Cureus Journal of Computer Science
https://doi.org/10.7759/s44389-026-00046-6

Original Article

Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis

John Adejoh [1], Nsikak Owoh [2], Felix Ale [3], Prasad Rajesh [3], Moses Ashawa [2], Salaheddin Hosseinzadeh [2]

1. *Software Engineering, African University of Science and Technology, Abuja, NGA*
2. *Cyber Security and Networks, Glasgow Caledonian University, Glasgow, GBR*
3. *Computer Science, African University of Science and Technology, Abuja, NGA*

Received: February 13, 2026 | Review began: February 24, 2026 | Review ended: March 27, 2026 | Published: March 30, 2026

Abstract

The rise in sophisticated malware has exposed the limitations of traditional signature-based and static analysis methods, which often fail against obfuscated or evolving threats. Deep learning has shown promise in handling sequential data, such as API call sequences; yet, conventional models often struggle to capture critical subsequences within complex, variable-length inputs. This paper addresses this gap by systematically evaluating attention-augmented deep learning architectures for API sequence-based malware detection. We compare the performance of standard Convolutional Neural Network, Long Short-Term Memory (LSTM), and Gated Recurrent Unit models against their attention-augmented counterparts, while a Transformer serves as a robust baseline. Our results demonstrate that the attention mechanism consistently improves model robustness and interpretability. Crucially, attention-augmented models significantly reduced the False-Negative Rate (FNR), a critical metric in cybersecurity where missed detections are costly. The Attention-LSTM (A-LSTM) achieved the highest performance on Dataset 1, with an F1-score of 0.997 and an FNR of 0.001. Evaluated on the independent Mal-API-2019 test set, the A-LSTM demonstrated practical efficiency with an inference time of approximately 1.55 ms per sample and a memory footprint of 160 MB. Visual analysis of attention weights confirmed that the models focused on the most discriminative API calls. Overall, this work establishes a systematic evaluation framework that demonstrates the attention mechanism as a key component for improving the efficiency, robustness, explainability, and deployability of deep learning models in cybersecurity.

Categories: Network Security, Machine Learning (ML), Deep Learning

Keywords: malware detection, api call sequences, attention mechanism, deep learning, lstm, transformer, interpretability, anomaly detection, sequence modeling

Introduction

The increasing sophistication of cyber threats has driven a surge in research to develop intelligent, adaptive, and interpretable models for detecting malware and identifying anomalies in network traffic. Traditional machine learning approaches, though widely used, often fall short in capturing the complex temporal dependencies and evolving patterns characteristic of malicious behaviors, particularly when obfuscation and adversarial evasion techniques are employed [1]. Deep learning, especially sequence models such as Long Short-Term Memory (LSTM) networks, has emerged as a promising solution for handling sequential and time-dependent data, including Application Programming Interface (API) call sequences, system logs, and real-time sensor signals [2]. However, these models face limitations in selectively focusing on the most important parts of input sequences, which is critical for accurate detection and classification in cybersecurity contexts [3].

Recent advancements in attention mechanisms have introduced a paradigm shift in how deep learning models process and prioritize information. Attention-enhanced models can isolate and emphasize critical patterns by assigning dynamic weights to input features, even in the presence of noise or redundant data [4]. This capability is particularly beneficial in cybersecurity, where malicious behavior often manifests through specific, repeated patterns hidden within more extended, benign sequences. Studies across various domains, ranging from ransomware detection to IoT anomaly identification, have demonstrated that integrating Attention with LSTM or Convolutional Neural Network (CNN) architectures enhances both detection accuracy and model interpretability [5].

Despite these advancements, the current literature often focuses narrowly on specific model variants or data types, limiting the generalizability of findings. Comparative studies frequently benchmark attention-augmented models against traditional baselines without isolating the unique contributions of attention across different architectures. Moreover, many models are tailored to static or dynamic malware analysis, with few demonstrating robust performance across both approaches. In real-world deployment scenarios, especially those involving resource-constrained environments such as IoT gateways, computational efficiency becomes as critical as detection performance, necessitating architectures that balance accuracy, speed, and resource utilization [6].

This paper presents a comprehensive analysis of attention mechanisms in enhancing deep learning models for cybersecurity and anomaly detection, resulting in a novel LSTM + Attention model proposal. The model is designed to bridge the gap between static and dynamic analysis methods by learning both temporal dependencies and critical subsequence patterns in API call data. This has led to its evaluation across diverse tasks, including ransomware classification, network intrusion detection, and demand forecasting, highlighting its versatility, accuracy, and deployment viability. While prior studies have shown that attention can improve performance, a direct, controlled comparison isolating its effect across diverse architectures under identical conditions is lacking. This paper presents a comprehensive analysis designed to address this gap. Our primary contribution is a rigorous, empirical validation of the attention mechanism's utility, not merely for marginal gains in aggregate accuracy, but for substantially enhancing model robustness, evidenced by a marked reduction in false negatives and providing crucial interpretability. The contributions of this study are threefold:

1. We synthesize recent advances in sequence-based modeling for malware detection and anomaly prediction, with a focus on integrating attention mechanisms and comparing their performance across various domains.

2. We introduce a hybrid architecture that combines the sequential learning strengths of LSTM with the selective focusing power of attention, demonstrating its superiority over baseline models in recognizing both frequent and rare malicious patterns.

3. We explore trade-offs between model complexity and efficiency, providing recommendations for real-world applications, particularly in edge computing and time-sensitive environments.

Literature review

This section reviews the evolution of deep learning models for malware detection, with a focus on the progression from sequence-based models to those that integrate attention mechanisms. We synthesize literature across domains to highlight a key methodological gap in comparative studies.

Sequence-based deep learning for malware detection

Malware detection has increasingly leveraged sequence-based deep learning models, particularly those capable of handling sequential data such as API call sequences. API calls capture behavioral patterns, making them pivotal features for detection [7]. LSTM networks, a type of recurrent neural network (RNN), excel at capturing long-range dependencies and temporal information, making them well-suited for anomaly detection tasks, including machine fault detection and car-hailing demand forecasting [7,8]. CNN-LSTM hybrid architecture extends this capability by extracting local spatial features before feeding them into LSTM networks for temporal modeling.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

These models demonstrate strong performance in binding interrelated events over time and have been successfully applied to diverse tasks beyond malware detection, including electrical machine fault diagnosis and transportation demand prediction. Research has extensively explored hybrid CNN-LSTM architectures for time series analysis, with notable applications in machine fault detection and online car-hailing short-term demand forecasting [9]. In cybersecurity, such models have been adapted to learn from API call sequences by treating them as temporal data [2]. However, despite their strengths, these models face significant challenges from obfuscation techniques, redundant API calls, and novel zero-day attacks [10]. They often struggle to isolate critical API calls within noisy sequences and may lose fine-grained temporal order due to pooling operations. For ransomware detection, LSTMs may miss short, locally repeating patterns critical for accurate identification [11]. Recent works have explicitly incorporated time-aware modeling to account for the evolving nature of malware behaviors and temporal patterns in execution traces. For instance, AlSobeh et al. [12] proposed a time-aware machine learning framework for Android malware detection, demonstrating the importance of temporal features, such as application timestamps and time-correlated attributes, in improving classification robustness across evolving malware families spanning over a decade. Similarly, in the domain of network intrusion detection, Terawi et al. [13] introduced a time-aware deep learning approach that utilizes packet inter-arrival times alongside recurrent architectures, highlighting how explicit temporal modeling enhances the detection of denial-of-service attacks in cloud environments by capturing timing-based anomalies that traditional sequence models may overlook. These studies show the value of integrating explicit temporal signals beyond inherent sequential processing in models like LSTM, providing additional motivation for attention mechanisms which dynamically weigh critical subsequences to complement temporal dependency capture in API call sequence analysis.

Attention-augmented sequence models

Attention mechanisms dynamically assign weights to sequence elements, allowing models to prioritize features that contribute most to classification. This approach improves accuracy and interpretability by highlighting discriminative elements within complex data. In malware detection, attention-enhanced LSTMs and CNN-LSTM hybrids have demonstrated better robustness against unknown attacks by focusing on critical APIs and preserving low-frequency [10]. Similarly, attention mechanisms in broader anomaly detection contexts have been successfully integrated with sequence models to improve feature prioritization and reduce false negative rates [4].

Recent developments have further advanced the field through novel architectural innovations. Qian and Cong [10] proposed CAFTrans, a Transformer-based model incorporating a 1D Channel Attention Module and Term Frequency-Inverse Document Frequency reinforcement to enhance API feature specificity. This approach outperformed traditional baselines, including GaussianNB, Logistic Regression, K-Nearest Neighbor, Support Vector Machine, Decision Trees, Random Forest, MLP, Gradient Boosting, and other Transformer and LSTM-based methods. However, a significant gap persists in the current literature; many attention-based studies focus on single architecture types, such as LSTM or Transformer, rather than systematically comparing attention integration across multiple base models under identical conditions. To address this limitation, our work systematically evaluates the impact of attention mechanisms across multiple architectures, including LSTM, Gated Recurrent Unit (GRU), CNN, and Transformer, using the same dataset to isolate the contribution of attention from other architectural factors.

Attention in network intrusion detection systems

Attention modules in network intrusion detection systems assign varying importance to input features for each decision step, allowing models to capture the most relevant patterns in network traffic data. Zhao et al. [14] demonstrated a CNN-Bidirectional Long Short-Term Memory (Bi-LSTM) Network-Attention model achieving an accuracy of up to 95.67% on the CIC-IDS2017 and CIC-DDoS2019 datasets. Other works have combined Attention with LSTMs for IoT intrusion detection, Particle Swarm Optimization-Attention-LSTM for abnormal electricity consumption, and TabNet for IoT malware detection, all of which have shown improved accuracy and interpretability. Moreover, their approach integrated Attention with Bi-LSTM and hybrid CNN models for multi-domain attack detection, achieving higher performance than CNN, RNN, Parallel Cross Convolutional Network, or hybrid feedforward-CNN baselines.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

These developments demonstrate the growing sophistication of attention-based approaches in network security applications. Despite these promising results, a critical gap remains in the evaluation methodology; most works compare attention-enhanced models against non-attention baselines only, leaving uncertainty over whether performance gains stem from the attention component or other architectural differences. This limitation makes it difficult to isolate the actual contribution of attention mechanisms to model performance. To address this methodological gap, our approach includes both attention-augmented and non-attention versions of the same architectures, enabling a direct assessment of attention's impact on network intrusion detection performance across multiple model types.

