JCTA Journal of Computing Theories and Applications ISSN:3024-9104

Research Article

Behavioral Malware Detection via API Call Sequences: A Comparative Study of LSTM and Transformer Architectures Using NLP-Inspired Representations

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

School of Computer Science and Artificial Intelligence, VIT Bhopal University, Bhopal, Madhya Pradesh 466114, India; e-mail : anunair0603@gmail.com; narottamdaspatel@vitbhopal.ac.in; saravanan.d@vitbhopal.ac.in; adarsh.patel@vitbhopal.ac.in

* Corresponding Author : Anusree K J

Abstract: The increasing sophistication of malware has rendered traditional signature-based detection methods insufficient, necessitating behavior-driven and adaptive analytical frameworks. This study presents a sequential deep learning framework that models system-level API call sequences as structured linguistic representations for behavioral malware detection. Unlike conventional comparative studies, this work systematically evaluates recurrent and attention-based architectures under controlled experimental conditions, with a particular focus on generalization performance and overfitting mitigation. Two neural architectures, a Long Short-Term Memory (LSTM) network and a Transformer-based attention model, are trained on publicly available API call sequence data for binary classification of malicious and benign executables. Beyond standard accuracy metrics, the study further examines model stability, convergence behavior, and the impact of long-range dependency modeling on detection robustness. Experimental results demonstrate that the Transformer architecture achieves superior performance, attaining 95.54% classification accuracy and consistent improvements in precision, recall, and F1-score, indicating a stronger ability to capture complex behavioral dependencies. These findings highlight the effectiveness of attention mechanisms in behavioral malware modeling and provide empirical evidence that NLP-inspired architectures offer a robust and scalable approach for real-world cybersecurity applications.

Keywords: Behavioral malware detection; Cybersecurity; Deep learning; Natural language processing; Sequence modeling; Software security; Sustainable digital security; Transformer.

Received: February, 26th 2026
Revised: March, 30th 2026
Accepted: March, 31st 2026
Published: April, 3rd 2026

1. Introduction

Traditional cybersecurity architectures face significant challenges due to the increasing volume and sophistication of malicious software (malware) [1]. 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, now characterized by obfuscation, polymorphism, and continuous mutation. Discriminating malicious intent from benign behavior [2] and identifying the functional logic of a malware sample remain critical challenges in cybersecurity.

A malware family is a group of malicious entities that share a common codebase or ancestry, often associated with the same threat actor. Although these variants introduce superficial syntactic changes to evade static analyzers, they exhibit consistent execution behavior [3]. Accurately identifying a new threat as malicious is therefore not only a detection task but also a crucial component of incident response, enabling security teams to prioritize threats, predict potential attacks, and formulate mitigation strategies [4].

File system operations, registry modifications, and network communications are captured through interactions between the operating system and the program via Application Programming Interface (API) calls. These API calls are difficult to obfuscate because they are essential for malware to achieve its objectives [5]. Notably, API call traces exhibit

structural similarities to natural language: they consist of a vocabulary of API functions, follow a specific syntax (execution order), and derive meaning from contextual relationships. For instance, a single RegOpenKey call may be benign in isolation but becomes suspicious when followed by encryption-related calls. By treating these sequences as a "behavioral language," researchers have applied Natural Language Processing (NLP) techniques to infer malicious intent [6]–[8].

1.1. Problem Statement

Developing models capable of effectively processing high-dimensional sequential data is essential for automated malware analysis pipelines [9]. Due to dataset constraints, this study focuses on binary malware detection, serving as a foundational step toward future multi-family classification. Effective models must capture both long-term dependencies (global execution context) and local patterns (short-range API interactions) [10]. However, achieving a balance between expressiveness (modeling complex behaviors) and robustness (handling noise) remains a persistent challenge.

Current approaches often rely on surface-level representations that fail to capture deep temporal dependencies inherent in malware behavior. API sequences are frequently modeled using a "bag-of-words" representation [11], which removes temporal order and limits behavioral interpretation. While classical machine learning (ML) methods such as Support Vector Machines (SVM) and Random Forest (RF) remain interpretable, many deep learning (DL) approaches lack systematic evaluation under controlled conditions [12]. There is therefore a need for structured analysis of how different architectures exploit the sequential nature of cybersecurity data, particularly under constraints such as long sequences and limited data availability [13]. This study addresses these challenges by investigating how sequence-based models can achieve accurate and stable malware detection while maintaining robustness and generalization within the dataset, without relying on manual feature engineering.

1.2. Why Natural Language Processing

Traditional malware detection methods rely heavily on handcrafted features and signature-based techniques, which are increasingly ineffective against polymorphic and obfuscated threats [4]. As a result, behavioral analysis has emerged as a more robust alternative, focusing on runtime execution patterns that are harder to evade [14].

API call sequences provide a natural sequential representation of program behavior [11]. Each API call corresponds to a system-level action (e.g., registry access, memory allocation, file operations, or network communication), and collectively these sequences form a behavioral signature [13]. NLP techniques are inherently designed to model contextual sequential data, making them well-suited for this problem.

By mapping API calls to tokens and execution traces to sentences, malware detection can be formulated as a sequence classification task. To justify this formulation, NLP-based models are compared with alternative approaches:

- Signature-based detection relies on static patterns and fails against polymorphic malware [1].
- N-gram and bag-of-words models with classical ML classifiers (e.g., SVM, RF) capture local co-occurrence patterns but ignore temporal order [11].
- Convolutional Neural Networks (CNN) capture local patterns but struggle with long-range dependencies due to limited receptive fields [10].
- Graph-based approaches model structural relationships but introduce significant preprocessing overhead and complexity.

