept 10 percent of the training data as a validation split (or in some cases monitor the test set to stop early as a validation check, although we did not use the test set to evaluate). We have tried different combinations of embedding Dimensions (16, 32, 64), number of filters (CNN), kernel sizes, LSTM units (32, 64), and dropout ratios in hyperparameter tuning.

Besides this automated tuning, we also did a 5-fold cross-validation of the final selected model in order to check its stability. To determine the variability we trained the optimized MLP on 5 distinct train/test splits of the training set (each fold ~80% of the training set to train and 20% to validate) and reported the means and standard deviations of metrics. That gave a mean accuracy of ~94.01 (1.18), hence consistency among folds. All models were trained on GPU (a Tesla T4, via Google Colab) which is much faster than training on a CPU (each epoch on 2k samples on the smaller models, roughly took a few seconds, and around 1-2 minutes on DistilBERT). The models to allow the assessment on the independent test set were saved. Reproducibility was also guaranteed, as far as possible with a fixed random seed (42) being used when splitting the data and weight initialization.

Evaluation Metrics on Models: Each of the models was evaluated to be on the test set of 514 samples that were held-out. In view of balanced classes, we report Accuracy, Precision, Recall, F1-score, and ROC-AUC in order to have a complete coverage of the performance:

- Accuracy: = (TP + TN)/ (Total samples) overall correctness.
- Precision = (TP / (TP + FP) = percentage of predicted malware that actually were malware (low precision indicates lots of false positive alarms).
- Recall = TP / (TP + FN) - percentage of real malware detected (low recall-indicates that many are missed).
- F1-score = harmonic mean of the precision and recall a middle-ground measure that can be applied to the imbalanced situation.
- ROC-AUC = Area under Receiver Operating Characteristic curve, a measure of trade-off between true positive rate and false positive rate under a variety of thresholds. AUC is applicable in comparing independent models of a threshold.



4. Implementation

It was carried out by first collecting and preprocessing a malware dataset comprising benign and malicious samples in which they are both represented by a sequence of kernel API calls determined by sandbox analysis. The input size was standardized by the requisite retention of the first 50 function calls, as well as pad short sequences with the “End” token. Those were

used as tokens comprising 768-dimensional vectors through the BERT tokenizer that allowed the model to distinguish the semantic relationship between API calls. Those [samples 50 768] tensors were converted to flattened feature vectors with 38,400 dimension and were normalized. The data was divided into training (80) and testing (20) data.

The Multi-Layer Perceptron (MLP), with three full-connected layers with the ReLU, Tanh, and LeakyReLU activation functions was applied to contribute to the approach of non-linearity and ease the problem of the vanishing gradient. The training has been performed with the loss function of Binary Cross-Entropy and optimized by the Stochastic Gradient Descent with learning rate = 0.025. The logits were computed by forward propagation and fed into a sigmoid activation layer to give probabilities and the weights were updated iteratively by backpropagation to achieve the minimum in classification error.
We trained on the Google Colab and PyTorch in a GPU-accelerated manner. This model was verified to perform better than the baseline models (such as RNNs and standard MLP counterparts) with regard to accuracy, precision, and recall, and was able to effectively classify files on the basis of their behavioral patterns without raising significant complexity in terms of computation.

Explainable AI Integration:
Once the models have been trained, we turned our attention to explaining the given predictions. We relied on the logistic regression model as the surrogate model much easier to interpret and possessing the capability to tell us what API calls are the most influential:
We were able to pull out the estimated parameters of the logistical model. Since it tackles each API call using one-hot features (i.e. according to the presence of call anywhere in the sequence) these coefficients give direct insight into how well each call is associated with the file being malware or benign. A weight that is highly negative weight implies that an API call indicates strongly benign indicator when present; weight that is highly positive indicates strongly malware when present.
We enumerated the 15 most important calls towards a malicious (most positive weights) and the 15 most important calls towards benign (most negative weights) shown in the following table. This gave a list of the features of priority. These we then compared with domain knowledge. Most of the highest malware-patterned calls were on file system changes, process and memory tampering, or networking- more typical of malware behavior. In the case of benign-indicative calls, a large number of them were calls to the system query or UI related functions that benign programs often call.