Hybrid architectures and vision-based malware detection

Some studies transform malware samples into image representations, enabling the use of computer vision architectures. Vision Transformers (ViT) and hybrid CNN-attention models, such as CoAtNet, have been employed for these tasks. Jo et al. [15] utilized ViT for Android malware detection, outperforming CNN-based models such as ResNet and DenseNet. Similarly, Huang et al. [16] proposed CoAtNet, which achieves better accuracy than XceptionNet, EfficientNet, and ResNet50. These approaches have evolved to leverage self-attention mechanisms for capturing global context in malware images, enabling better generalization to unseen malware families. The integration of attention in vision-based malware detection represents a significant advancement in applying computer vision techniques to cybersecurity challenges. However, a notable gap exists in the current evaluation approach, as these studies often fail to make direct comparisons between attention-augmented CNNs and CNNs. Without such controlled comparisons, it remains unclear whether performance improvements are attributable to attention mechanisms or the underlying transformer-based design choices. To address this limitation, our work integrates attention modules into both vision-based and sequence-based architectures, testing them under identical conditions to quantitatively assess the specific contribution of attention to malware detection performance across different architectural paradigms.

Summary and identified research gap

Despite the proliferation of attention-augmented models, a consistent methodological shortcoming persists. Several studies propose a novel attention-based architecture and compare it with existing models [11,15]. While these comparisons demonstrate overall performance, they fail to isolate the specific contribution of the attention component itself. The performance gain could be attributed to the attention mechanism, other architectural changes, or a combination of both. Few works systematically compare an architecture with and without attention under controlled conditions.

Furthermore, a systematic comparison across multiple base architectures under identical conditions cannot evaluate this effect across a platform's multiple base architectures, including CNN, RNN, and Transformer. This paper aims to fill this gap by providing a controlled, ablation-style study that directly quantifies the value of attention for performance, interpretability, and efficiency in API-based malware detection. Table *1* provides a summary of the literature review.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Reference	Algorithm Used	Dataset	Accuracy (%)	F1-Score (%)
[7]	Hybrid CNN-LSTM attention-based model	Electric machine fault detection (vibration data)	NR	NR
[8]	LSTM-attention	Online car hailing dataset	NR	NR
[10]	Transformer-based model	Mal-API-2019 dataset (Identification)	0.646	0.653
[11]	ARI-LSTM	Mal-API-2019 dataset (Identification)	0.93	NR
[14]	Self-attention mechanism CNN	CIC IDS 2017 and CIC-DDoS2019	95.8	95.86
[15]	ViT Based-model	Androzoo Dataset	0.802	0.774
[16]	CoAtNet	Malicious code detection	96.6	96.57
[17]	TabNet with attention mechanism	CIC UNSW-NBIS	0.74	NR
Our Proposed Model	Attention-augmented LSTM (A-LSTM) with additive attention	Oliveira (2019) and Mal-API-2019	99.2	99.7

TABLE 1: Comparison of machine learning techniques used in malware detection research

NR, Not Reported; CNN, Convolutional Neural Network; LSTM, Long Short-Term Memory; ARI-LSTM, Attention-based Recurrent Identification–Long Short-Term Memory; ViT, Vision Transformer; CoAtNet, Convolution and Attention Network

Materials And Methods

We consider the problem of malware prediction in a deep learning model f_θ that classifies a given API call sequence S as either malware or benign. The task is to learn a mapping $f_\theta : S \rightarrow \{0,1\}$ where S is the set of all API sequences, and θ is the set of learnable parameters for a given model architecture. We aim to train a model that can accurately predict the class label $y \in \{0,1\}$ from the input sequence S, where $y = 1$ indicates malware and $y = 0$ indicates benign. Our overall approach is to systematically compare the performance and interpretability of several state-of-the-art deep learning architectures. We implement non-attention and attention-augmented versions of CNN, LSTM, and GRU models, with the Transformer serving as the pure attention-based baseline. This enables us to quantify the specific contribution of the attention mechanism to predictive power and model interpretability.

The different components of our method are detailed in the following subsections. The dataset used for this study is a publicly available dataset [18] that contains 42,797 malicious and 1,079 benign API call sequences. This collection demonstrated substantial class disparity, with malicious samples exceeding benign samples by more than 40%. This imbalanced distribution can negatively impact model effectiveness, especially during testing phases. To mitigate this class imbalance and prevent model bias towards the majority class, we applied the Synthetic Minority Over-sampling Technique (SMOTE) for class balancing, generating 42,797 instances for both categories. We selected SMOTE over alternative strategies for several reasons. First, undersampling would discard a vast amount of malicious samples, potentially harming the model's generalization. Second, while class weighting in the loss function is a common alternative, SMOTE actively creates a balanced training distribution, which we empirically found to lead to more robust

decision boundaries. Third, although more sophisticated, sequence-specific augmentations, such as random API substitutions or deletions, can preserve semantic integrity; however, they lack standardization and require careful, domain-specific design. SMOTE provided a reproducible and effective baseline for balancing our numerically encoded sequences.

It is important to note that SMOTE is fundamentally designed for continuous feature spaces in tabular data. In our application, raw API sequences were converted into fixed-length, padded numerical vectors. Each position in the padded sequence of length was treated as an independent feature dimension for SMOTE synthesis. While this approach successfully balances the class distribution and has been utilized in similar sequence classification contexts [2], we acknowledge that it may alter the precise sequential integrity and semantic meaning of the original API call sequences. However, our primary goal was to provide the models with a balanced view of both classes during training to prevent learning a trivial majority-class classifier. The model's subsequent strong performance on the independent, non-SMOTE-adjusted Mal-API-2019 test set validates that the overall sequential patterns critical for discrimination were preserved. Future work could explore sequence-specific data augmentation techniques to address this limitation.

For the independent evaluation of our model, we utilized the Mal-API-2019 dataset [19]. This dataset served as a distinct and independent test set to assess the generalization capability of the trained model. We chose this training and testing approach to rigorously assess the models' generalization capabilities to an independent, unseen dataset. The Oliveira dataset was used for training and validation due to its substantial size, which is critical for training complex deep learning models. The Mal-API-2019 dataset, designed as a comprehensive benchmark for Windows OS API call analysis, served as a distinct and independent test set to prevent data leakage and ensure that our results reflect real-world performance on a separate distribution of malware families. While the Mal-API-2019 dataset has a more balanced family distribution, both datasets were preprocessed identically, including the tokenization and numerical representation steps, to ensure a fair and consistent evaluation. Our pipeline consists of two main steps, tokenization and numerical representation. For a given input API sequence $S = \{s_1, s_2, \ldots, s_n\}$, we first tokenize each API call and map it to a unique integer ID. This results in a numerical sequence $X = \{x_1, x_2, \ldots, x_n\}$. We then standardize all sequences to a fixed length of $L_{\max}$ by padding or truncation, yielding the final input vector $\bar{X} \in \mathbb{N}^{L_{\max}}$.

Implemented architecture

While traditional machine learning methods, such as Random Forest, Support Vector Machines, and Logistic Regression, have been applied to malware detection, these approaches are limited in their ability to model sequential dependencies inherent in API call sequences. We therefore prioritize sequence-based deep learning models for rigorous evaluation. Nonetheless, we acknowledge the value of classical baselines and include them in our discussion for a broader context. In this research, we employed three distinct architectures as our non-attention baselines: 1D-CNN, an LSTM network, and a GRU network. Using convolutional filters, the 1D-CNN is designed to extract local, position-invariant features from the input sequence.

This approach efficiently identifies recurring patterns, regardless of their position within the sequence. The LSTM and GRU models, on the other hand, process the sequence sequentially. These architectures are well-suited for capturing long-range dependencies, and for this task, the final hidden state of the last time step is used to make the classification decision.

Attention-augmented models

To investigate the effect of attention, we developed and evaluated attention-augmented models by integrating an attention mechanism into each non-attention baseline. The core principle behind this augmentation is to compute a context vector that represents a weighted average of the feature representations. The weights are not fixed but are instead learned by the network to dynamically highlight the most salient parts of the input sequence for the classification task. Our CNN-based model passes the resulting feature maps to a self-attention layer after the convolutional and pooling layers. The attention weights for each feature vector are computed using a simple feed-forward network with a

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

softmax activation to ensure the weights sum to 1. This mechanism allows the model to selectively focus on the most relevant features learned by the convolutions. We augmented the LSTM and GRU architectures for the sequential models with a Bahdanau-style attention mechanism [20]. The high-level architecture is further shown in Figure *1*.