In contrast, NLP-based sequential models—particularly Long Short-Term Memory (LSTM) and Transformer architectures—provide a balanced solution: they preserve temporal order, scale to long sequences, and enable end-to-end learning without manual feature engineering. These models can automatically extract discriminative behavioral patterns while capturing contextual relationships within API sequences [15]. Consequently, NLP-driven approaches offer an expressive and scalable paradigm for malware detection, with strong potential for extension to multi-family 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) for API call-based malware detection.
- To evaluate and compare the performance of LSTM and Transformer models under identical experimental conditions.
- To analyze the effectiveness of API call sequences in capturing behavioral features that distinguish malicious from benign software.
- To determine which architecture provides superior generalization, accuracy, and robustness.
- To establish a baseline for future multi-family classification tasks.

1.4. Main Contributions

The main contributions of this work are summarized as follows:

- A rigorous controlled comparison of LSTM and Transformer architectures for API-based malware detection under identical experimental conditions, including unified preprocessing, consistent hyperparameter tuning, and identical dataset splits. This addresses limitations in prior studies where inconsistent evaluation setups often confound comparisons.
- Empirical evidence demonstrating the effectiveness of Transformer-based architectures in modeling long-range dependencies in API call sequences for malware detection.
- A cohesive preprocessing pipeline that transforms raw API sequences into structured representations suitable for DL models.
- A comprehensive evaluation using multiple metrics, including accuracy, recall, and F1-score, supported by cross-validation for robust assessment.
- An in-depth error analysis highlighting challenges in distinguishing malicious and benign behaviors, with implications for future family-level classification.

The remainder of this paper is organized as follows. Section 2 reviews related work on NLP-based malware detection and classification. Section 3 describes the proposed framework and preprocessing pipeline. Section 4 details the experimental setup. Section 5 presents the results and discussion, including performance evaluation and analysis. Finally, Section 6 concludes the paper and outlines future research directions.

2. Literature Review

The field of automated threat intelligence has evolved significantly, transitioning from static heuristic techniques to deep learning-based modeling. This section reviews major developments in malware detection, with particular emphasis on classification approaches and their extension toward malware family analysis.

2.1. Malware Classification Approaches

Signature-based detection, which compares file hashes against a database of known threats, has long been a fundamental component of cybersecurity systems [5]. While effective against known malware, such approaches fail to detect polymorphic and metamorphic variants that continuously modify their code to evade detection [1]. As a result, research has shifted toward behavioral detection methods that analyze program execution in controlled environments [16]. By examining runtime characteristics, particularly API call sequence, modern approaches aim to identify previously unseen malware through pattern recognition and anomaly detection.

2.2 API Call-Based Malware Analysis

API calls have emerged as a key feature for behavioral fingerprinting [4]. Prior studies demonstrate that API call sequences provide a robust representation of malicious activity, as these interactions are difficult to modify without affecting program functionality [17]. Recent work models these sequences as "behavioral transcripts," in which the contextual relationships between calls determine their significance. This perspective reinforces the suitability of sequence-based modeling for capturing underlying behavioral patterns.

2.3 Machine Learning Models

Early research applied traditional machine learning techniques and text-processing methods to automate malware detection [18]. Feature extraction strategies such as n-grams, decision trees, and bag-of-words representations were widely used on API sequences [1]. Kolter and Maloof (2006) demonstrated that malicious intent can be inferred from such features. However, these approaches often ignore temporal order and sequential dependencies, leading to the loss of execution context and increased feature dimensionality. This limitation restricts their ability to capture fine-grained behavioral patterns.

2.4 .Deep Learning and NLP-Based Models

The adoption of NLP techniques has significantly advanced sequence-based malware analysis. Recurrent models, particularly LSTM networks, were introduced to capture temporal dependencies and address the vanishing gradient problem [9]. Studies such as Wang and Yiu [9] and Kolosnjaji et al. [10] showed that LSTM models effectively capture execution history and outperform traditional approaches. More recently, Transformer-based architectures have demonstrated strong performance in sequence modeling tasks [19]. Unlike recurrent models, Transformers use multi-head self-attention to model relationships between API calls regardless of their relative positions, enabling more effective learning of long-range dependencies.

2.5. Research Gap

Despite significant progress, a clear gap persists in the systematic, controlled comparison of deep learning architectures for malware detection [20]. While malware family classification is the ultimate objective, the foundational task of binary classification based on API call sequences warrants further investigation as a baseline for more complex scenarios[21]. Recent studies in cybersecurity—such as transformer-based teacher–student models for phishing detection [8], generative adversarial network (GAN)-based data augmentation for intrusion detection [12], and reinforcement learning for adaptive cyber defense [22]—highlight the growing need for robust and well-evaluated architectures. However, direct and fair comparisons between sequence models under identical experimental conditions remain limited.

This study addresses this gap by conducting a controlled comparative evaluation of LSTM and Transformer architectures for binary malware detection using API call sequences within a unified framework. It is important to clarify that the overall pipeline—API sequence extraction, tokenization, embedding, and sequence modeling—follows established approaches similar to Aggarwal and Di Troia [5] and Demirkıran et al. [23]. The contribution of this work lies not in proposing a new pipeline, but in providing a rigorously controlled comparison under identical preprocessing, hyperparameter settings, and dataset partitions. The use of a publicly available benchmark dataset, also employed in prior studies [4], enables partial cross-study comparison. A direct comparison with existing work is presented in the Results and Discussion section to contextualize the performance achieved in this study.

3. Proposed Method

This section describes the end-to-end pipeline used for behavioral malware detection. The system follows a structured sequence of steps: (1) raw API call sequences are extracted; (2) sequences undergo normalization, cleaning, tokenization, and padding; (3) tokens are mapped into dense vector representations through a learnable embedding layer; (4) embeddings are processed by either an LSTM encoder or a Transformer encoder; and (5) a classification head produces a binary prediction (malicious or benign). The overall workflow is illustrated in Figure 1, which summarizes the transformation from raw behavioral traces to final classification output.