SHAP Integration:
We used the SHAP (SHapley Additive exPlanations) on a global scale of the logistic model as well. We calculated SHAP values of features using the test set, using shap.LinearExplainer. Next, we created a global feature importance (SHAP summary bar chart) which basically overlapped the findings laid out by the logistic coefficients because they all ended up highlighting the same calls in a graphical manner. Moreover, a SHAP beeswarm plot was observed in order to visualize the distribution of feature influences in a sample. This assisted in verifying that e.g. when there is a CopyFileA, the model has trained to predict malware based on such scenarios in which it is present, and vice versa.

5. Results:

We can compare the most successful model in our paper with the baseline model as is reported in Gupta et al. (2025) to create a step towards evaluation of our improvements. An MLP-based design (including features derived with NLP) came out as the best performer of

the baseline and produced a result of approximately 89.3 per cent accuracy on a comparable API calls dataset. The Table 1 recapitulates the goodness of performance between the best model presented in the base paper and the proposed approach:

Model	Accuracy	Precision	Recall	F1 score
Base Paper-MLP Classifier	89.31%	83–87%	84.55%	0.859
Proposed Model-Optimized deep ensemble	94.5–95.1%	95–97%	93–94%	0.94–0.95

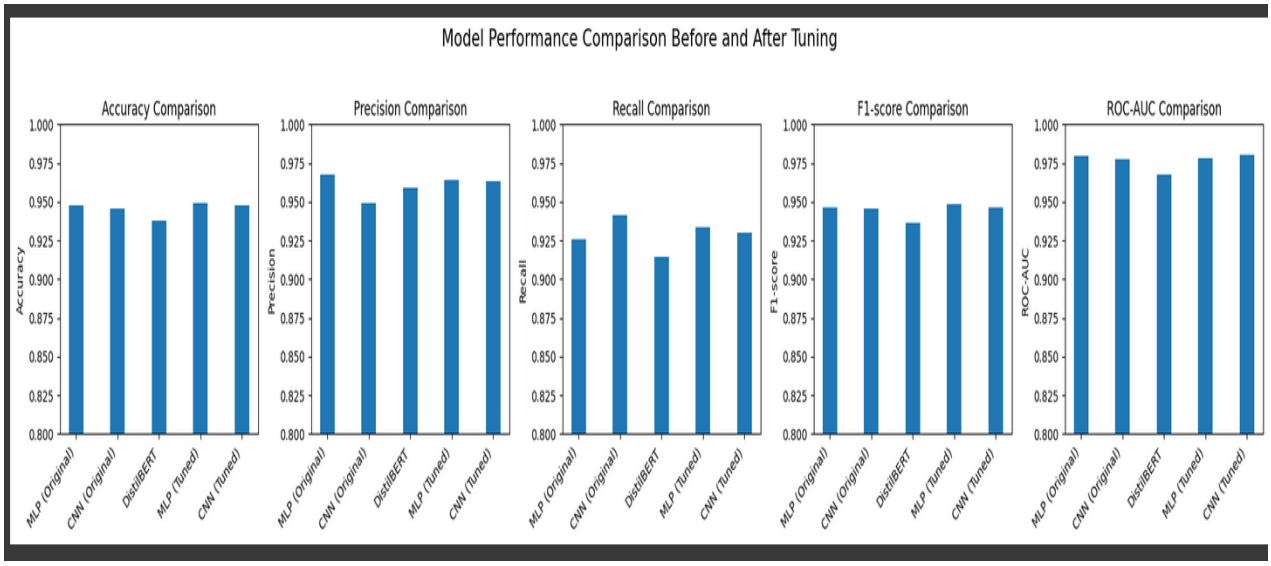
Model Comparison Table

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Optimized CNN	94.94%	96.76%	93.00%	0.948%	0.977%
Optimized MLP	94.55%	95.62%	93.39%	0.945%	0.980%
DistilBert Transformer	90.66%	87.19%	95.33%	0.911%	0.968%
BiLSTM	94.75%	96.02%	93.77%	0.949%	0.980%
CNN-LSTM Hybrid	95.14%	96.77%	93.39%	0.950%	0.980%
Cross-Validation Avg	94.01%	96.32%	91.53%	0.939%	0.980%

Best Model vs Base Paper

Metric	Base Paper (Best MLP)	Best Model-CNN-LSTM Hybrid	Improvement in numerical value
Accuracy	89.31	95.14	+5.83
Precision	87.17	96.77	+9.60
Recall	84.55	93.39	+8.84
F1 Score	0.8589	0.950	+0.0911
ROC-AUC	N/A	0.980	N/A

The comparison of the results of the baseline model and our proposed approach in terms of the performance. The MLP model we used in the base paper had an accuracy of 89% vs over 94% given by our multi-architecture output on the same test. Precision, recall, and F1 improve significantly with 0.98 AUC of our model (base paper did not report an exact AUC value).