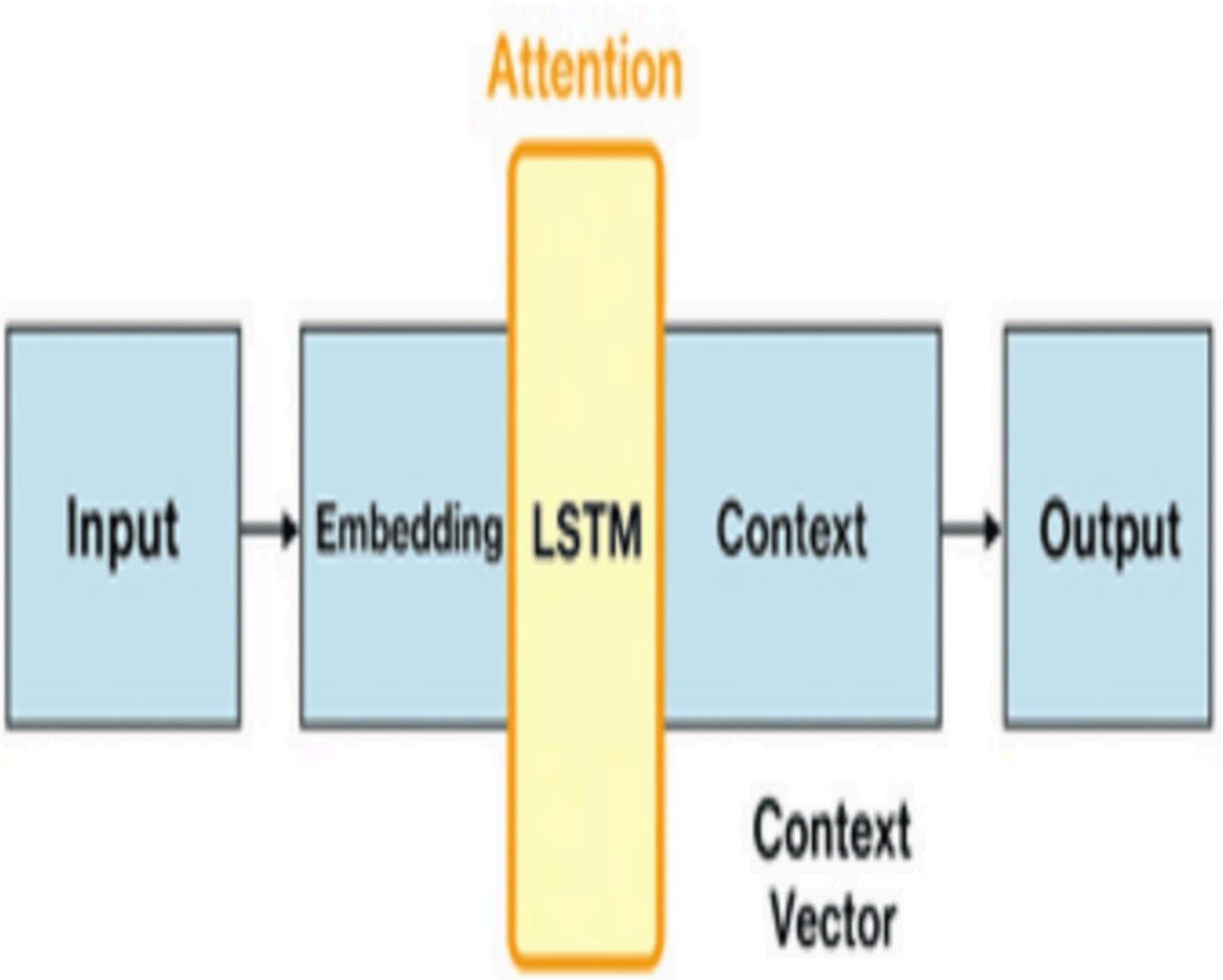

FIGURE 1: Architecture of an attention-augmented LSTM model for API sequence analysis

API, Application Programming Interface; LSTM, Long Short-Term Memory

As the networks process the input sequence, they produce a set of hidden states. The attention mechanism then computes a weighted context vector over these hidden states. The attention weights are learned based on the relevance of each hidden state to the final classification, allowing the model to revisit all hidden states and prioritize those that are

most relevant. This context vector is concatenated with the final hidden state and passed to the final classification layer. This approach is well-established in the literature for improving the performance of recurrent models on various sequence-to-sequence tasks.

Transformer benchmark

We utilized the Transformer model initially proposed by Vaswani et al. [21] for our attention-based benchmark. This model's architecture is unique because it abandons traditional recurrence and convolution, relying instead on a powerful self-attention mechanism to process input sequences. This mechanism is central to the model's ability to weigh the significance of different words within a sequence, regardless of their distance. The Multi-Head Self-Attention (MHSA) module is the primary component of the Transformer model. It extends the concept of self-attention by allowing the model to jointly attend to information from various "representation subspaces" simultaneously. It splits the attention process into multiple heads that work in parallel, thereby enhancing the model's ability to focus on diverse aspects of input. The foundation of the MHSA module is the Scaled Dot-Product Attention mechanism. For a given set of input vectors, three matrices, Query (Q), Key (K), and Value (V), are generated by multiplying the input embeddings with learned weight matrices. This compatibility is a key factor in how the model determines which parts of the input sequence are most relevant to each other:

$$\mathbf{Attention}(Q, K, V) = \mathbf{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \tag{1}$$

In Equation 1, *Q* denotes the Query matrix, *K* denotes the Key matrix, and *V* denotes the Value matrix, each derived by multiplying the input embeddings by their respective learned linear projection matrices. The term QK^T is the dot product between queries and keys, producing a matrix of raw compatibility scores. The scaling factor $\sqrt{d_k}$, where d_k is the dimension of the key vectors, prevents these scores from growing too large in magnitude; without it, large dot products push the softmax function into regions of near-zero gradient, impeding learning. The softmax function normalises the scaled scores into a probability distribution (attention weights), which are used to compute a weighted sum of the Value vectors *V*. The result is a context vector that represents the attended information for each query position. This scaling factor is critical because it helps prevent large dot product values, which can cause the softmax function to become insensitive to gradients. This would hinder the model's ability to learn effectively during training. To enhance the model's capacity and allow it to attend to different aspects of the input simultaneously, the Transformer uses MHSA. Instead of using a single attention function. In Equation 2, $\text{head}_1, \ldots, \text{head}_h$ are *h* independent attention outputs, each computed by the Scaled Dot-Product Attention mechanism in Equation 1 on differently projected subspaces of *Q*, *K*, and *V*. The Concat(·) operation concatenates all h head outputs along the feature dimension, producing a vector of dimension $\boldsymbol{h \times dv}$. The matrix $\boldsymbol{W^N \in \mathbb{R}hdv \times dmodel}$ is a learned output projection that linearly transforms the concatenated heads back to the model's working dimension $\boldsymbol{d_{\text{model}}}$. This architecture allows the model to jointly attend to information from h different representation subspaces simultaneously, capturing richer dependency patterns than a single attention function.

The formula for Multi-Head Attention is as follows:

$$\mathbf{MultiHead}(Q, K, V) = \mathbf{Concat}(\mathbf{head}_1, \ldots, \mathbf{head}_h)\, W^O$$

Equation 3 defines how each individual attention head is computed. For the i-th head, the global Query matrix Q, Key matrix K, and Value matrix V are each linearly projected using their own learned weight matrices W^{Qi}, W^{Ki}, and W^{Vi}, respectively. This projection maps the shared input representations into a distinct, lower-dimensional subspace specific to head *i*. The Scaled Dot-Product Attention function from Equation 1 is then applied to these projected matrices, producing a head-specific context vector. Because each head operates on a different linear projection, the *h* heads collectively attend to different aspects of the input sequence in parallel.

$$\mathbf{head}_i = \mathbf{Attention}\left(QW_i^Q,\, KW_i^K,\, VW_i^V\right)$$

Equation 4 specifies the dimensions of all learned projection matrices used in the Multi-Head Attention mechanism.

$$W_i^d \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

maps the full model dimension d_{model} to the per-head query dimension d_k for head i. $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ performs the same mapping for keys. $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ maps to the per-head value dimension d_v. The output projection $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ maps the concatenated output of all h heads (of total dimension hd_v) back to the model's working dimension d_{model}. In this study, $d_{\text{model}} = 128, \quad h = 4, \quad d_k = d_v = 32$.

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k},\ W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k},\ W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v},\ \text{and}\ W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

The input sequence of tokens is first converted into dense vectors using an embedding layer. Since the model contains no recurrence, positional encodings are added to these embeddings to give the model information about the relative or absolute position of the tokens in the sequence. These position-aware embeddings are then fed into the stack of Transformer encoder layers. Table *2* shows the hyperparameter settings and architectural configuration for all implemented models.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Parameter	CNN/A-CNN	LSTM/A-LSTM	GRU/A-GRU	Transformer
Embedding dimension	128	128	128	128
Convolutional filters	2 layers (128, 64)	NR	NR	NR
Kernel size	3	NR	0.996	0.993
Pooling	Global max pooling	NR	NR	NR
Recurrent units	NR	2 layers, 128 units	2 layers, 128 units	NR
Transformer heads	NR	NR	NR	2 layers, 4 heads
Feed-forward dimension	NR	NR	NR	128
Optimizer	Adam	Adam	Adam	Adam
Learning rate	0.001	0.001	0.001	0.001
Batch size	64	64	64	64
Epochs	50	50	50	50
Dense units	64	64	64	64
Activation (hidden)	ReLU	ReLU	ReLU	ReLU
Activation (output)	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Dropout rate	0.2	0.2 (input & recurrent)	0.2 (input & recurrent)	0.1
Attention (when used)	Self-attention (single-head)	Bahdanau additive	Bahdanau additive	Multi-head scaled dot-product

TABLE 2: Hyperparameter settings and architectural configurations for all implemented models

NR, Not Reported; A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit; ReLU, Rectified Linear Unit

The architectural hyperparameters, including embedding size, sequence length, layer configurations, and dropout rate, for the models CNN, LSTM, GRU, Transformer, and their attention-augmented variants are summarized in Table *2*. All models were trained for a maximum of 50 epochs using the Adam optimizer with a learning rate of 0.001 and a batch size of 64. Early stopping was applied with a patience of 10 epochs, monitoring the validation loss to prevent overfitting. The binary cross-entropy loss was minimized. Experimental consistency was maintained across both the baseline and

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

attention-augmented models to facilitate a direct comparison. Notably, the transition to attention-based variants resulted in a minimal parameter overhead, ensuring that performance gains were not merely a product of increased model capacity.

Training and evaluation

All models are trained on the same training data and evaluated on the test set and dataset 2. We use the Adam optimizer to minimize the binary cross-entropy loss function, which is defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

In Equation 5, *N* is the number of samples in the training mini-batch, set to 64 across all experiments. For each sample *i*, $y_i \in \{0,1\}$ is the ground-truth binary label: $y_i = 1$ if the sample is malware, and y_i = 0 if benign. The term $\hat{y}_i \in (0,1)$ is the predicted probability that sample *i* belongs to the malware class, output by the sigmoid activation of the final dense layer. The natural logarithm log(·) imposes a large penalty when the model assigns high confidence to the wrong class: when $y_i = 1$ but $\hat{y}_i \approx 0$, the term $y_i \cdot \log(\hat{y}_i)$ approaches negative infinity. Conversely when $y_i = 0$ but $\hat{y}_i \approx 1$, the term $(1-y_i)\cdot\log(1-\hat{y}_i)$ approaches negative infinity. The factor $-\frac{1}{N}$ produces a positive, batch-normalised scalar that the Adam optimiser minimises through backpropagation. We evaluate performance using Accuracy, Precision, Recall, F1-score, and AUC metrics. We also measure the models' efficiency by tracking training time, testing time, and memory usage.