3.1. Dataset Preprocessing

3.1.1. Dataset Description

This study utilizes the publicly available dataset "Malware Analysis Datasets: API Call Sequences" [10], which contains behavioral profiles derived from Windows API call

sequences [5]. The dataset includes diverse malware categories—Backdoor, Spyware, Worm, Trojan, Virus, Rootkit, Ransomware, and Adware—along with benign samples.

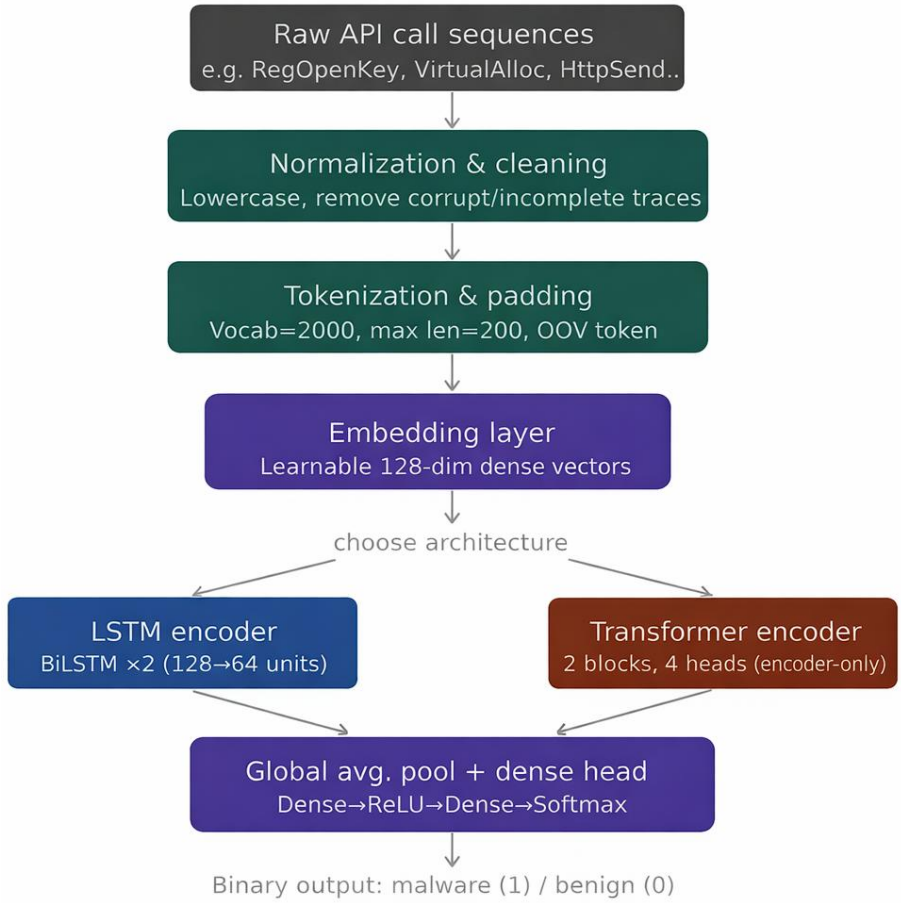

Figure 1. End-to-end pipeline for API call-based malware detection, including preprocessing, embedding, sequence modeling (LSTM/Transformer), and classification.

The dataset comprises approximately 58,507 samples. Based on the test partition (15% split), the class distribution is approximately 56.0% benign (4,658 samples) and 44.0% malware (4,118 samples), indicating a mild class imbalance. To address this, stratified sampling is applied across all partitions, and per-class metrics (precision, recall, and F1-score) are reported alongside overall accuracy.

Scope limitation. This study performs binary classification (malicious vs. benign) using the provided labels. Although the dataset contains multiple malware families, they are consolidated into a single "malware" class to establish a strong baseline. Consequently, intra-family behavioral variations are not explicitly modeled, and multi-family classification is deferred to future work.

The dataset has been previously used in API-call-based malware studies, including Catak et al. [4], enabling partial cross-study comparisons. It is structured as a tabular dataset with two attributes: API call sequences and corresponding labels. Sequence lengths range from 15 to 200 API calls, capturing execution-level behavioral patterns.

3.1.2. API Call Sequence Extraction and Normalization

Preprocessing begins with extracting raw API call sequences while preserving their original order, as temporal structure is essential for capturing behavioral patterns. Each API call is treated as a distinct token representing system-level operations such as file handling, registry access, process control, and network communication. To ensure consistency, API names are normalized by removing case variations and standardizing naming conventions. This step mitigates inconsistencies arising from differences in system environments or documentation, ensuring that semantically equivalent API calls are mapped to the same token representation [24]

3.1.3. Data Cleaning and Quality Assurance

Data cleaning procedures are applied to improve dataset reliability. Incomplete or corrupted API traces—such as those resulting from interrupted execution or insufficient sandbox monitoring—are removed to prevent ambiguity during training. Class distribution is further examined to ensure balanced representation across partitions. Stratified sampling is used to maintain a proportional class distribution across the training and validation sets, reducing potential bias and improving model generalization.

3.1.4. Tokenization and Numerical Encoding

Tokenization is performed using the Keras Tokenizer [15], converting API sequences into integer-based representations. To balance expressiveness and computational efficiency, the vocabulary is limited to 2,000 unique API tokens. Out-of-vocabulary (OOV) tokens are introduced to handle rare or unseen API calls.