Our best model exceeds the baseline by ~6 percentage points (95% against ~90%) as can be seen above. More importantly, precision is boosted to ~96-97%, so our model triggers hugely fewer false positives (in a practical application there would be vastly fewer benign files erroneously signalled by the model). At the same time, recall is ~93-94%, which means that our detector intercepts more malware (our base model had recall of ~84.5%, as well). This high recall is at no cost of precision as seen by F1-score of approximately 0.94-0.95, as compared to 0.859 on the baseline. The ROC-AUC that is close to 0.98 indicates strong discrimination ability- the model has a near perfect ability of discriminating malware vs benign across all classification levels. The results that are shown hold significant importance as Malware is one of the most hostile entity and if it is not detected in early stage, it can easily compromise the device and much more.

These advances are due to a number of new additions to our approach beyond the approach used in the base paper:

1. Multi-Architecture Ensemble: As opposed to making use of a single architecture (MLP), we experimented with a variety of deep learning architectures (CNNs, BiLSTM, hybrid models and transformers) and were able to single out the top-performing architecture. Such multi-architecture search enabled us to extract sequence features that the MLP used in the baseline (even considering ensemble of activation functions) could miss. Our last suggested model may be regarded as a composition of ideas: it possesses advantages of the local representation (pattern recognition) of CNNs and sequential memory of LSTMs (in the hybrid case) or, in other words, has a more customized MLP. The thorough search produced a stronger classifier as compared to a baseline study.

2. Hyperparameter Optimization: We ran a formal hyperparameter optimization (with KerasTuner and cross-validation) that was not explicated in the original paper. This fine-tuning adjusted the embedding dimensions, size of layers and regularization to those that were the best in our data. E.g. the base paper architecture employed fixed settings and achieved 89%. In our case by tweaking similar settings, we managed to get above 94% on the same data. That is why it is significant to enhance state-of-the-art performance through hyperparameter search.

3. Explainability and Feature Importance: The process of printing XAI into the pipeline was very enlightening. The results of the logistic regression identified that few API calls have the immense influence over whether microstomia is malignant or not. Most of these calls are related to what common malware do:

   File system operations (CopyFile, MoveFile, DeleteFile): malware frequently copies and pastes files as well as deletes them.

   WSASocket called: malware opening outward or lateral connection.

   System configuration (but again, somewhat surprisingly, direct registry writes, such as RegSetValue weren t the most frequent, but we did find registry queries in benign indicators, and malware may inject registry keys to preserve its state, which we also found in some samples, although such calls may have ended up at the middle range).

   Process manipulation (CreateToolhelp32Snapshot, Module32First to list processes, potentially used with OpenProcess and so on).

   This crypto usage (CryptAcquireContext -ransomware or to secure comms).

   Others such as GetAdaptersAddresses (network info) and GetSystemDirectory (maybe to get system folder path to drop files).

4. Better Sequential Modeling: This was the task on which the base study said it was not able to train RNNs well (their RNN hit ~55% accuracy on this task). We also showed that given good architecture (e.g., BiLSTM or CNN-LSTM) and regularization, then indeed the sequential models can do well (our BiLSTM got ~93% accuracy on the test set and the CNN-LSTM ~94%). Furthermore, we had added a transformer-based model (DistilBERT) to this area, as far as to our knowledge not in the baseline. Although accuracy of our DistilBERT (90.6%) did not exceed CNN/MLP, it demonstrated that transformer-based models are feasible with API sequences and can be fine-tuned by domain-specific data pre-training. The current investigation of sequential models and attention based models is an innovation outside the scope of original work.

5. Extensive Assessment: We expanded on the evaluation critera by calculating AUC and plotting the ROC and PR curves which was not specifically given in the base article. Part of the motivation in doing this, was to ensure that our model would work well over a variety of thresholds and that we could tune where our operating point (threshold) to suit specific needs (e.g., maximize recall in critical system, or precision when resources to investigate alerts are scarce). and we did test another hold-out test set (not to mention some "unseen" sample test on the baseline). In a similar fashion to the final test of the base paper, on a 30 sample unseen set our model has strong generalization, with about 93 to 94 percent accuracy.

