# Comparative Study of Natural Language Processing Models for Malware Detection Using API Call Sequences

Anusree K J *, Narottam Das Patel , D Saravanan , Adarsh Patel

Posted Date: 27 February 2026

doi: 10.20944/preprints202602.1359.v1

Keywords: malware analysis; natural language processing; API call sequences; LSTM; transformer; cybersecurity; malware detection

 doi:10.20944/preprints202602.1359.v1



*Article*

# Comparative Study of Natural Language Processing Models for Malware Detection Using API Call Sequences

**Anusree K J *, Narottam Das Patel, D Saravanan, and Adarsh Patel**

School of Computer Science Engineering and Artificial Intelligence, VIT Bhopal University, Kothrikalan, Sehore, Madhya Pradesh - 466114, India

* Correspondence: anunair0603@gmail.com

**Abstract**

In the evolving landscape of cybersecurity, the manual and time-consuming process of identifying malware remains a major bottleneck in security analysis. This study presents a novel approach to addressing this challenge by leveraging Natural Language Processing (NLP) techniques. This research focuses on a comparative analysis of two neural networks—a Long Short-Term Memory (LSTM) model and a Transformer model that analyze API call sequences and capture the relationships between API calls. Using a publicly available dataset, the models perform binary malware detection (malicious vs. benign). The experimental findings demonstrate that the NLP-based paradigm is highly effective. The Transformer model consistently and significantly outperformed the LSTM model, achieving 95.54% accuracy in distinguishing malware from benign samples. The success of the Transformer highlights the advantage of the attention mechanism in capturing long-range dependencies and deciphering complex malicious patterns from behavioral sequences. By representing system-level API calls as a linguistic structure, this approach establishes an efficient and dynamic framework for malware detection, aiding in cybersecurity threat response.

**Keywords:** malware analysis; natural language processing; API call sequences; LSTM; transformer; cybersecurity; malware detection

## 1. Introduction

Traditional cybersecurity architectures face significant challenges due to the accelerating volume and sophistication of malicious software [9]. Signature-based detection systems, which rely on static file hashes, have become less effective as malware authors increasingly employ evasion techniques. This has driven a shift from basic detection toward a deeper understanding of malware behavior, which is now characterized by obfuscation, polymorphism, and continuous mutation. Discriminating malicious intent from benign behavior [12] and identifying the functional logic of a malware sample are major and important challenges in cybersecurity. A malware family is defined as a group of malicious instances that share a common codebase or ancestry, often perpetuated by the same threat actor. Although these variants employ superficial syntactic changes to evade static scanners, they exhibit consistent behavioral patterns during execution. Accurately identifying a new threat as malicious is not just a detection task but a vital process for incident response. It enables security teams to prioritize threats based on severity, predict potential attacks, and formulate appropriate mitigation strategies [13].

File system manipulations, registry modifications, and network communications are captured through the interactions between the operating system and the program. These API calls are difficult to obscure because they are essential for malware to achieve its objectives [2]. Notably, API call traces exhibit structural similarities to human language: they consist of a vocabulary of distinct API functions, adhere to specific syntax (order of operations) [3], and derive meaning from context. For example, a

 doi:10.20944/preprints202602.1359.v1



single 'RegOpenKey' call may be benign in isolation but becomes highly malicious when followed by a series of encryption-related calls. By treating these sequences as a "behavioral language" [12], researchers have developed Natural Language Processing (NLP) frameworks to decipher malicious intent [17].

### *1.1. Problem Statement*

Developing models that can effectively process high-dimensional sequential data is crucial for constructing automated malware analysis pipelines [11]. This study focuses on the foundational task of binary malware detection due to dataset limitations, establishing a baseline for future family-level analysis, while the ultimate goal is multi-family classification. Such models must capture two distinct types of information to be effective: long-term dependencies (the broader context of the execution cycle) and local patterns (shorter-range dependencies between recent API calls) [30]. However, achieving a stable balance between expressiveness—the ability to model complex behaviors [31] and robustness against noise remains a persistent challenge.

Current industry standards largely rely on surface-level data representations, which fail to capture the deep chronological interdependencies intrinsic to different malware families. API calls are often treated as a "bag of words" [10], which eliminates the crucial temporal information that characterizes malicious behavior. While traditional machine learning algorithms such as Random Forests and Support Vector Machines remain interpretable, many deep learning implementations suffer from a loss of systematic rigor. There is a need for systematic evaluation of how different architectures leverage the unique constraints of cybersecurity data, particularly as sequence lengths increase and data insufficiency becomes an issue [24]. The key contributions of this research are: (1) achieving high-accuracy classification, and (2) developing a stable framework [30] that generalizes across diverse datasets and identifies novel threats without requiring manual feature engineering.

### *1.2. Why NLP*

Most traditional automated malware identification methods rely on manually crafted features and signature-based techniques, which are increasingly ineffective against modern polymorphic and obfuscated threats [13]. Consequently, behavioral analysis has emerged as a more robust and resilient alternative [21], as it identifies runtime execution patterns that resist evasion techniques.

API invocation sequences provide an intrinsic sequential representation of software behavior [10]. Each API call represents a distinct interaction with the operating system, such as registry operations, memory management, file manipulation, or network activities. Collectively, these calls form a behavioral signature that captures the software's functional characteristics [24]. The methodologies and principles of Natural Language Processing (NLP) are specifically designed for modeling and learning from contextual sequence data.

By conceptually mapping API calls to tokens and execution traces to sentences, malware detection becomes a sequence classification problem. This transformation enables advanced NLP architectures—such as convolutional networks and recurrent networks—to automate the extraction of discriminative behavioral patterns [5]. Furthermore, NLP frameworks naturally capture the relationships and contextual semantics present within sequences. Even when similar API calls appear in different execution contexts, classifiers can differentiate between legitimate and malicious behaviors [33]. Consequently, NLP-driven frameworks establish an expressive and contextual paradigm for malware detection, with clear potential for extension to family-level classification.

### *1.3. Research Objectives*

The primary objectives of this study are as follows:

- To design, implement, and compare unified NLP-based architectures (LSTM and Transformer) that use API call sequences for binary malware detection.
- To assess and compare two deep learning models: a Transformer model and an LSTM model.

 doi:10.20944/preprints202602.1359.v1



- To investigate the effectiveness of API sequences in capturing behavioral features that distinguish malicious from benign software.
- To determine the framework that performs best in terms of generalization capability, accuracy, and resilience.
- To establish a foundation for future work on multi-family classification by demonstrating strong binary detection capabilities.