Sequences are standardized to a fixed length of 200 tokens through truncation and padding. Longer sequences are truncated to preserve computational feasibility, while shorter sequences are padded at the end to maintain structural consistency. This results in input matrices of size (number of samples × 200), where each element corresponds to a token index. Label encoding is performed using the LabelEncoder from scikit-learn [15], transforming textual class labels into numerical form suitable for model training while preserving the mapping between encoded values and original labels.

3.2. Problem Formulation

A supervised sequence classification approach is employed to distinguish between malicious and benign software. Given an input sequence of API calls:

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

where x_i denotes 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 $f: X \rightarrow y$ where $y \in \{0,1\}$, with 0 indicating benign samples and 1 indicating malicious samples. To optimize the model, the categorical cross-entropy loss function is minimized:

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

where θ represents the model parameters, N is the number of training samples, C denotes the number of classes (in this case, $C = 2$), $y_{i,c}$ is the ground-truth indicator (0 or 1) for sample i belonging to class c, $\hat{y}_{i,c}$ is the predicted probability for sample i belonging to class c. For the binary classification setting (C = 2), this formulation is equivalent to the binary cross-entropy loss commonly used in two-class problems.

For model training and evaluation, the dataset is split into training, validation, and test sets at 70:15:15. Stratified sampling is applied to preserve class distribution across all partitions, ensuring balanced representation and reducing sampling bias.

3.3. Natural Language Processing Models for Sequence Classification

This section describes the two sequence modeling architectures employed in this study: a bidirectional LSTM network and an encoder-only Transformer model. Both architectures operate on the same preprocessed input representations and are evaluated under identical experimental conditions to ensure a fair comparison.

3.3.1. Long Short-Term Memory (LSTM) Network

A bidirectional LSTM architecture is used as the baseline model for sequential data processing. The model consists of three main components:

- Embedding Layer: API tokens are mapped into dense vector representations using a learnable embedding matrix with a dimensionality of 128. This representation captures semantic relationships between API functions.

- Sequence Processing Layers: The embedded sequences are processed using two stacked bidirectional LSTM layers with 128 and 64 hidden units, respectively. Bidirectional processing enables the model to capture contextual information from both forward and backward directions, improving the representation of sequential dependencies.
- Classification Head: The output representations are aggregated using global average pooling, followed by two fully connected layers with ReLU activations. A final softmax layer produces class probabilities for binary classification [25].

The architecture of the LSTM model is illustrated in Figure 2, while the complete hyperparameter configuration for both models is summarized in Table 2.

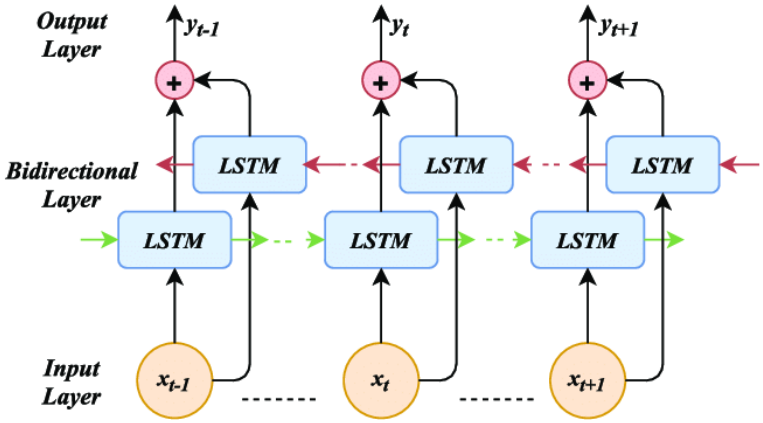

Figure 2. Bidirectional LSTM architecture with stacked layers and global average pooling for sequence classification.

To mitigate overfitting, regularization techniques are applied, including L2 weight regularization $(\lambda = 0.001)$ and recurrent dropout (30%). Model optimization is performed using the Adam optimizer with a learning rate of 0.0005.

3.3.2. Transformer-Based Architecture

An encoder-only Transformer architecture is employed for sequence modeling [19]. The decoder and cross-attention components of the original Transformer are not used, as the task is sequence classification rather than sequence generation. The overall architecture of the model is illustrated in Figure 3.

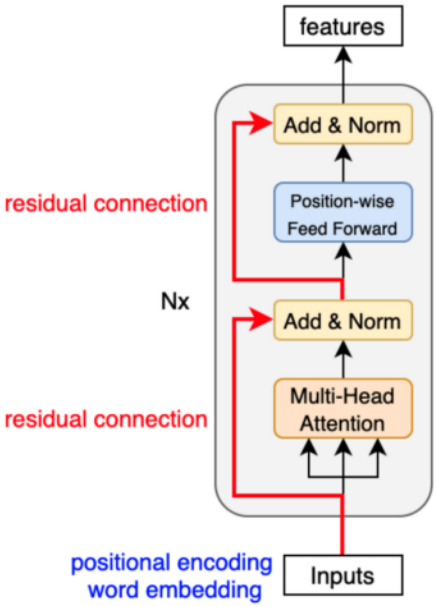

Figure 3. Encoder-only Transformer architecture with multi-head self-attention, residual connections, and feed-forward layers for sequence classification.

The model consists of the following components:

- Input Representation: Token embeddings are combined with positional embeddings to encode both semantic and temporal information within API sequences.
- Multi-Head Self-Attention: Two Transformer blocks are used, each with four attention heads. This mechanism enables the model to capture relationships across different representation subspaces and identify long-range dependencies within sequences.
- Feed-Forward Network: A position-wise feed-forward network with ReLU activations is applied after the attention mechanism to enhance feature transformation.
- Residual Connections and Normalization: Residual connections are applied around each sub-layer to facilitate gradient flow, while layer normalization stabilizes training.
- Classification Module: The sequence representation is aggregated using global average pooling and passed through dense layers before final softmax classification.