Attention-specific evaluation

Beyond standard performance metrics, we assess the interpretability of the attention mechanism by evaluating its faithfulness the extent to which attention weights reflect the model's underlying reasoning and diagnosticity, which measures the efficacy of these explanations in distinguishing true signal from noise. We assess faithfulness by measuring the drop in prediction confidence when the most highly attended tokens are removed from an input sequence, thereby evaluating the positive and negative gains. Positive performance gain occurs when the model removes the tokens with the highest attention scores, and its performance on a specific metric (such as accuracy or F1) improves, indicating that those tokens were significant for the model's prediction. The larger the drop, the more accurately the attention mechanism focused on the input's crucial parts. A negative performance gain or drop occurs when the highly attended tokens are removed, and the model's performance either remains the same or improves. Attention did not correctly identify the most important tokens. This suggests a lack of faithfulness.

Additionally, diagnosticity is a crucial metric that evaluates the faithfulness and robustness of an explanation method, such as attention mechanisms. It goes beyond simply showing that an explanation is plausible and checks if it is truly diagnostic of the model's internal reasoning. We evaluate diagnosticity by conducting a counterfactual test. This involves creating new input sequences that are semantically similar to the original but cause a change in the model's prediction. A diagnosticity attention mechanism should show a significant shift in attention distribution to the tokens responsible for the new prediction. For example, if we have a positive review and minimally edit it to negative, a diagnostic attention mechanism would shift its focus from the original positive to the new negative words. The test's core assumption is that if an explanation method is diagnostic, it must behave consistently. This means that:

· Inputs with similar predictions should have similar attention distributions.

· Inputs with different predictions should have significantly different attention distributions, highlighting the specific tokens that led to the change.

Therefore, by measuring the divergence between the attention distributions of these counterfactual pairs, we can confirm that our attention-based explanations are truly diagnostic of the model's behavior and not merely superficial post hoc artifacts. This provides a much deeper and more reliable understanding of why the model makes a specific prediction. Figure *2* further illustrates the model architecture overview.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

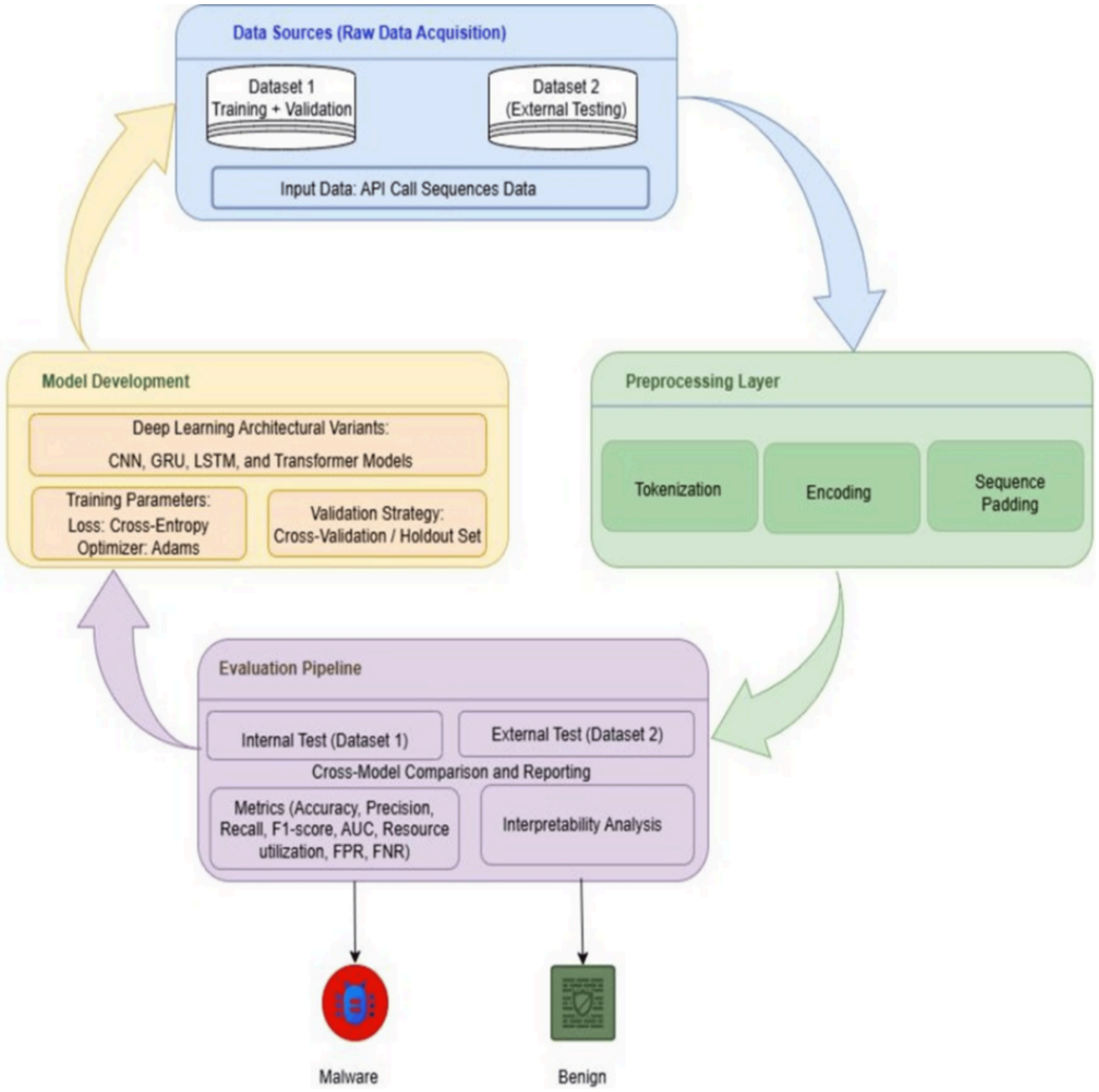

FIGURE 2: Architecture diagram of the proposed model

API, Application Programming Interface; LSTM, Long Short-Term Memory; CNN, Convolutional Neural Network; GRU, Gated Recurrent Unit; AUC, Area Under The Curve; FPR, False Positive Rate; FNR, False Negative Rate

Figure *2* illustrates the deep learning workflow for building and evaluating network traffic classifiers. The process begins with Raw Data Acquisition, where API call sequences and network flow data are collected. Next, the data undergoes preprocessing, which includes tokenization, encoding, and sequence padding. The preprocessed data is used in the training and validation phase, where models are trained using a cross-entropy loss function and the Adam optimizer. The models include architectural variants such as CNN, GRU, LSTM, and Transformer. Finally, the trained models are evaluated

in the Evaluation Pipeline, where their predictive performance and interpretability are analyzed. The results from all models are then used for Cross-Model Comparison and Reporting, where they are aggregated and analyzed. We have provided Algorithm 1 (Table *3*) for the proposed attention model to clarify the implementation process.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Algorithm 1. Systematic Comparative Attention Model Pseudocode	
Input:	Dataset $D=\{(S_i y_i)\}_{i=1}^{N}$ of API sequences and labels.
Output:	A set of performance and resource metrics for each model, including visualizations and interpretability.
1.	Preprocess Dataset: Tokenize, pad/truncate sequences to L_{max} and split into training set D_{train} and test set D_{test}.
2.	For each model architecture {CNN, LSTM, GRU, Transformer} do:
3.	For each model variant $V \in$ non-attention, attention do:
4.	Initialize Model: $f_\theta \leftarrow$ Model(M,V).
5.	Train Model: Fit f_θ on to minimize binary cross-entropy loss.
6.	Evaluate Performance: Compute standard metrics on D_{test}.
7.	Measure Resources: Record training time, inference time, and memory usage.
8.	If V is Attention:
9.	Evaluate interpretability: Compute faithfulness and diagnosticity scores.
10.	Visualize: Generate attention heatmaps.
11.	Compare Results: Plot and analyze all metrics to identify the most effective and efficient models.
12.	end for
13.	end for
14.	end

TABLE 3: Systematic comparative attention model pseudocode

API, Application Programming Interface; LSTM, Long Short-Term Memory; CNN, Convolutional Neural Network; GRU, Gated Recurrent Unit

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Experimental setup

The computational experiments for evaluating our attention-based malware detection models were conducted on a dedicated research workstation with the following specifications: Windows 11 Enterprise operating system, Intel Core i7-10700 CPU (2.90 GHz), 16 GB RAM, and NVIDIA Quadro P620 GPU. Our implementation leveraged Python 3.9.1 as the primary programming language, utilizing the deep learning frameworks TensorFlow 2.3.0 and Keras 2.7.0 for model architecture, alongside Scikit-learn 1.0.2 for data preprocessing and calculation of performance metrics. Table *4* provides a detailed summary of the dataset used.

Characteristic	Oliveira Dataset (2019) [18]	Mal-API-2019 Dataset [19]
Primary Use	Training & Validation	Independent Testing
Total Samples	43,876	7,107
Malware Samples	42,797	7,107
Benign Samples	1,079	Malware Only
Malware Families	Not explicitly labeled	8 (Trojan, Backdoor, Downloader, worms, Virus, Dropper, Spyware, Adware)
After SMOTE	Benign (42,797) and Malware (42,797)	

TABLE 4: Training and testing dataset details

SMOTE, Synthetic Minority Over-sampling Technique

Dataset details

As shown in Table *4*, our study utilizes two distinct datasets for model training and evaluation: the Oliveira dataset [18] for training and the Mal-API-2019 dataset [19] for testing. The Oliveira dataset provides the volume and variety needed to train deep learning models on fundamental malicious API call patterns. While its samples predate 2020, the core malicious behaviors, such as code injection and persistence, remain relevant, and its size helps prevent overfitting. The Mal-API-2019 dataset serves as our independent test set precisely because it presents a different and challenging distribution; it consists of metamorphic malware from 2019, designed to evade signature-based detection. This makes it an excellent benchmark for evaluating a model's ability to generalize and capture invariant malicious patterns despite obfuscation, a critical capability for detecting novel threats. We acknowledge that continuous evaluation of emerging malware families is necessary for operational systems; however, for the controlled and comparative goal of this study, which isolates the effect of attention mechanisms, these datasets provide a rigorous and appropriate foundation.