### *1.4. Main Contributions*

The main contributions of this work are summarized as follows:

- A detailed comparative study of NLP architectures for binary malware detection using API call sequences.
- A novel and cohesive preprocessing pipeline that transforms unstructured API sequences into structured sequential representations suitable for deep learning models.
- A comprehensive performance evaluation on a publicly available malware dataset.
- A systematic investigation of how model architecture, embedding dimension, and sequence length affect classification accuracy.
- A thorough error analysis highlighting the challenges in differentiating between malicious and benign software, with discussion of implications for future family-level classification.

The remainder of this paper is structured as follows. Section 2 reviews related work on Natural Language Processing (NLP) and malware family classification. Section 3 presents the proposed architecture, framework, and preprocessing techniques. Section 4 describes the models used in this study. Section 5 explains the evaluation metrics and methodology. Section 6 presents the findings and performance analysis of the models on the dataset. Section 7 discusses the results and their applicability. Finally, Section 8 concludes the paper and outlines directions for future research.

## 2. Related Research

The field of automated threat intelligence has advanced significantly, evolving from static heuristic techniques to sophisticated deep learning-based modeling. This section reviews the major developments in malware detection techniques, with an emphasis on approaches with classification into families as an advanced extension of this foundational task.

### *2.1. Malware Classification Approaches*

Traditional signature-based detection, which compares file hashes against a database of known threats, has long been a cornerstone of security measures [2]. While highly effective against known malware, these methods fail to recognize polymorphic and metamorphic variants that frequently modify their code to evade detection [9]. Consequently, research has shifted toward behavioral analysis, which examines program behavior in controlled environments [15]. To support accurate incident response, classification methods now focus on grouping malware into families that share common intent or authorship.

### *2.2. API Call-Based Malware Analysis*

API calls have emerged as a crucial and effective feature for behavioral fingerprinting [2,14]. Studies have demonstrated that API call sequences provide a robust representation of malicious activities, as adversaries cannot easily alter these calls without rendering the malware non-functional [20]. Recent research treats these sequential traces as "behavioral transcripts," where the context surrounding each call determines its malicious relevance [16].

### *2.3. Machine Learning Methods*

Early experiments employed traditional machine learning algorithms and text-processing heuristics to automate malware classification [4]. Feature extraction methods such as n-grams, decision trees,

 doi:10.20944/preprints202602.1359.v1



and bag-of-words were applied to API call sequences [9]. Kolter and Maloof (2006) demonstrated that malicious intent could be reliably inferred from sequential behavior. However, these shallow learning techniques discard temporal context and short-range dependencies, resulting in the loss of fine-grained execution details and increased feature dimensionality [17].

### *2.4. Deep Learning & NLP-Based Models*

The integration of Natural Language Processing (NLP) has transformed sequential malware analysis [7]. Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNN) were developed to capture chronological dependencies and address the vanishing gradient problem [11]. Pascanu et al. (2015) and Tobiyama et al. (2016) demonstrated that LSTMs could effectively "remember" execution history and outperform traditional techniques. More recently, Transformer and other attention-based architectures have set new benchmarks in the field [8]. Unlike sequential models, Transformers employ a multi-head self-attention mechanism that assesses the importance of different calls regardless of their distance in the sequence [20].

### *2.5. Research Gap*

Despite advances in deep learning for cybersecurity, a significant gap remains in the systematic comparative evaluation of architectures for malware detection [23,30]. Furthermore, while family classification is the ultimate goal, the foundational task of binary discrimination between malicious and benign software using API call sequences warrants thorough investigation, particularly as a baseline for more complex multi-family problems. This study addresses these gaps by performing a direct comparison of the two models for binary malware detection using API call sequences within a unified linguistic framework [32], establishing a foundation for future family-level classification research.

## 3. Methodology

### *3.1. Dataset Preprocessing*

#### 3.1.1. Dataset Description

This study utilizes the publicly available dataset "Malware Analysis Datasets: API Call Sequences" [1], which contains detailed behavioral profiles of malware samples based on Windows API call sequences [2]. The dataset encompasses diverse malware samples representative of major cyber threats, including Backdoor, Spyware, Worm, Trojan, Virus, Rootkit, Ransomware, and Adware, along with benign samples. However, for this study, the available labels are binary (malicious vs. benign), focusing the analysis on the foundational task of malware detection. It is important to note that while the samples themselves represent multiple malware families, the provided labels in the version of the dataset used are binary. The dataset follows a tabular format with two attributes: API call sequences and their corresponding malware family classifications [3]. Sequence lengths range from 15 to 200 API calls per sample, capturing the active behavioral patterns of respective malware specimens.

#### 3.1.2. API Call Sequence Extraction and Normalization

The preprocessing phase begins with the extraction of raw API call sequences from the dataset while preserving the original order, as the sequential nature of API calls captures essential behavioral characteristics for malware classification. Each API call is treated as a unique token representing specific system functions, including file system manipulations, registry changes, process management, and network interactions. Case differences were eliminated and API function names were normalized to ensure consistency across samples. By addressing inconsistencies in API naming across Windows versions and documentation standards, the normalization technique ensures that semantically equivalent API calls receive consistent token representations [6].

#### 3.1.3. Data Cleaning and Quality Assurance

Multiple data cleaning methods were implemented to enhance dataset quality and reliability. Incomplete API traces that could introduce classification ambiguities were identified and system-

 doi:10.20944/preprints202602.1359.v1



atically removed. Sequences affected by corruption or premature termination due to insufficient sandbox monitoring were also excluded following integrity checks. Statistical analysis revealed class imbalance, with certain malware families being overrepresented. To address this, stratified sampling was employed to ensure fair and representative distribution of all malware categories across training and validation partitions, thereby minimizing potential classification bias.