Regularization is applied using dropout rates of 30% in attention layers and 50% in dense layers. The model is optimized using the Adam optimizer with a learning rate of 0.0003, adjusted to accommodate the higher model complexity. Both architectures are implemented using TensorFlow 2.0 and the Keras API, with hyperparameters selected based on validation performance.

4. Experimental Setup

This section describes the experimental setup used to evaluate the proposed sequence models, including training configuration, hardware environment, and implementation details. The experiments are designed to ensure a fair comparison between architectures under consistent conditions. The evaluation focuses on binary classification of malicious and benign samples using API call sequences. The preprocessing pipeline and model architectures follow the framework described in Section 3.

4.1. Training Configuration

Training deep learning models on API sequence datasets involves high-dimensional embeddings and long sequence lengths, which require substantial computational resources [18]. Careful configuration of training parameters is therefore essential to ensure both efficiency and model stability.

4.1.1. Hardware

Processing long API sequences and training sequence models impose significant computational demands. The experiments were conducted using the following hardware configuration:

- Processor (CPU): Multi-core processors such as Intel Xeon or Core i7/i9 were used to support parallel data preprocessing and pipeline execution.
- GPU Acceleration: Training was performed using NVIDIA GPUs (e.g., Tesla V100, RTX 3090, or A100) with CUDA support, enabling efficient handling of large tensor operations in both LSTM and Transformer models.
- Memory (RAM): System memory ranging from 32 GB to 64 GB was used to ensure efficient loading and processing of large batches of sequential data without bottlenecks.

4.1.2. Framework and Implementation

The models were implemented using widely adopted deep learning frameworks. Neural architectures were developed using TensorFlow/Keras and PyTorch [18], while attention-based components were supported using the Hugging Face Transformers library. Data preprocessing and manipulation were performed using NumPy and Pandas, and Scikit-learn was used for dataset partitioning and evaluation metrics.

Hyperparameters were selected based on validation performance to balance model expressiveness and generalization while mitigating overfitting. The key training settings are summarized below:

- Learning Rate: The Adam optimizer was used with an initial learning rate in the range of 10^{-14} to 10^{-13}, with adjustments during training as needed.
- Batch Size: Batch sizes ranged from 32 to 128, depending on available GPU memory and sequence length.

- Sequence Length: All experiments used a fixed sequence length of 200 tokens, balancing computational efficiency and performance.
- Loss Function: Categorical cross-entropy was used for optimization, consistent with the binary classification formulation described in Section 3.
- Embedding Dimension: API sequences were projected into dense vector spaces with dimensions typically set to 128; additional experiments considered higher dimensions (e.g., 256 and 768).

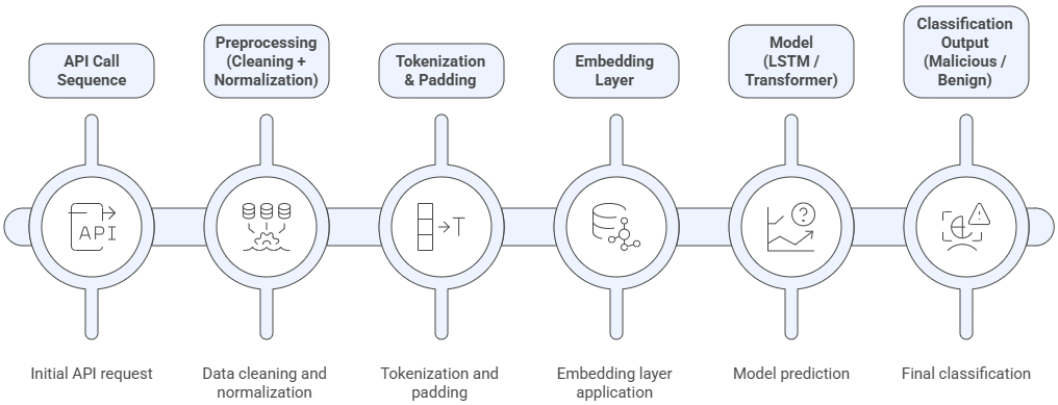

Figure 4. Overview of the experimental workflow, illustrating the data flow from API call sequences through preprocessing, embedding, model inference, and final classification.

5. Results and Discussion

This section presents the experimental results for binary malware detection using API call sequences. All reported performance metrics are based on the held-out test set, ensuring an unbiased evaluation of model generalization. Training and validation sets are used exclusively for model optimization and early stopping. In addition to the LSTM and Transformer models, four classical baselines are evaluated: Logistic Regression (LR), Support Vector Machine (SVM with RBF kernel, $C = 10$), Random Forest (RF, 200 estimators), and XGBoost. Classical models use TF-IDF-weighted unigram and bigram features (top 5,000). Furthermore, 5-fold cross-validation is applied to assess model stability.

5.1. Performance Evaluation on Test Set

Model performance is evaluated using accuracy, precision, recall, F1-score, and balanced accuracy. In the context of malware detection, recall for the malware class is particularly critical, as false negatives pose higher security risks than false positives. Table 1 summarizes the performance of all evaluated models.

Table 1. Overall performance comparison of classical and deep learning models.

Model	Accuracy (%)	Precision	Recall	F1-Score
Logistic Regression (LR)	74.1	0.743	0.741	0.742
Support Vector Machine (SVM)	78.6	0.788	0.786	0.787
Random Forest (RF)	81.3	0.814	0.813	0.813
XGBoost	80.7	0.808	0.807	0.807
LSTM	90.58	0.9069	0.9058	0.9058
Transformer	95.54	0.9559	0.9554	0.9555

The results show that deep learning models significantly outperform classical approaches, with improvements exceeding 14 percentage points in accuracy. The Transformer achieves the best performance, surpassing the LSTM by 4.96 percentage points (5.48% relative improvement) [26]. This performance gain is attributed to the Transformer's self-attention mechanism, which captures long-range dependencies across API sequences more effectively than sequential processing in LSTM models.