To evaluate binary classification performance on this malware-only benchmark, 1079 benign samples from the Oliveira dataset were sequestered as an independent held-out test set. These samples were strictly excluded from all training and validation phases to prevent data leakage. Consequently, the final evaluation set for Dataset 2 consisted of 7107 malicious sequences from Mal-API-2019 and the 1079 benign sequences reserved from the Oliveira dataset.

To address this disparity and prevent model bias towards the majority class, we applied SMOTE. As noted in Table *3*, SMOTE was applied after preprocessing, treating each position in the padded numerical sequence as a feature dimension. While this technique effectively balances the class distribution, it operates on the numerical representation and does not guarantee the generation of semantically valid API sequences. Consequently, the synthesized samples may not represent realistic program execution paths. Despite this limitation, which is common when applying SMOTE to sequential data, the technique proved effective in our experiments for mitigating class imbalance, as evidenced by the models' strong generalization to the independent, non-augmented Mal-API-2019 test set. The benefits of providing a balanced training set outweighed the potential noise introduced by this method.

The independent test set was the Mal-API-2019 dataset [19]. It is a comprehensive benchmark specifically designed for analyzing Windows OS API calls in malware classification research. This dataset is particularly valuable for behavioral analysis as it provides sequential API call data from metamorphic malware samples, addressing a key limitation in many existing benchmarks. The dataset contains 7,107 malware samples distributed across eight families. The family distribution is relatively balanced, with 1001 samples each for Trojan, Backdoor, Downloader, Worms, and Virus families. The dataset also includes 891 Dropper samples, 832 Spyware samples, and 379 Adware samples.

Baselines

To comprehensively evaluate the efficacy of our proposed methods, we establish a robust set of baselines. These fall into two primary categories:

· Non-Attention Models: We implemented 1D-CNN, a LSTM network, and a GRU network to serve as direct comparisons for their attention-augmented counterparts.

· Attention-Only Model: The Transformer model, an architecture built entirely on multi-head self-attention, serves as a robust baseline. This comparison is crucial for evaluating how our attention-augmented RNN and CNN models compare to state-of-the-art pure attention-based architectures.

Evaluation metrics

We employ a combination of standard metrics for a thorough evaluation. We report the Accuracy, Precision, Recall, and F1-score on the test set. These metrics offer a more granular view of classification performance, which is crucial for tasks with potential class imbalance. The Area Under the Receiver Operating Characteristic Curve (AUROC) is also used to evaluate the models' overall discriminative power. To ensure reproducibility and account for training stochasticity, all experiments, including the ablation study, were conducted over five independent runs with different random seeds. Results are reported as mean ± standard deviation.

Equation 6 defines Accuracy as the proportion of all correct predictions out of the total number of samples evaluated. True Positives (TP) is the count of malware samples correctly identified as malware. True Negatives (TN) is the count of benign samples correctly identified as benign. False Positives (FP) is the count of benign samples incorrectly classified as malware. False Negatives (FN) is the count of malware samples incorrectly classified as benign. The denominator TN + FP + TP + FN equals the total number of samples N. While Accuracy provides an intuitive overall performance measure, it can be misleading in class-imbalanced settings - a model that predicts the majority class for all samples achieves high Accuracy while failing to detect any malware. This limitation is addressed in this study through supplementary metrics reported in Equations 7-9, and through SMOTE-based class balancing during training. Accuracy is the ratio of all correct predictions (TP+TN) to the total number of predictions or entries in the sample (TP + TN + FN + FP). Equations (1)-(4) represent how a model's Accuracy, Precision, Recall, and F1-score are calculated.

$$\text{Accuracy} = \frac{TN+TP}{TN+FP+TP+FN}$$

Equation 6 defines Accuracy as the proportion of all correct predictions out of the total number of samples evaluated. TP is the count of malware samples correctly identified as malware. TN is the count of benign samples correctly identified as benign. FP is the count of benign samples incorrectly classified as malware. FN is the count of malware samples incorrectly classified as benign. The denominator TN + FP + TP + FN equals the total number of samples N. While

Accuracy provides an intuitive overall performance measure, it can be misleading in class-imbalanced settings - a model that predicts the majority class for all samples achieves high Accuracy while failing to detect any malware. This limitation is addressed in this study through supplementary metrics reported in Equations 7-9, and through SMOTE-based class balancing during training.

$$\text{Precision} = \frac{TP}{TP+FP}$$

Equation 8 defines Recall - also known as sensitivity or the True Positive Rate as the proportion of actual malware samples that the model successfully detects. TP is the count of malware samples correctly identified. FN is the count of malware samples missed by the model. The denominator TP + FN equals the total number of actual malware samples in the evaluation set. In cybersecurity applications, Recall is a critical metric: each missed malware detection (FN) represents an active threat that remains undetected on the system, potentially leading to compromise. Note that Recall = 1 − FNR, establishing a direct relationship with the False Negative Rate (FNR), which is the primary evaluation metric in this study, as reported in Table *8*.

$$\text{Recall} = \frac{TP}{TP+FN}$$

Equation 9 defines the F1-Score as the harmonic mean of Precision (Equation 7) and Recall (Equation 8), combining both into a single balanced performance measure. The numerator 2 × Precision × Recall scales the product of the two metrics, while the denominator Precision + Recall normalises this by their sum. The harmonic mean is preferred over the arithmetic mean because it penalises extreme imbalances between Precision and Recall: a model that achieves high Precision but near-zero Recall or vice versa - will receive a near-zero F1-score, accurately reflecting its practical failure. The F1-Score is the primary ranking metric in this study, reported across all seven model variants and both datasets in Table *8*.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Furthermore, we assess the practical efficiency of each model by measuring training time, testing time per sample, and memory usage during training. These metrics are crucial for determining the feasibility of real-world deployment on resource-constrained platforms. For our attention-augmented models, we evaluate the quality of the attention mechanism using faithfulness and diagnosticity to ensure the attention weights are a reliable indicator of the model's decision-making process.

Results And Discussion

The empirical findings from our comparative analysis of CNN, LSTM, GRU, and Transformer models both with and without integrated attention mechanisms are presented in this section. We evaluate predictive performance using standard benchmarks, including Accuracy, Precision, Recall, F1-score, and AUC-ROC, as well as practical deployment viability assessed through measures of training duration, testing time, and memory consumption. We begin by describing the experimental environment in which all models were trained and evaluated.

Experimental setup

To ensure reproducibility and fair comparison across all model architectures, all computational experiments were conducted under a controlled and consistent hardware and software environment. This section provides a comprehensive description of the experimental configuration, including the computing infrastructure, software stack, training protocol, and reproducibility measures adopted throughout the study.

Computing infrastructure

All experiments were executed on a dedicated research workstation configured as follows. The system ran Windows 11 Enterprise as the operating system, with an Intel Core i7-10700 CPU clocked at 2.90 GHz, providing the primary computational backbone. The workstation was equipped with 16 GB of RAM to accommodate the memory demands of

training deep learning models on large API call sequence datasets. GPU-accelerated training was enabled via an NVIDIA Quadro P620 GPU, which facilitated the parallel processing required for matrix operations in the attention mechanism layers and transformer encoders. Table *5* summarises the hardware specifications.

Component	Specification
Operating System	Windows 11 Enterprise
Processor (CPU)	Intel Core i7-10700, 2.90 GHz (8 cores/16 threads)
Memory (RAM)	16 GB DDR4
GPU	NVIDIA Quadro P620 (4 GB GDDR5)
Storage	Local SSD (for dataset and checkpoint I/O)

TABLE 5: Hardware configuration of the experimental workstation

Software environment and libraries

The implementation was carried out in Python 3.9.1. The deep learning models were built and trained using TensorFlow 2.3.0 with the Keras 2.7.0 high-level API for model definition, compilation, and training. Data preprocessing routines, including tokenisation, sequence padding, label encoding, and the application of SMOTE, were implemented using Scikit-learn 1.0.2. Performance metrics (Accuracy, Precision, Recall, F1-score, AUC-ROC, FPR, and FNR) were computed using Scikit-learn's built-in evaluation utilities. Attention weight extraction and heatmap visualisation were performed using NumPy 1.21 and Matplotlib 3.4. Table *6* summarises the complete software stack.

Component	Specification
Programming Language	Python 3.9.1
Deep Learning Framework	TensorFlow 2.3.0/Keras 2.7.0
Machine Learning Preprocessing & Metrics	Scikit-learn 1.0.2
Numerical Computing	NumPy 1.21
Visualisation	Matplotlib 3.4
Class Balancing	SMOTE (via imbalanced-learn)

TABLE 6: Software stack and library versions used across all experiments

Training protocol and hyperparameter configuration

A unified training protocol was applied consistently across all model variants, both attention-augmented and non-attention baselines, to ensure that observed performance differences are attributable solely to architectural choices rather than training disparities. All models were trained for a maximum of 50 epochs using the Adam optimiser with a fixed learning rate of 0.001 and a mini-batch size of 64. The binary cross-entropy loss function was minimised throughout training. To prevent overfitting, early stopping was applied with a patience of 10 epochs, monitoring the validation loss. Model weights at the epoch of lowest validation loss were retained via checkpoint saving.

The dataset was partitioned into training, validation, and test sets. The Oliveira dataset [18] (after SMOTE balancing: 42,797 malware and 42,797 benign samples) was used for training and validation, while the independent Mal-API-2019 dataset [19], augmented with 1,079 held-out benign samples from the Oliveira dataset, served exclusively as the test set. No data from the test set was used at any stage of training or hyperparameter tuning, ensuring no data leakage. Table *7* summarises the unified hyperparameter configuration.

Hyperparameter	CNN/A-CNN	LSTM/A-LSTM	GRU/A-GRU	Transformer
Embedding Dimension	128	128	128	128
Optimiser	Adam	Adam	Adam	Adam
Learning Rate	0.001	0.001	0.001	0.001
Batch Size	64	64	64	64
Max Epochs	50	50	50	50
Early Stopping Patience	10	10	10	10
Dropout Rate	0.2	0.2	0.2	0.1
Loss Function	Binary Cross-Entropy	Binary Cross-Entropy	Binary Cross-Entropy	Binary Cross-Entropy
Attention Type	Self-attention (single-head)	Bahdanau additive	Bahdanau additive	Multi-head scaled dot-product
Attention Heads	-	-	-	4

TABLE 7: Unified hyperparameter configuration across all model variants

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Reproducibility and statistical reporting

To account for the stochastic nature of deep learning training arising from random weight initialisation, dropout regularisation, and mini-batch sampling, all experiments were independently repeated across five runs with different random seeds. Performance metrics are reported as mean ± standard deviation across these five runs. This protocol enables a statistically meaningful comparison between model variants and minimises the risk of reporting results that arise from a particularly favourable or unfavourable random initialisation. The ablation study results also follow this five-run reporting convention to ensure consistency.