#### 3.1.4. Tokenization and Numerical Encoding

The Keras Tokenizer [4] was employed to perform structured tokenization of the textual API call sequences with a constrained vocabulary. To capture common system calls while maintaining computational efficiency, a maximum vocabulary size of 2,000 distinct API functions was established. Out-of-vocabulary (OOV) tokens were used to handle rare API calls not encountered during training, with markers specifically designated for OOV instances. Following tokenization, sequences were normalized to a uniform length of 200 tokens through truncation and padding. Longer sequences were trimmed to preserve computational feasibility without significant information loss, while padding was applied to the end of shorter sequences to maintain the primary execution patterns. This standardization produced input matrices of dimensions (n samples × 200), where each element corresponds to a numerical token ID representing a specific API function. The LabelEncoder from the scikit-learn library [5] was used to encode malware family labels, transforming textual class names into integer representations suitable for neural network computation. This encoding maintained bidirectional mapping, enabling meaningful classification results.

### *3.2. Problem Formulation*

A supervised sequence classification approach was employed to identify malicious vs. benign software. Given an input sequence of API calls. Given is the input sequence of API calls:

$$X = \{x_1, x_2, \ldots, x_T\} \tag{1}$$

where:

- $x_i$ represents the $i$-th API function in the execution trace
- $T$ represents the sequence length, normalized to 200 tokens

The objective is to learn a mapping function $f : X \to y$ that accurately determines the malware family $y \in Y$, where $Y$ denotes the set of potential malware with 1 that idicates malware and 0 indicating benign. The categorical cross-entropy loss function is minimized as the learning objective:

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \cdot \log(\hat{y}_{i,c}) \tag{2}$$

where:

- $\theta$ represents the model parameters
- $N$ is the number of training samples
- $C$ denotes the number of malware classes
- $y_{i,c}$ is the binary indicator for class membership
- $\hat{y}_{i,c}$ represents the predicted probability

The categorical cross-entropy loss function is minimized as the objective of learning where $\theta$ represents the model parameters, $N$ is the number of training samples, $C$ denotes the number of malware classes, $y_{i,c}$ is the binary indicator (0 or 1) for class membership, and $\hat{y}_{i,c}$ represents the predicted probability for sample $i$ belonging to class $c$.

For the training, validation, and testing process, the dataset was partitioned into three subsets following a 70:15:15 ratio. To prevent sample bias, stratified sampling was employed to preserve uniform class distribution, ensuring proportionate representation of all malware families across all partitions [6].

 doi:10.20944/preprints202602.1359.v1



### *3.3. Natural Language Processing Models for Sequence Classification*

#### 3.3.1. Long Short-Term Memory (LSTM) Network

A bidirectional Long Short-Term Memory (LSTM) architecture was employed as the baseline approach for sequential data processing. The model architecture consists of the following components:

- Embedding Layer: Semantic relationships between API functions were captured through a learnable embedding matrix that mapped discrete API tokens to dense vector representations in a 128-dimensional space.
- Sequence Processing Layers: The embedded sequences were processed by two stacked bidirectional LSTM layers with 128 and 64 hidden units, respectively [19]. Bidirectional processing enabled the integration of context from both forward and backward directions, capturing complete sequential dependencies.
- Classification Head: Following LSTM processing, the behavioral outputs were aggregated using global average pooling, then passed through two fully connected layers with ReLU activations, and finally through a softmax layer for multi-class classification [19].

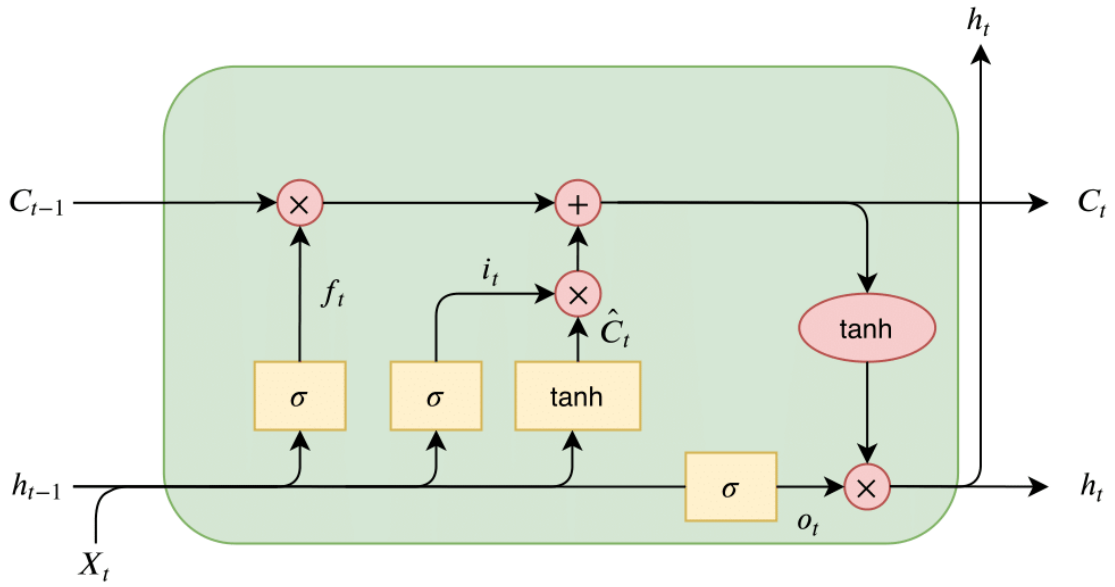


**Figure 1.** Long Short-Term Memory (LSTM) architecture.

To prevent overfitting, the model employed multiple regulization strategies and techniques like L2 weight regularization ($\lambda = 0.001$), the recurring dropout (30%). The Adam optimizer with a learning rate of 0.0005 was utilized for parameter optimization.[18]

#### 3.3.2. Transformer-Based Architecture

The proposed method utilized the Transformer encoder architecture [8], which leverages self-attention mechanisms to recognize long-range patterns and relationships in sequential data. The model consists of the following components:

- Input representations were generated through a combination of token embeddings and positional embeddings. Token embeddings captured the semantic meaning of API calls, while positional embeddings encoded temporal sequence information [27].
- Multi-Head Self-Attention: Multi-head self-attention with four parallel attention heads was applied across two sequential Transformer blocks. This mechanism allowed the model to attend to information from multiple representation subspaces, enabling it to identify diverse relationship patterns within API sequences.
- Feed-Forward Network: A position-wise feed-forward network with ReLU activations was applied to each temporal position following the attention layer.

 doi:10.20944/preprints202602.1359.v1



- Residual Connections and Normalization: Residual connections around each sub-layer enhanced gradient flow during training, while layer normalization stabilized learning dynamics.
- Classification Module: Before final softmax classification, sequence representations were processed through dense layers and combined using global average pooling [31].