To contextualize these findings, Table 2 compares the results with prior studies that used similar API-call-based pipelines.

Table 2. Comparison with prior API call-based malware detection studies.

Study	Method	Dataset	Task	Best F1-Score
Aggarwal and Di Troia [5]	API embeddings + Transformer	Catak [4]	Multi-class	0.5106
Demirkıran et al. [23]	Ensemble of pre-trained Transformers	Oliveira [1]	Multi-class	0.5650
Transformer (Ours)	Encoder-only Transformer	Oliveira [1] (binary)	Binary	0.9555
LSTM (Ours)	Bidirectional LSTM	Oliveira [1] (binary)	Binary	0.9058

Direct numerical comparison is limited by differences in datasets and task formulations (binary vs. multi-class). Prior studies focus on multi-class malware family classification, which is inherently more challenging and typically results in lower performance metrics. Nevertheless, the results demonstrate that the proposed approach achieves strong performance within its defined scope.

5.2. Training Stability and Convergence Analysis

To assess model stability, 5-fold cross-validation results are summarized in Table 3.

Table 3. Cross-validation results (mean ± standard deviation).

Model	Accuracy (%) (Mean ± Std)
Logistic Regression (LR)	74.0 ± 0.8
Support Vector Machine (SVM)	78.4 ± 0.7
Random Forest (RF)	81.1 ± 0.5
XGBoost	80.5 ± 0.6
LSTM	90.3 ± 0.6
Transformer	95.2 ± 0.4

The Transformer exhibits both higher accuracy and lower variance, indicating more stable performance across different data splits. Early stopping (patience = 3) is applied based on validation loss, and reported test results correspond to the best-performing checkpoint.

5.3. Class-Level Analysis and Error Characteristics

Class-level performance is summarized in Table 4.

Table 4. Class-wise performance comparison on the test set.

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

The Transformer consistently outperforms the LSTM across both classes. Notably, it achieves higher malware recall (0.97), significantly reducing false negatives. This is critical in cybersecurity applications, where missed detections can lead to severe consequences. The confusion matrices for both models are shown in Figure 5 and Figure 6. From the confusion matrices, the Transformer reduces false negatives (133 vs. 344 in LSTM) while also lowering false positives, demonstrating a more reliable decision boundary.

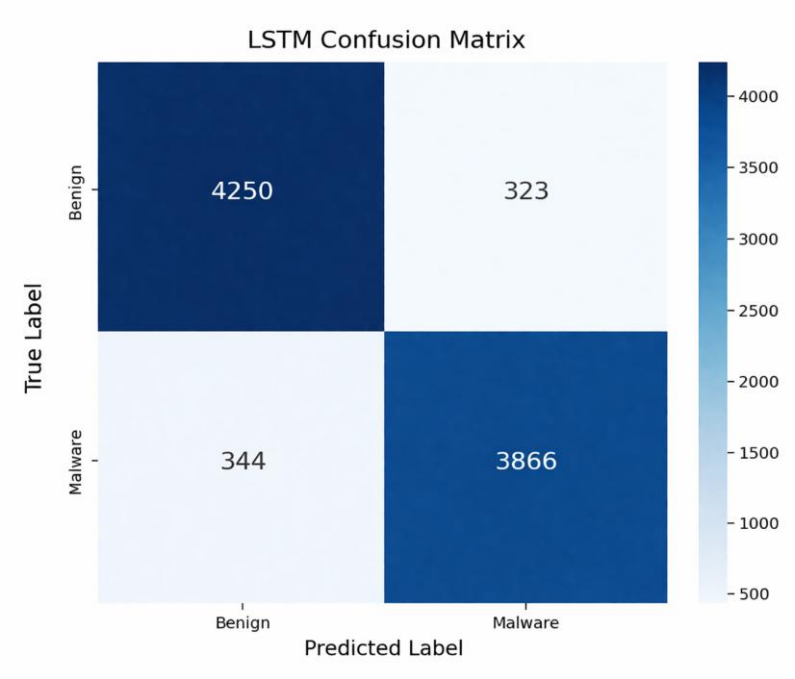

Figure 5. Confusion matrix of the LSTM model.

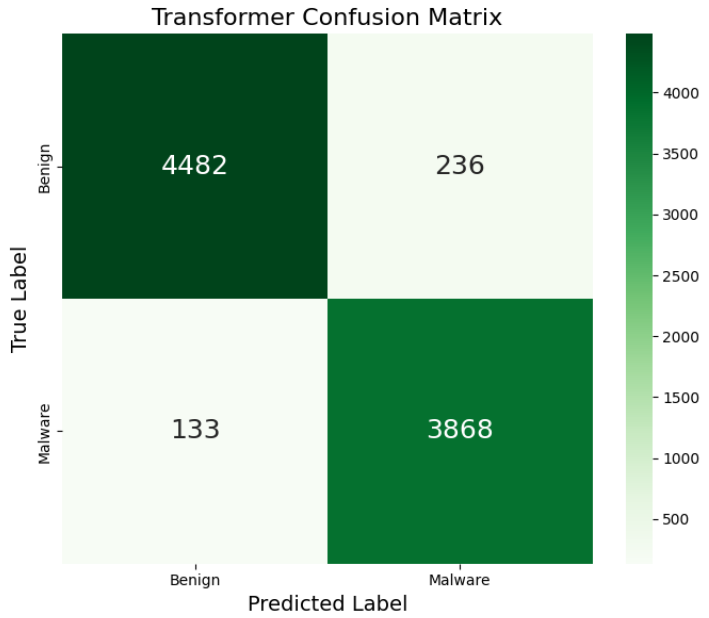

Figure 6. Confusion matrix of the Transformer model.