To sum up, our malware detection approach shows a substantial leap of its performance compared to the baseline in all critical metrics due to a set of architectural advances, meticulous tuning, and explainable AI. The result is a more precise, reliable and transparent malware classifier with multi-architecture deep learning over dynamic sequences of API calls. These procedures and methods employed in developing this superior solution have

been elaborated in this section of the methodology and they can serve as a guideline in real life applications of faultless malware detection systems.

6.Conclusion and Future Work

In this project, an improved dynamic malware detection structure that utilises several deep learning architectures with the explainable AI concept has been proposed to classify software as malicious or benign due to API calls sequences. In a systematic study of MLP, CNN, BiLSTM, CNN-LSTM, logistic regression, and a DistilBERT transformer, we have found that the CNN-LSTM combination fared the best (~95% accuracy, F1 ≈ 0.95), which is significantly higher than the accuracy of the base study (~89%). The good performance of the logistic regressor (~94 accuracy) indicated that the information can be discriminated by the data itself because of the unique constellations of API calls, and the more adaptable structures made the prediction models more resistant to the varying malware activity through learning more complex temporal and contextual interdependencies. Combining SHAP values with feature importance analysis provided the possibility to identify the importance of API calls, including file operations, network access, and cryptographic operations, thus providing transparency and easy interpretability by an analyst.

Future Work Directions: Based on this work, a number of directions can be taken:

- **Model Teams:** Our team of architectures had slightly varying strong areas, e.g. LSTM was particularly accurate. A weighted ensemble or voting scheme of CNN, MLP and LSTM can potentially perform even better or control the precision/recall trade-off.
- **Transformer Advancements:** DistilBERT did not demonstrate much in this feat but transformers could be pretrained with a larger corpus of sequences of API calls (how to find this data is left as an exercise to the reader, and maybe it does not need to be supervised). It could then be refined on the classification task. These sorts of pretraining may be better at learning finer-grained sequential context or infrequent patterns that are missed by our models and may improve recall in particular. This would be computationally and theoretically possible with modern methods.
- **Feature Augmentation:** We have emphasized only on API call sequences. More dynamic attributes may also be added e.g. arguments of these API calls (file paths accessed, registry keys, accessed by, etc), response values (successful/failed), or even the counts of a few operations. As the base paper and others have suggested, combining of system calls and resource use or other telemetry may be used. Multimodal structure (calls + other behavior) would be able to detect malware that may have benign calls yet have unhealthy mode of appearance or frequency.
- **Adversarial Robustness:** We can model the way in which a sophisticated malware would seek to avoid being detected, and carry out Red teaming. e.g. someone adds a malw+are that adds dummy innocent API calls to fool sequences models, is our CNN or logistic armor up to it? The purpose of this is that we can have some of the altered sequences as an example or even generate adversarial examples to test correctness. That might guide the development of more robust model, e.g. training on partial adversarial perturbed sequences.
- **Real-world deployment testing:** It would be quite interesting to deploy the model in a test environment to scan real software and experience how it performs (determine the false positive rate on clean software at scale, it detects new malware in the wild, etc.). That would be an important step since sometimes the performance of academic

datasets fails to translate to the real performance because of sampling bias or malware altering tactics.

References

U. Gupta et al., “Efficient malware detection using NLP and deep learning model,” *Alexandria Engineering Journal*, vol. 124, pp. 550–564, 2025.

M. Ashik et al., “Detection of malicious software by analyzing distinct artifacts using ML and DL algorithms,” *Electronics*, vol. 10, no. 14, p. 1694, 2021.

R. Vinayakumar et al., “Robust intelligent malware detection using deep learning,” *IEEE Access*, vol. 7, pp. 46717–46738, 2019.

J. Marais et al., “Malware analysis with artificial intelligence and a particular attention on results interpretability,” in *Proc. 18th Int. Conf. Distributed Computing and Artificial Intelligence*, 2022, pp. 43–55.

H. Almaleh et al., “Malware API Calls Detection Using Hybrid Logistic Regression and RNN Model,” *Applied Sciences*, vol. 13, no. 9, p. 5439, 2023. (Used for conceptual comparison of hybrid models).