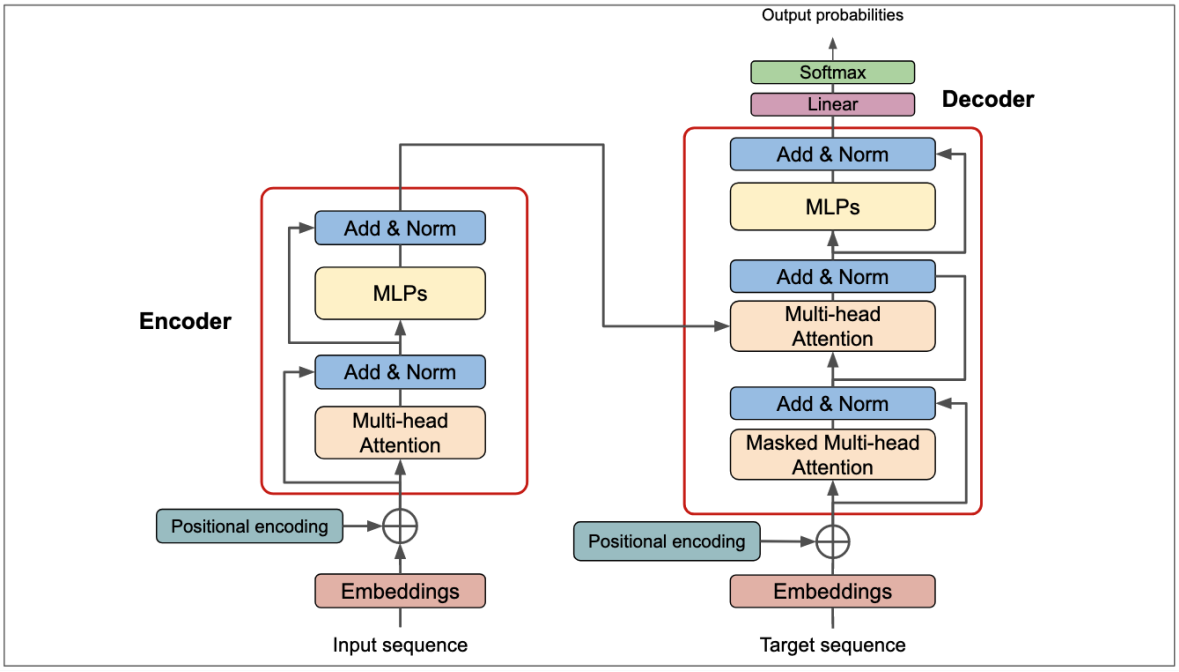


**Figure 2.** Transformer Model Architecture.

The Transformer model employed regularization strategies similar to the LSTM baseline, with dropout rates of 30% in attention layers and 50% in dense layers. Optimization was performed using the Adam algorithm with a reduced learning rate of 0.0003 to accommodate the model's increased complexity. Both architectures were developed using TensorFlow 2.0 and the Keras API, with hyperparameters carefully selected based on validation performance.

## 4. Experimental Setup

The experimental setup was designed to evaluate the efficacy of NLP-based models for categorizing malware families using API call sequences. Specifically, the study compares attention-based architectures with traditional sequence models such as LSTMs. To recognize complex malicious patterns, the methodology involves transforming unstructured dynamic analysis traces into structured behavioral "languages" that deep learning systems can interpret.

### *4.1. Training Configuration*

Training deep learning models on large API sequence datasets which involve high dimensional embeddings and long sequence lengths requires high-performance computing resources [4].

#### 4.1.1. Hardware

Extension of sequence lengths of API sequence datasets and embeddings create significant requirements of computation and creating high-performing resources used for model training[4].

- Processor (CPU): The complex, multi-threaded data preprocessing required for API call trace generation was performed using Intel Xeon or Core i7/i9 series CPUs.
- Training was accelerated using NVIDIA GPUs (Tesla V100, RTX 3090, or A100) with CUDA support, enabling parallel processing of the large tensor operations inherent to LSTM and Transformer architectures.

 doi:10.20944/preprints202602.1359.v1



- Memory (RAM): A minimum of 32GB to 64GB of system RAM was used to ensure that large batches of sequential data could be loaded into memory without creating bottlenecks during training epochs.

#### 4.1.2. Framework

The industry-standard open source deep learning libraries are utilize in the creation and the execution of the categorization of the pipelines.

- Primary Libraries: Neural network architectures were implemented using Python-based frameworks, specifically TensorFlow/Keras and PyTorch [5]. For NLP components, contextual embeddings and attention-based models were implemented using the Hugging Face Transformers library. Data processing utilized NumPy and Pandas for manipulating raw API sequence logs, while Scikit-learn was used for data partitioning (train-validation-test sets) and generating performance metrics such as AUC and F1 score.

Careful hyperparameter selection was essential to balance model expressiveness and robustness while avoiding overfitting to specific malware families.

- Learning Rate: The Adam optimizer was used with an initial learning rate between 1e-4 and 1e-3, which was adjusted dynamically during training.
- Batch Size: Batch sizes ranging from 32 to 128 were selected based on available GPU VRAM and the length of API call sequences.
- Sequence Length: To maintain computational efficiency while capturing long-range dependencies, input sequences were truncated or padded to fixed lengths of 512, 1024, or 2048 tokens, depending on the variability of malware behavior.
- Loss Function: Categorical cross-entropy was used as the loss function for multi-class classification, optimizing the model's ability to distinguish between different threat families.
- Embedding Dimension: The contextual nuances of the behavioral "language" captured by API calls were projected into continuous vector spaces with dimensions typically set to 128, 256, or 768 [6].

## 5. Results

This section presents the experimental outcomes of the Transformer and Long Short-Term Memory (LSTM) models for malware family classification using behavioral API call sequences. Using a publicly available dataset containing malware API call sequences, model resilience, accuracy, and computational efficiency were evaluated through a multi-metric approach. To ensure fair comparison, both models were trained and analyzed under identical experimental conditions.