5.4. Family-Wise Classification Insights

Although this study focuses on binary classification, additional insights can be drawn from the underlying distribution of malware families. Different malware families exhibit distinct behavioral patterns, which may influence classification performance. Malware such as ransomware and spyware tends to generate strong, consistent API call patterns (e.g., encryption routines or network communication), making them relatively easier to detect. In contrast, other categories—such as adware or benign-like utilities—often share overlapping behavioral characteristics, increasing the likelihood of misclassification.

This observation suggests that the reported performance is influenced not only by model capability but also by the separability of behavioral patterns across malware families. While the binary formulation simplifies the classification task, it may obscure fine-grained distinctions between families. These findings are consistent with prior studies indicating that API

call sequences provide discriminative behavioral signatures, but their effectiveness depends on the granularity of the classification task and the diversity of malware behaviors. Consequently, extending the approach to multi-family classification remains an important direction for future work.

5.5. Discussion

The experimental results demonstrate that the Transformer consistently outperforms the LSTM in binary malware detection, achieving a 4.96% absolute improvement in accuracy (95.54% vs. 90.58%). This improvement highlights the limitations of recurrent architectures in modeling long-range dependencies within API call sequences.

5.5.1. Advancements Over Sequential Models

Cross-validation results confirm stable performance across different data splits, with the Transformer achieving 95.2±0.4% and the LSTM 90.3±0.6%. Additional experiments further emphasize the importance of sequence modeling. Reducing the sequence length from 200 to 50 tokens results in a 4.8% performance degradation for the Transformer and a 3.2% degradation for the LSTM. Similarly, randomizing API call order reduces performance by 6.2% and 4.1%, respectively. These results confirm that both temporal order and long-range dependencies are critical for accurate malware detection. These findings align with prior work demonstrating that attention-based architectures are more effective in capturing long-range dependencies in sequential data [19]. In contrast, sequential processing and gradient limitations inherently constrain recurrent models.

5.5.2. Interpretability and Behavioral Alignment

An additional advantage of the Transformer lies in its interpretability. Attention mechanisms enable the model to highlight API calls that most significantly contribute to classification decisions [27]. Examples of such patterns include:

- CreateRemoteThread and VirtualAlloc for code injection,
- RegSetValue for persistence mechanisms,
- HttpSendRequest for data exfiltration.

These patterns are consistent with known malware behaviors, indicating that the model captures meaningful semantic relationships rather than relying solely on statistical correlations. This alignment improves interpretability and supports practical adoption in cybersecurity workflows.

5.5.3 Practical Implications and Limitations

The results indicate strong performance within the evaluated dataset; however, generalization across different datasets or platforms is not explicitly validated. Similar limitations have been noted in prior studies, where dataset-specific characteristics influence model performance [9]. Behavior-based detection is robust against obfuscation and polymorphism because it relies on execution semantics rather than static signatures. However, deployment in real-world environments requires further validation under diverse and evolving threat scenarios. Overfitting is mitigated through early stopping (patience = 3) and dropout regularization (0.3 in attention layers and 0.5 in dense layers). The reported results correspond to the best validation checkpoint, ensuring reliable evaluation.

6. Conclusion

This study presents a controlled comparative evaluation of sequence-based architectures for behavioral malware detection using API call sequences. The results demonstrate that the Transformer consistently outperforms the LSTM across accuracy, stability, and error reduction. Its ability to capture long-range dependencies plays a key role in improving detection performance, particularly in reducing false negatives. Despite these advantages, several limitations remain. The evaluation is conducted on a single dataset, which may not fully represent the diversity of real-world malware. Additionally, the Transformer's higher model complexity may pose challenges for deployment in resource-constrained environments.

Future work will focus on extending the approach to multi-family classification, evaluating cross-dataset generalization, and exploring hybrid architectures that integrate sequence modeling with structural representations. Further investigation into adversarial robustness and real-time deployment is also required. Overall, the findings suggest that attention-based

architectures offer a scalable, effective solution for behavioral malware detection, with strong potential for practical cybersecurity applications.

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.

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.

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 conflict of interest

References

[1] A. Olivera, "Malware Analysis Datasets: API Call Sequences," *Kaggle.com*, 2020. 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 Comput. Sci.*, vol. 6, p. e285, Jul. 2020, doi: 10.7717/peerj-cs.285.

[3] M. S. Masari, M. A. Danladi, I. L. Onyinye, and L. K. Tohomdet, "Android Malware Detection Using Machine Learning with SMOTE-Tomek Data Balancing," *J. Comput. Theor. Appl.*, vol. 3, no. 3, pp. 302–313, Jan. 2026, doi: 10.62411/jcta.15084.

[4] F. O. Catak and A. F. Yazı, "A Benchmark API Call Dataset for Windows PE Malware Classification," *ArXiv*. Feb. 21, 2021. [Online]. Available: http://arxiv.org/abs/1905.01999

[5] S. Aggarwal and F. Di Troia, "Malware Classification Using Dynamically Extracted API Call Embeddings," *Appl. Sci.*, vol. 14, no. 13, p. 5731, Jun. 2024, doi: 10.3390/app14135731.

[6] A. H. Alhazmi, "A robust and dynamic malware detection and classification model using behavioral-based analysis and BERT technique," *PLoS One*, vol. 20, no. 9, p. e0327604, Sep. 2025, doi: 10.1371/journal.pone.0327604.