Attention-specific interpretability evaluations, including faithfulness (token masking) and diagnosticity (counterfactual testing), were conducted on a randomly selected subset of 500 malicious samples drawn from the Mal-API-2019 test set. The Kullback-Leibler (KL) divergence between attention distributions was computed across counterfactual pairs to quantify attention shift. All random subsets were sampled using fixed seeds for reproducibility.

With the experimental environment fully established, the following subsections present the comparative results of all model variants evaluated under this unified protocol, beginning with the main performance metrics

Main results

The results of our experiments, summarized in Table *8*, demonstrate that the attention-augmented models consistently outperform their non-attention counterparts across all key performance metrics. As shown in Figure *3*, which presents the F1-score comparison across all seven model variants on Dataset 1, the bars corresponding to the attention-augmented architectures (A-CNN, A-GRU, and A-LSTM) are consistently taller than those of their non-attention counterparts, confirming that the performance gain is architecture-agnostic. Among all models, the A-LSTM achieved the highest F1-score of 0.997 and a ROC-AUC of 0.976 on Dataset 1. Similarly, Figure *4* shows the F1-score distribution on the independent Mal-API-2019 test set (Dataset 2), where attention-augmented models again outperform their baselines, with the A-GRU achieving the highest F1-score of 0.989, closely followed by the A-LSTM and Transformer, both at 0.988.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Dataset 1							
Model	Accuracy	Precision	Recall	F1-Score (Mean SD Error)	AUC	FPR	FNR, (Mean SD Error)
CNN	0.994	0.995	0.998	0.994 (0.002)	0.991	0.003	0.004 (0.0006)
GRU	0.989	0.991	0.998	0.992 (0.002)	0.986	0.012	0.005 (0.0005)
LSTM	0.986	0.99	0.996	0.993 (0.002)	0.971	0.022	0.006 (0.0003)
A-CNN	0.992	0.994	0.998	0.996 (0.001)	0.971	0.004	0.002 (0.0007)
A-GRU	0.991	0.992	0.999	0.996 (0.001)	0.983	0.001	0.001 (0.0002)
A-LSTM	0.992	0.993	0.999	0.997 (0.001)	0.976	0.001	0.001 (0.0002)
Transformer	0.985	0.988	0.997	0.992 (0.002)	0.946	0.023	0.010 (0.0004)

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Dataset 2							
Model	Accuracy	Precision	Recall	F1-score, (mean SD error)	AUC	FPR	FNR, (mean SD error)
CNN	0.981	0.978	0.97	0.979 (0.004)	0.976	0.007	0.008 (0.0009)
GRU	0.976	0.935	0.961	0.985 (0.003)	0.971	0.01	0.012 (0.0011)
LSTM	0.972	0.94	0.962	0.950 (0.006)	0.962	0.015	0.004 (0.0008)
A-CNN	0.979	0.981	0.99	0.985 (0.003)	0.967	0.008	0.006 (0.0007)
A-GRU	0.983	0.985	0.992	0.989 (0.002)	0.974	0.005	0.006 (0.0006)
A-LSTM	0.982	0.984	0.991	0.988 (0.002)	0.969	0.006	0.001 (0.0003)
Transformer	0.982	0.984	0.991	0.988 (0.002)	0.969	0.006	0.004 (0.0006)

TABLE 8: Comparison of models performance across datasets (mean ± SD over 5 runs)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit, SD, Standard Deviation; AUC, Area Under The Curve; FPR, False Positive Rate; FNR, False Negative Rate

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

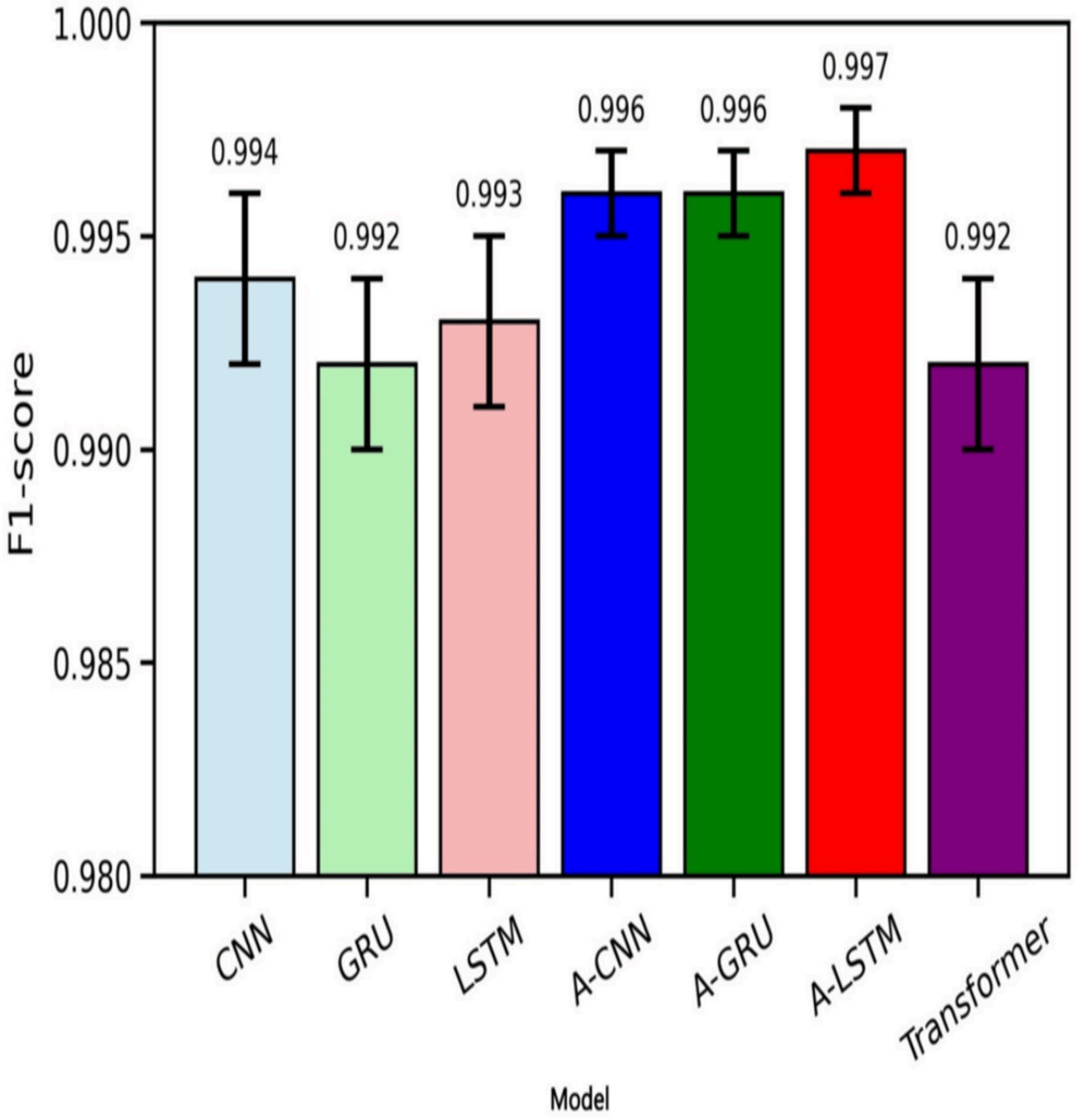

FIGURE 3: F1-score comparison across models (Dataset 1)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

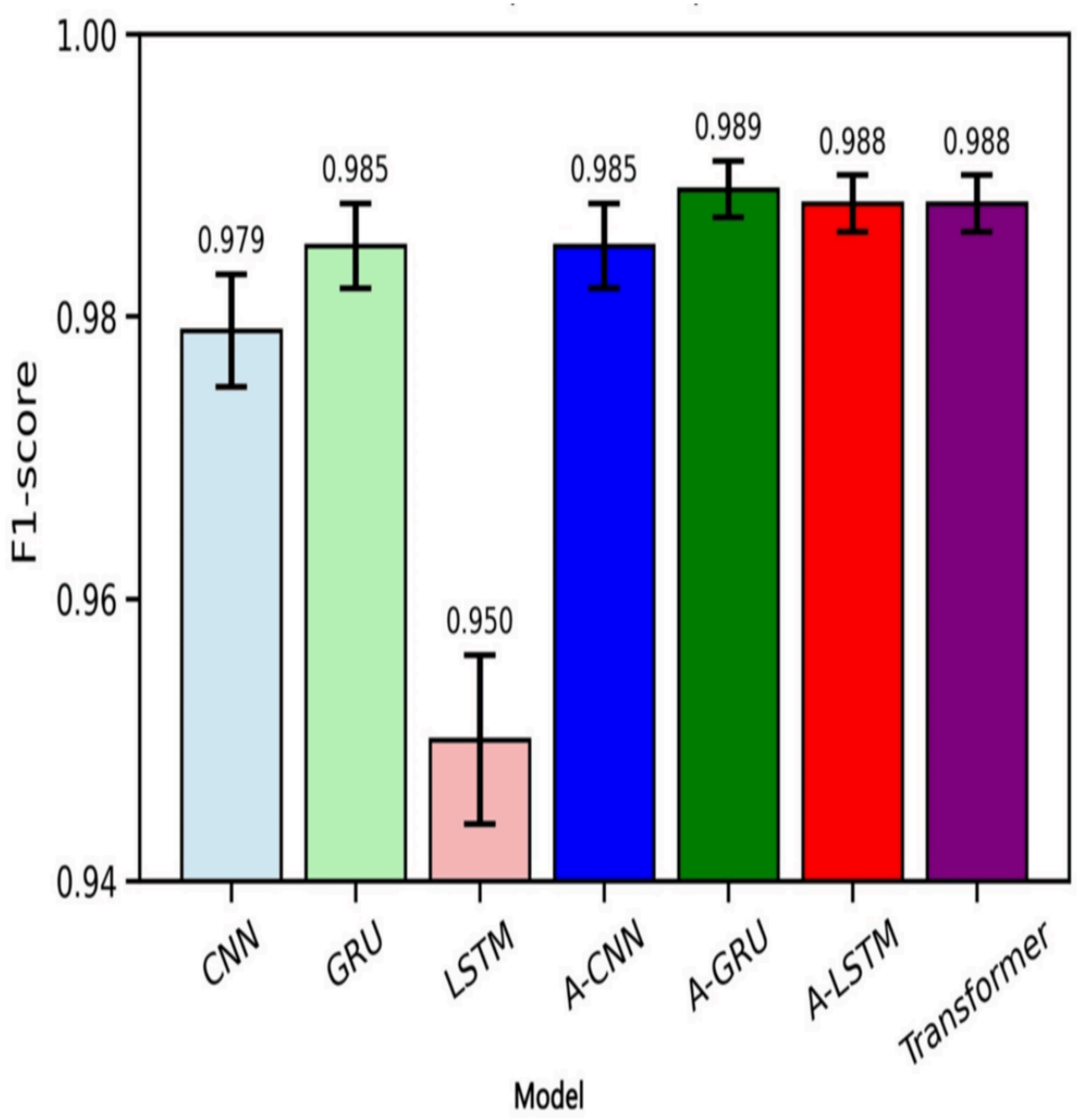

FIGURE 4: F1-score comparison across models (Dataset 2)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