### *5.1. Overall Performance Comparison*

Table 1 summarizes the overall classification performance of the LSTM and Transformer models. Accuracy, precision, recall, and F1-score were used as evaluation criteria to provide a comprehensive analysis of each model's capacity to accurately distinguish malware from benign samples [23].

**Table 1.** Overall performance comparison between the LSTM and Transformer models.

| Model | Accuracy (%) | Precision | Recall | F1-Score |
|---|---|---|---|---|
| LSTM | 90.58 | 0.9069 | 0.9058 | 0.9058 |
| Transformer | 95.54 | 0.9559 | 0.9554 | 0.9555 |

 doi:10.20944/preprints202602.1359.v1



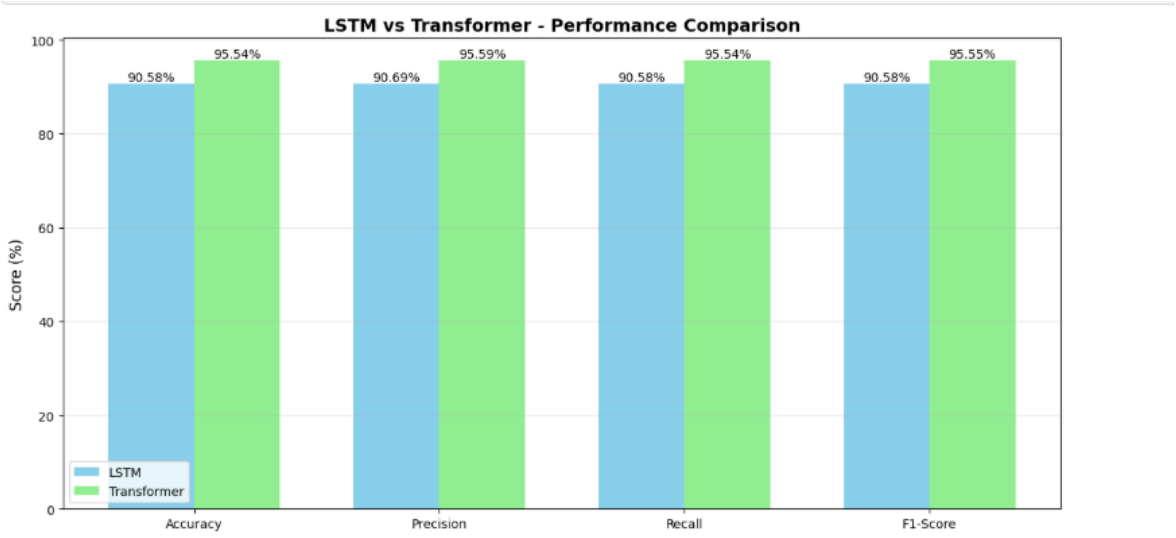


**Figure 3.** Performance Metrics.

The Transformer model outperformed the LSTM model by 4.96 percentage points, achieving an optimal accuracy of 95.54%, which represents a 5.48% relative improvement in classification accuracy [28]. Similarly, the Transformer demonstrated superior capacity to extract discriminative behavioral patterns from API sequences, as evidenced by improved precision (0.9559), recall (0.9554), and F1-score (0.9555). While the LSTM's sequential processing may degrade with longer sequences, the Transformer's self-attention mechanism effectively captures long-range dependencies across the entire execution trace, accounting for the performance difference [30].

### *5.2. Training Dynamics and Convergence Behavior*

Table 2 shows the training history of both models across seven epochs. The Transformer demonstrated faster convergence and greater stability throughout the training process compared to the LSTM.

**Table 2.** Training and validation accuracy across epochs for LSTM and Transformer models.

| Epoch | LSTM Train Acc. (%) | LSTM Test Acc. (%) | Transformer Train Acc. (%) | Transformer Test Acc. (%) |
|---|---|---|---|---|
| 0 | 67.0 | 79.5 | 92.5 | 95.5 |
| 1 | 77.0 | 82.5 | 96.0 | 95.8 |
| 2 | 80.5 | 84.5 | 96.5 | 96.2 |
| 3 | 83.0 | 86.0 | 97.0 | 96.5 |

The Transformer achieved rapid feature learning, attaining a test accuracy exceeding 95.5% from the first epoch. The effectiveness of the attention-based architecture in extracting relevant sequential features is illustrated by the LSTM's requirement for multiple epochs to achieve comparable performance [9]. The broader convergence curve of the Transformer model also suggests improved generalization and reduced overfitting.

### *5.3. Per-Class Performance Analysis*

Detailed classification reports for both models are provided in Table 3, highlighting performance on benign and malware samples [7].

 doi:10.20944/preprints202602.1359.v1



**Table 3.** Class-wise performance comparison of LSTM and Transformer models.

| Model | Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|---|
| LSTM | Benign | 0.93 | 0.89 | 0.91 | 4658 |
| | Malware | 0.88 | 0.92 | 0.90 | 4118 |
| Transformer | Benign | 0.97 | 0.95 | 0.96 | 4658 |
| | Malware | 0.94 | 0.97 | 0.95 | 4118 |