[7] A. S. Kale, F. Di Troia, and M. Stamp, "Malware Classification with Word Embedding Features," *ArXiv*. Mar. 03, 2021. [Online]. Available: http://arxiv.org/abs/2103.02711

[8] P. H. Hussan and S. M. Mangj, "BERTPHIURL: A Teacher-Student Learning Approach Using DistilRoBERTa and RoBERTa for Detecting Phishing Cyber URLs," *J. Futur. Artif. Intell. Technol.*, vol. 1, no. 4, pp. 417–428, Feb. 2025, doi: 10.62411/faith.3048-3719-71.

[9] X. Wang and S. M. Yiu, "A multi-task learning model for malware classification with useful file access pattern from API call sequence," *ArXiv*. Oct. 19, 2016. [Online]. Available: http://arxiv.org/abs/1610.05945

[10] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep Learning for Classification of Malware System Call Sequences," in *Lecture Notes in Computer Science*, 2016, pp. 137–149. doi: 10.1007/978-3-319-50127-7_11.

[11] B. K. Mamade and D. M. Dabala, "Exploring The Correlation between Cyber Security Awareness, Protection Measures and the State of Victimhood: The Case Study of Ambo University's Academic Staffs," *J. Cyber Secur. Mobil.*, Jun. 2021, doi: 10.13052/jcsm2245-1439.1044.

[12] M. A. Rahman, G. A. Francia, and H. Shahriar, "Leveraging GANs for Synthetic Data Generation to Improve Intrusion Detection Systems," *J. Futur. Artif. Intell. Technol.*, vol. 1, no. 4, pp. 429–439, Feb. 2025, doi: 10.62411/faith.3048-3719-52.

[13] M. Ahmed, A. Qureshi, J. Ahmed Shamsi, and M. Marvi, "Sequential Embedding-based Attentive (SEA) classifier for malware classification," in *2022 International Conference on Cyber Warfare and Security (ICCWS)*, Dec. 2022, pp. 28–35. doi: 10.1109/ICCWS56285.2022.9998431.

[14] 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," *CEUR Workshop Proc.*, vol. 3260, pp. 124–134, 2022.

[15] Z. Zhang, P. Qi, and W. Wang, "Dynamic Malware Analysis with Feature Engineering and Feature Learning," *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 01, pp. 1210–1217, Apr. 2020, doi: 10.1609/aaai.v34i01.5474.

[16] C. Avci, B. Tekinerdogan, and C. Catal, "Analyzing the performance of long short-term memory architectures for malware detection models," *Concurr. Comput. Pract. Exp.*, vol. 35, no. 6, pp. 1–1, Mar. 2023, doi: 10.1002/cpe.7581.
[17] T. Quertier, B. Marais, G. Barrué, S. Morucci, S. Azé, and S. Salladin, "A Lean Transformer Model for Dynamic Malware Analysis and Detection," *ArXiv*. Aug. 05, 2024. [Online]. Available: http://arxiv.org/abs/2408.02313
[18] B. Marais, T. Quertier, and G. Barrue, "Semantic Preprocessing for LLM-based Malware Analysis," *ArXiv*. Oct. 03, 2025. [Online]. Available: http://arxiv.org/abs/2506.12113
[19] Jobanpreet Kaur *et al.*, "Comparative Analysis of Transformer and LSTM Architectures for Cybersecurity Threat Detection Using Machine Learning," *EAI Endorsed Trans. AI Robot.*, vol. 4, Sep. 2025, doi: 10.4108/airo.9759.
[20] A. Rahali and M. A. Akhloufi, "MalBERT: Using Transformers for Cybersecurity and Malicious Software Detection," *ArXiv*. Mar. 05, 2021. [Online]. Available: http://arxiv.org/abs/2103.03806
[21] A. Walker and S. Sengupta, "Malware Family Fingerprinting Through Behavioral Analysis," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Nov. 2020, pp. 1–5. doi: 10.1109/ISI49825.2020.9280529.
[22] A. P. Binitie, S. I. Onyemenem, N. C. Anujeonye, A. A. Ojugo, F. A. Egbokhare, and T. C. Aghaunor, "A Graph-Augmented Isolation Forest Using Node2Vec and GraphSAGE for Mobile User Behavior Anomaly Detection," *J. Comput. Theor. Appl.*, vol. 3, no. 3, pp. 369–383, Feb. 2026, doi: 10.62411/jcta.15494.
[23] F. Demirkıran, A. Çayır, U. Ünal, and H. Dağ, "An ensemble of pre-trained transformer models for imbalanced multiclass malware classification," *Comput. Secur.*, vol. 121, p. 102846, Oct. 2022, doi: 10.1016/j.cose.2022.102846.
[24] P. Gysel, C. Wüest, K. Nwafor, O. Jašek, A. Ustyuzhanin, and D. M. Divakaran, "EAGLEEYE: Attention to Unveil Malicious Event Sequences From Provenance Graphs," in *2024 APWG Symposium on Electronic Crime Research (eCrime)*, Sep. 2024, pp. 27–42. doi: 10.1109/eCrime66200.2024.00009.
[25] M. Gao, P. Wu, and L. Pan, "MINES: Multi-perspective API Call Sequence Behavior Fusion Malware Classification," in *Lecture Notes in Computer Science*, 2024, pp. 210–220. doi: 10.1007/978-981-97-5562-2_13.
[26] S. Abbas, S. Amjad, S. Craß, and S. A. Moeinzadeh Mirhosseini, "Analysis of Blockchain-IoT Connection Patterns based on Clients Type," in *2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics*, Aug. 2024, pp. 579–586. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00107.
[27] R. Kharsae, F. Kurugollu, A. Anjum, A. Amira, and A. Bouridane, "Malware Family Classification with Explainable BERT (xBERT) Using API Calls," in *2024 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, Dec. 2024, pp. 324–333. doi: 10.1109/BDCAT63179.2024.00057.