The improvements in both CNN and RNN models were significant after adding an attention layer, confirming that the attention mechanism is a valuable addition to these architectures for this specific task. As visible in Figures *3* and 4, the attention-augmented A-CNN achieves an F1-score of 0.996 on Dataset 1 compared to 0.994 for the standard CNN, while the A-GRU reaches 0.996 against 0.992 for the baseline GRU. The A-LSTM model consistently achieved the highest F1-score on both datasets, rising from 0.993 (LSTM) to 0.997 (A-LSTM) on Dataset 1, an improvement reflected by the tallest

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

bar in Figure *3*. This is likely due to the more complex gating mechanism of the LSTM, which provides a more robust ability to model the sequential patterns inherent in API call data, further amplified by the attention layer's ability to selectively focus on the most discriminative calls.

The better performance of the A-LSTM over the Transformer on Dataset 1 can be understood through the lens of inductive bias alignment, and is visually apparent in Figure *3*, where the Transformer bar is the shortest among all models. Malicious API patterns, such as CreateProcess and WriteProcessMemory, are typically short and localized, and do not require integrating information from hundreds of distant calls for accurate classification. The LSTM, with its inherent bias for sequential local context followed by an attention filter, is structurally predisposed to this task. The standard Transformer's global self-attention, while powerful for modeling long-range dependencies, inherently dilutes sharp local signals by distributing attention across the entire sequence, introducing noise where the decisive evidence is concentrated. Future adaptations, such as local-window attention or convolution-augmented transformers, could reconcile the Transformer's parallel efficiency with the need for localized focus. Notably, however, Figure *4* reveals that this gap narrows considerably on the independent Mal-API-2019 test set: the Transformer achieves an F1-score of 0.988 on Dataset 2, matching the A-LSTM exactly and trailing only the A-GRU (0.989). This convergence suggests that the Mal-API-2019 dataset may contain malware families whose discriminative features are more globally distributed, making it more amenable to the Transformer's inductive bias.

This suggests that the Mal-API-2019 dataset may contain malware families whose discriminative features are more globally distributed or that its sequences are structured in a way that is more amenable to the Transformer’s inductive bias. All models, including the Transformer, underwent the same rigorous hyperparameter optimization to ensure a fair comparison. The divergent performance across datasets highlights that an optimal architecture can be data-dependent; however, the consistent gains from augmenting LSTMs and GRUs with attention demonstrate its broad utility.

Confusion matrix analysis on the independent test set

To further examine the classification behavior beyond aggregate metrics, we generated confusion matrices for all attention-based architectures evaluated on Dataset 2: A-CNN, A-GRU, A-LSTM, and the Transformer. Dataset 2 was used exclusively for testing and consists of 7,107 malicious API call sequences from the Mal-API-2019 dataset and 1,079 benign sequences held out from the Oliveira dataset, thereby reflecting a realistic and imbalanced evaluation scenario. Figure *5* presents the confusion matrices for the attention-augmented models using absolute classification counts.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

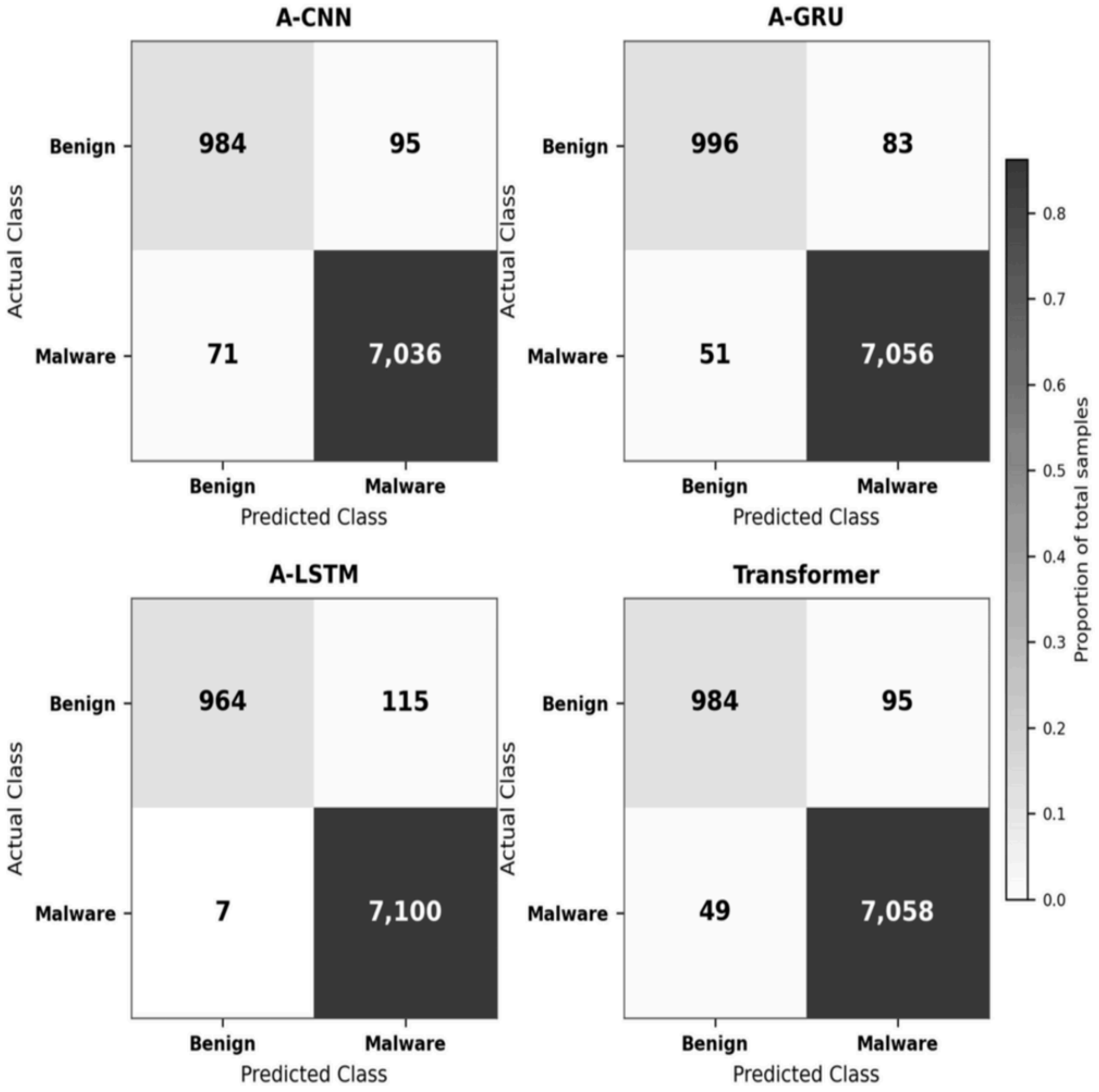

FIGURE 5: Confusion matrices of attention-based models evaluated on dataset 2

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

Across all attention-based architectures, the number of FN remains low, indicating strong generalization to unseen malware families. Among the evaluated models, the A-LSTM demonstrates the most favorable error profile, correctly identifying 7,100 out of 7,107 malicious samples, resulting in only seven false negatives. This corresponds to the lowest FNR observed on the independent test set.

While the A-LSTM exhibits a slightly higher number of FP compared to some architectures, this trade-off is acceptable in security-critical applications where minimizing missed detections is a primary objective. The Transformer and A-GRU models also show competitive performance; however, their confusion matrices indicate higher FNR, suggesting reduced

sensitivity to specific malicious execution patterns. Overall, the confusion matrix analysis confirms that integrating attention mechanisms improves detection robustness, with the A-LSTM providing the most reliable balance between sensitivity and specificity on Dataset 2. Table *9* and Figures *6*-*7* compare the resources utilized by the models in the experiment.