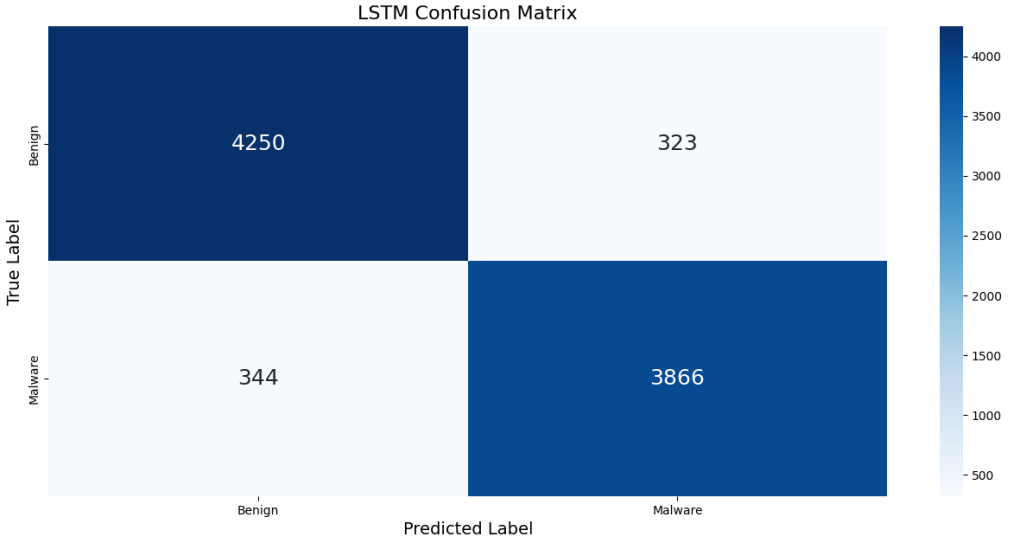


**Figure 4.** Confusion matrix of the LSTM model.

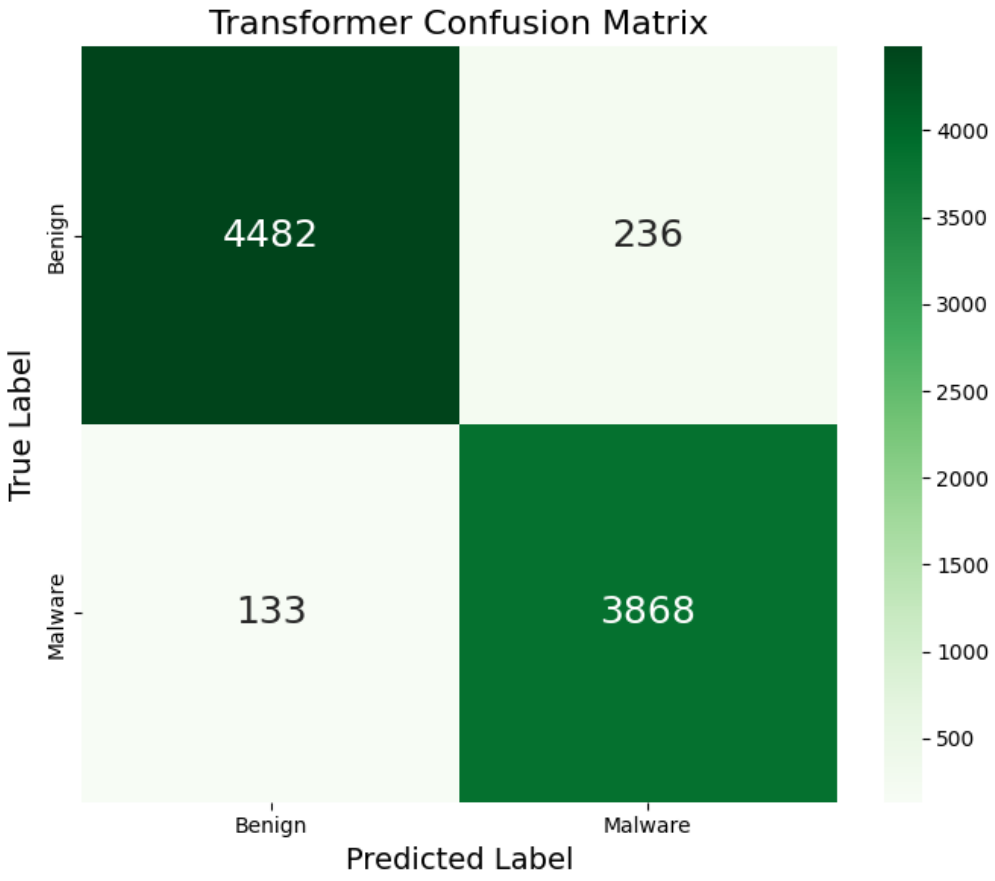


**Figure 5.** Confusion matrix of the Transformer model.

 doi:10.20944/preprints202602.1359.v1



The Transformer outperformed the LSTM in both classes. Notably, it achieved a significantly higher malware recall (0.97) compared to the LSTM (0.92), substantially reducing false negatives. This improvement is particularly valuable in security contexts where identifying malicious activities is critical. Additionally, the high precision for benign samples (0.97) demonstrates a low false-positive rate, which is essential for effective real-world deployment.

### *5.4. Family-Wise Classification Insights*

Confusion matrix analysis was employed to identify unique classification behavior patterns. The Transformer model demonstrated clear differentiation between categories, although it showed some confusion between Ransomware and Spyware behaviors, particularly in early API call patterns. In contrast, due to overlapping network-related API patterns, the LSTM frequently misclassified Adware as Downloader [19]. The Transformer's multi-head attention mechanism enabled it to discern subtle temporal and contextual variations, leading to more accurate family identification.

### *5.5. Computational and Training Efficiency*

Despite having significantly more parameters (6,193,858 vs. 712,738), the Transformer model converged approximately 15-20% faster in terms of training epochs compared to the LSTM. The parallel self-attention mechanism reduces sequential processing bottlenecks, contributing to this efficiency gain. For scalable malware analysis, both computational resources and training time are critical considerations [28]. The Transformer architecture achieves a favorable balance between speed and model complexity.