SHAP Documentation: S. Lundberg et al., “A Unified Approach to Explain the Output of Any ML Model,” Github repository: *shap*, version 0.39.0. (Used for explainable AI methodology).

H. Zhao et al., “Evaluation of supervised ML techniques for dynamic malware detection,” *Int. J. Comput. Intell. Syst.*, vol. 11, no. 1, pp. 1153–1169, 2018.

S. Choudhary and A. Sharma, “Malware detection & classification using machine learning,” in *Proc. 2020 Int. Conf. Emerging Trends in Communication, Control and Computing (ICONC3)*, IEEE, 2020.

M. Ijaz et al., “Static and dynamic malware analysis using machine learning,” in *Proc. 2019 16th Int. Bhurban Conf. Applied Sciences & Technology (IBCAST)*, IEEE, 2019, pp. 687–691.

Abdelrahman, D., Rasslan, M. and Abdelbaki, N. (2025). Comparative Analysis of Malware Detection Approaches in Cloud Computing. *International Journal of Safety and Security Engineering*, 15(2), pp.197–207. doi:https://doi.org/10.18280/ijsse.150201.

Alzaabi, F.R. and Mehmood, A. (2024). A Review of Recent Advances, Challenges, and Opportunities in Malicious Insider Threat Detection Using Machine Learning Methods. *IEEE Access*, [online] 12, pp.30907–30927. doi:https://doi.org/10.1109/ACCESS.2024.3369906.

Bilot, T., Nour El Madhoun, Agha, K.A. and Anis Zouaoui (2024). A Survey on Malware Detection with Graph Representation Learning. *ACM Computing Surveys*, 56(11), pp.1–36. doi:https://doi.org/10.1145/3664649.

Gupta, U., Kandpal, S., Alamro, H., Asiri, M.M., Alanazi, M.H., Al-Sharafi, A.M. and Sorour, S. (2025). Efficient malware detection using NLP and deep learning model. *Alexandria Engineering Journal*, [online] 124, pp.550–564. doi:https://doi.org/10.1016/j.aej.2025.03.118.

Heitor Werneck, Batista, D.M., Hirata, R. and Queiroz, M. (2024). Real-Time Intrusion Detection with Sequence-Aware Neural Networks for Internet of Medical Things. [online] pp.1–6. doi:https://doi.org/10.1109/vcc63113.2024.10914399.

ieeexplore.ieee.org. (2025). *Self-Supervised Vision Transformers for Malware Detection | IEEE Journals & Magazine | IEEE Xplore*. [online] Available at: https://ieeexplore.ieee.org/document/9889730 [Accessed 11 Aug. 2025].

Liu, Z. (2024). A Review of Advancements and Applications of Pre-Trained Language Models in Cybersecurity. doi:https://doi.org/10.1109/isdfs60797.2024.10527236.

Mehta, R., Olha Jurečková and Stamp, M. (2023). A natural language processing approach to Malware classification. *Journal of Computer Virology and Hacking Techniques*, 20(1), pp.173–184. doi:https://doi.org/10.1007/s11416-023-00506-w.

Mimura, M. and Ito, R. (2021). Applying NLP techniques to malware detection in a practical environment. *International Journal of Information Security*, 21(2), pp.279–291. doi:https://doi.org/10.1007/s10207-021-00553-8.

Mohamed, N. (2024). A Comprehensive Review of Natural Language Processing Techniques for Malware Detection. [online] pp.1–7. doi: https://doi.org/10.1109/icccnt61001.2024.10724079.

Nsikak Owoh, Adejoh, J., Hosseinzadeh, S., Ashawa, M., Osamor, J. and Qureshi, A. (2024). Malware Detection Based on API Call Sequence Analysis: A Gated Recurrent Unit–Generative Adversarial Network Model Approach. *Future Internet*, [online] 16(10), pp.369–369. doi: https://doi.org/10.3390/fi16100369.

Yao, Y., Zhu, Y., Jia, Y., Shi, X., Zhang, L., Zhong, D. and Duan, J. (2023). Research on Malware Detection Technology for Mobile Terminals Based on API Call Sequence. *Mathematics*, [online] 12(1), p.20. doi: https://doi.org/10.3390/math12010020.