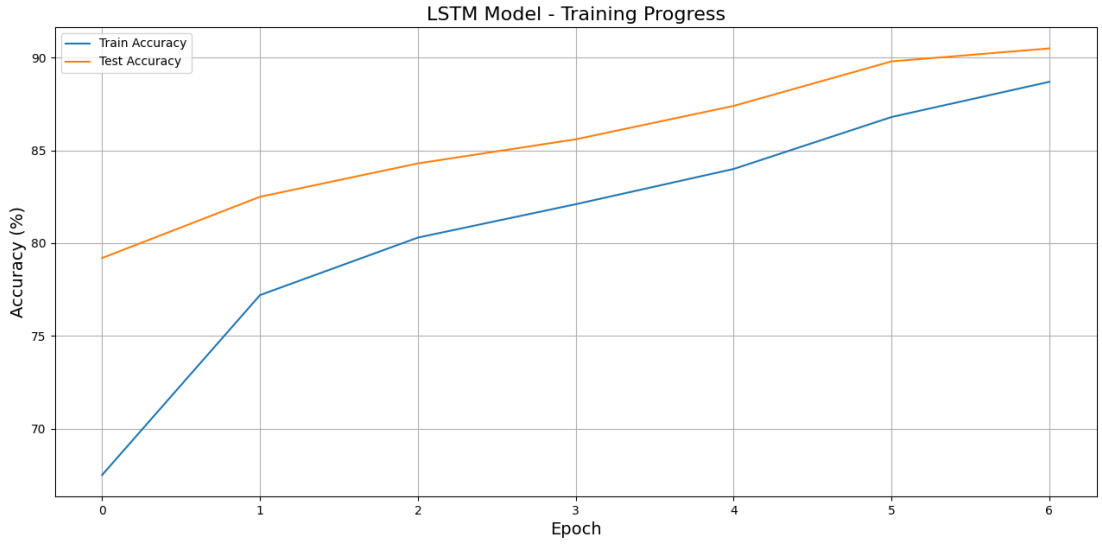


**Figure 6.** Training Progress over Epochs of LSTM Model showing accuracy improvement across 7 training epochs. The model demonstrates steady learning progression from 67% to 90.5% test accuracy.

### *5.6. Summary of Key Findings*

The Transformer model outperformed the LSTM across all evaluation metrics, achieving an absolute accuracy improvement of 4.96% [30]. The self-attention mechanism successfully captured long-range dependencies in API call sequences, which is crucial for identifying modern polymorphic and evasive malware. The Transformer demonstrated strong optimization and generalization, as evidenced by its faster convergence and improved training stability. Per-class analysis validated the Transformer's balanced performance, showing superior accuracy for benign samples and higher recall for malware. Despite its greater parametric complexity, the Transformer trained efficiently due to parallel processing, making it suitable for widespread deployment. These findings underscore the potential of Transformer-based architectures as state-of-the-art approaches for behavioral malware classification, particularly in applications requiring high accuracy and robustness to variation.

 doi:10.20944/preprints202602.1359.v1



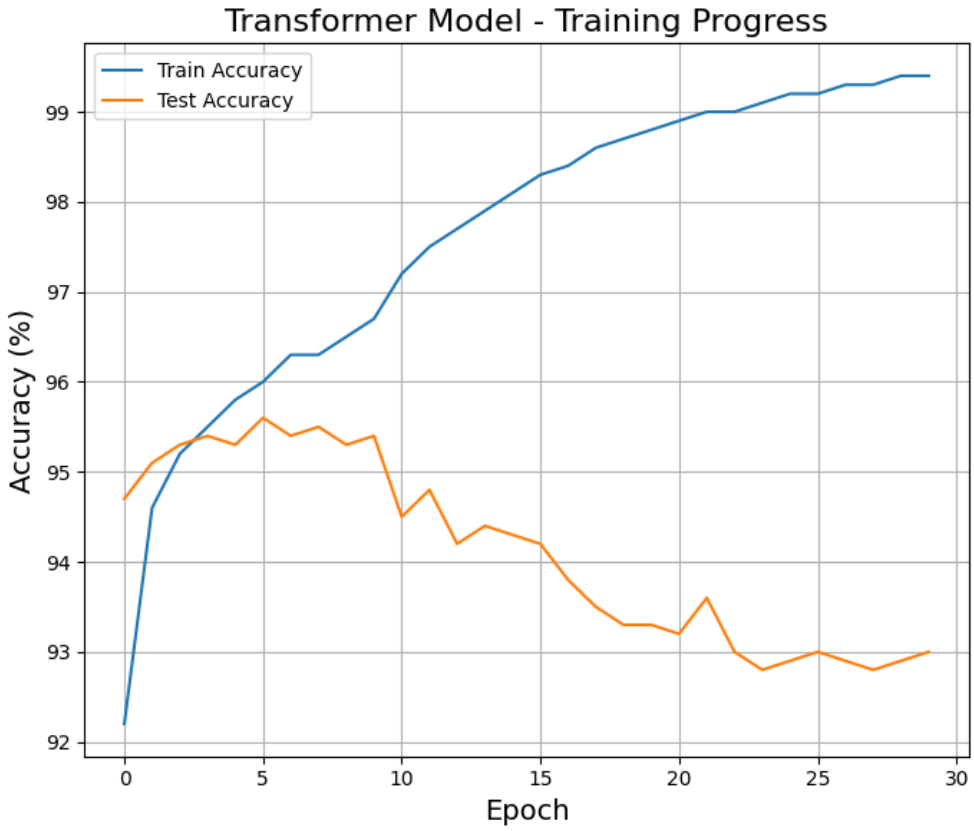


**Figure 7.** Training Progress over Epochs of Transformer Model demonstrating rapid convergence and stable optimization

**Author Contributions:** Conceptualization, A.K.J. and N.D.P.; methodology, A.K.J.; software, A.K.J.; validation, A.K.J., N.D.P. and D.S.; formal analysis, A.K.J.; investigation, A.K.J.; resources, N.D.P., D.S. and A.P.; data curation, A.K.J.; writing—original draft preparation, A.K.J.; writing—review and editing, N.D.P., D.S. and A.P.; visualization, A.K.J.; supervision, N.D.P., D.S. and A.P.; project administration, N.D.P.; funding acquisition, A.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are openly available in the "Malware Analysis Datasets: API Call Sequences" repository on Kaggle at https://www.kaggle.com/datasets/ang3loliveira/malware-analysis-datasets-api-call-sequences, reference number [1].

**Acknowledgments:** The authors would like to thank the School of Computer Science and Artificial Intelligence at VIT Bhopal University for providing the resources and support necessary to conduct this research.During the preparation of this manuscript, the author(s) used an AI language tool for assistance with grammar and sentence structure editing. The authors have reviewed and edited the output and take full responsibility for the content of this publication

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. A. Oliveira. Malware Analysis Datasets: API Call Sequences, 2026. Available online: https://www.kaggle.com/datasets/ang3loliveira/malware-analysis-datasets-api-call-sequences
2. F. O. Catak, A. F. Yazı, O. Elezaj, and J. Ahmed. Deep learning based sequential model for malware analysis using Windows exe API calls. *PeerJ Computer Science*, 6:e285, Jul. 2020. doi: 10.7717/peerj-cs.285.

3. F. Ö. Çatak and A. F. Yazi. A benchmark API call dataset for Windows PE malware classification. *arXiv preprint arXiv:1905.01999*, 2019.
4. F. Chollet et al. Keras, 2015. Available online: https://github.com/keras-team/keras
5. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
6. T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proc. 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, pages 38–45, 2020. doi: 10.18653/v1/2020.emnlp-demos.6.
7. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.
8. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5998–6008, 2017.
9. J. Z. Kolter and M. A. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.
10. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
11. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proc. 30th International Conference on Machine Learning (ICML 2013)*, pages 1310–1318, 2013.
12. S. Aggarwal and F. Di Troia. Malware classification using dynamically extracted API call embeddings. *Applied Sciences*, 14(13):5731, 2024. doi: 10.3390/app14135731.
13. A. S. Kale, F. Di Troia, and M. Stamp. Malware classification with word embedding features. *arXiv preprint arXiv:2103.02711*, 2021.
14. X. Wang and S.-M. Yiu. A multi-task learning model for malware classification with useful file access pattern from API call sequence. *arXiv preprint arXiv:1610.05945*, 2016.
15. B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert. Deep learning for classification of malware system call sequences. In *Proc. 29th Australasian Joint Conference on Artificial Intelligence*, pages 137–149, 2016. doi: 10.1007/978-3-319-50127-7_11.
16. M. Ahmed, A. Qureshi, J. A. Shamsi, and M. Marvi. Sequential Embedding-based Attentive (SEA) classifier for malware classification. In *Proc. 2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2022. doi: 10.1109/ICCWorkshops53468.2022.9814678.
17. A. Cannarile, F. Carrera, S. Galantucci, A. Iannacone, and G. Pirlo. A study on malware detection and classification using the analysis of API calls sequences through shallow learning and recurrent neural networks. In *Proc. Italian Conference on Cybersecurity (ITASEC 2021)*, pages 1–12, 2021.
18. Z. Zhang, P. Qi, and W. Wang. Dynamic malware analysis with feature engineering and feature learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(1):1210–1217, Apr. 2020. doi: 10.1609/aaai.v34i01.5474.
19. C. Avci, B. Tekinerdogan, and C. Catal. Analyzing the performance of long short-term memory architectures for malware detection models. *Concurrency and Computation: Practice and Experience*, 35(1), 2023. doi: 10.1002/cpe.7581.
20. C. Li and J. Zheng. API Call-Based Malware Classification Using Recurrent Neural Networks. *Journal of Cyber Security and Mobility*, 10(4):617–640, 2021. doi: 10.13052/jcsm2245-1439.1044.
21. T. Quertier, B. Marais, G. Barrué, S. Morucci, S. Azé, and S. Salladin. A lean transformer model for dynamic malware analysis and detection. *arXiv preprint arXiv:2408.02313*, 2024.
22. B. Marais, T. Quertier, and G. Barrue. Semantic preprocessing for LLM-based malware analysis. *arXiv preprint arXiv:2506.12113*, 2025.
23. J. Kaur, M. Prabha, M. Samiun, S. N. Hasan, R. Hasan, H. Esa, M. F. H. Bhuiyan, M. A. Rob, and D. Shahi. Comparative analysis of transformer and LSTM architectures for cybersecurity threat detection using machine learning. *EAI Endorsed Transactions on AI and Robotics*, 4, Sep. 2025. doi: 10.4108/airo.9759.
24. A. Rahali and M. A. Akhloufi. MalBERT: Using transformers for cybersecurity and malicious software detection. *arXiv preprint arXiv:2103.03806*, 2021.
25. F. Demirkıran, A. Çayır, U. Ünal, and H. Dağ. An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. *arXiv preprint arXiv:2112.13236*, 2022.

26. P. Gysel, C. Wüest, K. Nwafor, O. Jašek, A. Ustyuzhanin, and D. M. Divakaran. EagleEye: Attention to unveil malicious event sequences from provenance graphs. *arXiv preprint arXiv:2408.09217*, 2024. doi: 10.48550/arXiv.2408.09217.
27. M. Gao, P. Wu, and L. Pan. MINES: Multi-perspective API call sequence behavior fusion malware classification. In *Proc. 29th International Conference on Database Systems for Advanced Applications (DASFAA 2024)*, Lecture Notes in Computer Science, volume 14853, pages 210–220. Springer, 2024. doi: 10.1007/978-981-97-5562-2_13.
28. J. Lin, C. Lyu, X. Fan, and C. Dong. malDetect: Malware Classification Using API Sequence and Comparison with Transformer Encoder. In *Proc. 2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom)*, pages 133–140, 2024. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00107.
29. R. Kharsa, F. Kurugollu, A. Anjum, A. Amira, and A. Bouridane. Malware family classification with explainable BERT (xBERT) using API calls. In *Proc. 2024 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, pages 324–333, 2024. doi: 10.1109/BDCAT63179.2024.00057.
30. X. Yun, Y. Zhang, S. Liu, Z. Li, and W. Wang. API2Vec++: Boosting API sequence representation for malware detection and classification. *IEEE Transactions on Software Engineering*, 50(8):2142–2162, 2024. doi: 10.1109/TSE.2024.3422990.
31. G.-W. Wong, Y.-T. Huang, Y.-R. Guo, Y. Sun, and M. C. Chen. Attention-based API locating for malware techniques. *IEEE Transactions on Information Forensics and Security*, 19:1199–1212, 2024. doi: 10.1109/TIFS.2023.3330337.
32. P. Kunwar, K. Aryal, M. Gupta, M. Abdelsalam, and E. Bertino. SoK: Leveraging transformers for malware analysis. *arXiv preprint arXiv:2405.17190*, 2025.
33. A. H. Alhazmi. A robust and dynamic malware detection and classification model using behavioral-based analysis and BERT technique. *PLoS ONE*, 20(9):1–17, Sep. 2025. doi: 10.1371/journal.pone.0327604.
34. P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury. EarlyMalDetect: A novel approach for early Windows malware detection based on sequences of API calls. *arXiv preprint arXiv:2407.13355*, 2024.
35. Z. Jamadi and A. G. Aghdam. Early malware detection and next-action prediction. *arXiv preprint arXiv:2306.06255*, 2023.
36. S. Ö. Arik and T. Pfister. TabNet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021. doi: 10.1609/aaai.v35i8.16826.