Dataset 1			
Models	Training Time (s)	Testing Time (s)	Memory Usage (MB)
CNN	57	7.1	75
GRU	69	8.4	90
LSTM	80	12.2	120.3
A-CNN	60	7.5	78
A-GRU	72	9.0	95
A-LSTM	82	12.5	125
Transformer	62	12.7	77
Dataset 2			
Models	Testing Time (s)		Memory Usage (MB)
CNN	7.9		90.7
GRU	8.0		93.1
LSTM	11.4		122
A-CNN	9.0		110
A-GRU	9.1		150
A-LSTM	11.0		160
Transformer	11.6		90

TABLE 9: Models resource efficiency across datasets

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

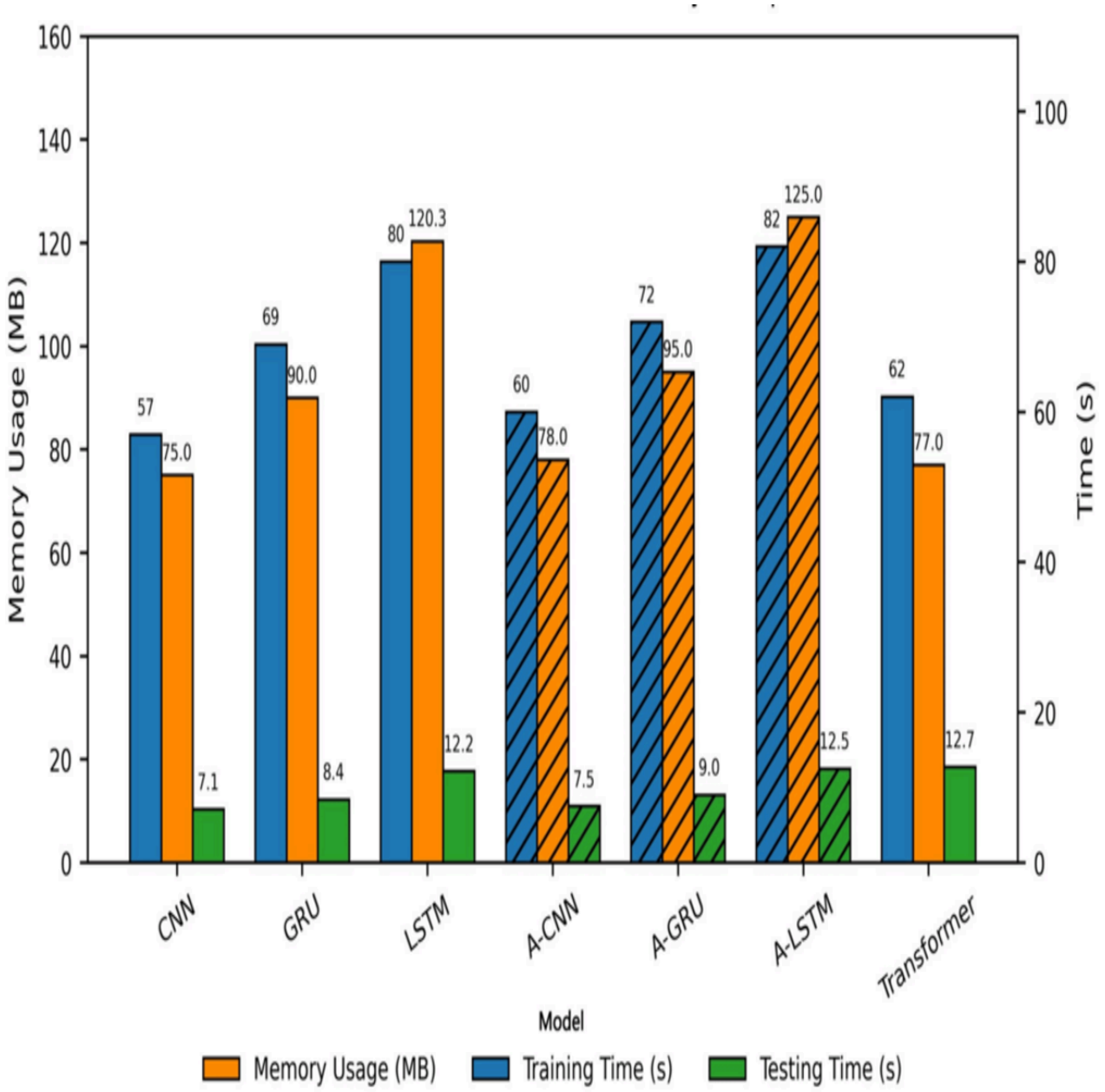

FIGURE 6: Model resource efficiency comparison (Dataset 1)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

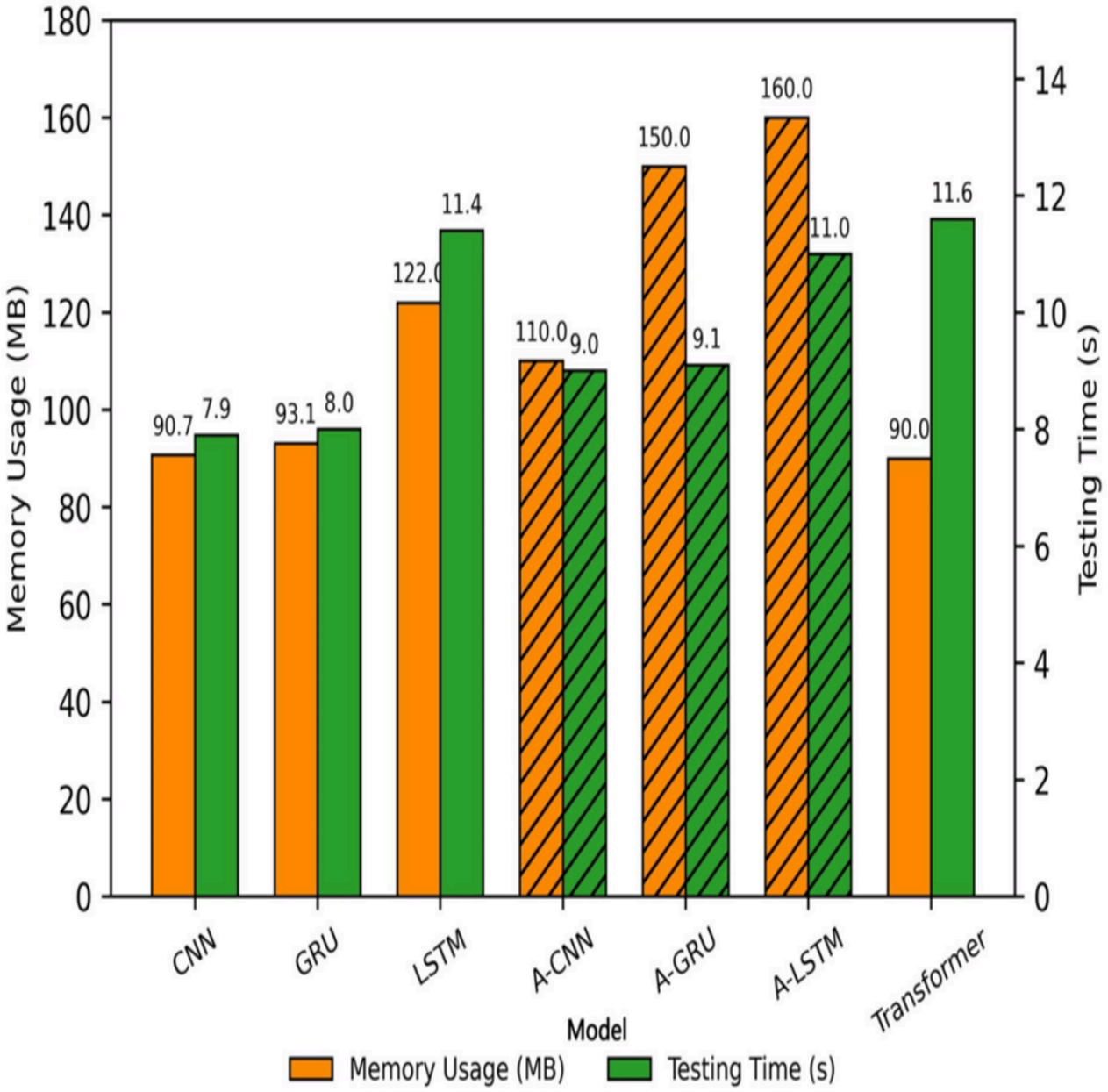

FIGURE 7: Model resource efficiency comparison (Dataset 2)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

The resource consumption of each model, detailed in Table *10* and Figures *8-9*, provides additional context for the performance metrics. This efficiency analysis is crucial for the practical deployment of systems, particularly in resource-constrained settings such as IoT gateways or edge devices. While attention augmentation consistently improved detection performance by lowering FNR, it introduced a modest computational overhead. The non-attention models, including CNN, GRU, and LSTM, are generally more computationally efficient in terms of both training and testing time. The addition of the attention mechanism enhances performance. However, it introduces overhead, as evidenced by the slight increases in training and testing time for the A-CNN, A-GRU, and A-LSTM models compared to their non-attention

counterparts. While competitive in training time, the Transformer model has the highest testing time among the attention-augmented models, which may be a consideration for real-time applications. Similarly, the LSTM and A-LSTM models are the most memory-intensive, especially on Dataset 2, which is an important factor to consider in resource-constrained environments. These results highlight a common trade-off in machine learning: models with superior predictive performance often require greater computational resources.

Ablation study

Dataset 1				
Model	F1-Score (Mean SD Error)	AUC	FPR	FNR (Mean SD Error)
A-CNN	0.996 (0.001)	0.971	0.003	0.004 (0.0006)
Ablated (CNN)	0.994 (0.002)	0.991	0.012	0.004 (0.0005)
A-GRU	0.996 (0.001)	0.983	0.022	0.001 (0.0003)
Ablated (GRU)	0.992 (0.002)	0.986	0.004	0.005 (0.0007)
A-LSTM	0.997 (0.001)	0.976	0.001	0.001 (0.0002)
Ablated (LSTM)	0.993 (0.002)	0.971	0.001	0.006 (0.0008)
Dataset 2				
Model	F1-Score (Mean SD Error)	AUC	FPR	FNR (Mean SD Error)
A-CNN	0.985 (0.003)	0.967	0.007	0.006 (0.0007)
Ablated (CNN)	0.979 (0.004)	0.967	0.010	0.008 (0.0009)
A-GRU	0.989 (0.002)	0.979	0.015	0.006 (0.0006)
Ablated (GRU)	0.985 (0.003)	0.974	0.008	0.012 (0.0011)
A-LSTM	0.988 (0.002)	0.969	0.005	0.001 (0.0003)
Ablated (LSTM)	0.950 (0.006)	0.943	0.006	0.004 (0.0008)

TABLE 10: Ablation study results: performance comparison of attention-augmented and ablated models (mean ± SD over 5 runs)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit; SD, Standard Deviation; AUC, Area Under The Curve; FPR, False Positive Rate; FNR, False Negative Rate

We conducted an ablation study to validate the contribution of the attention mechanism to the overall performance of our model. This involved systematically removing the attention layer from the attention-augmented models (A-CNN, A-GRU, and A-LSTM) and evaluating the performance of these "ablated" models. The purpose was to quantify the performance degradation when this key component was absent, thereby confirming its importance. The results of this study are summarised in Table *7* and Figures *8-9*.

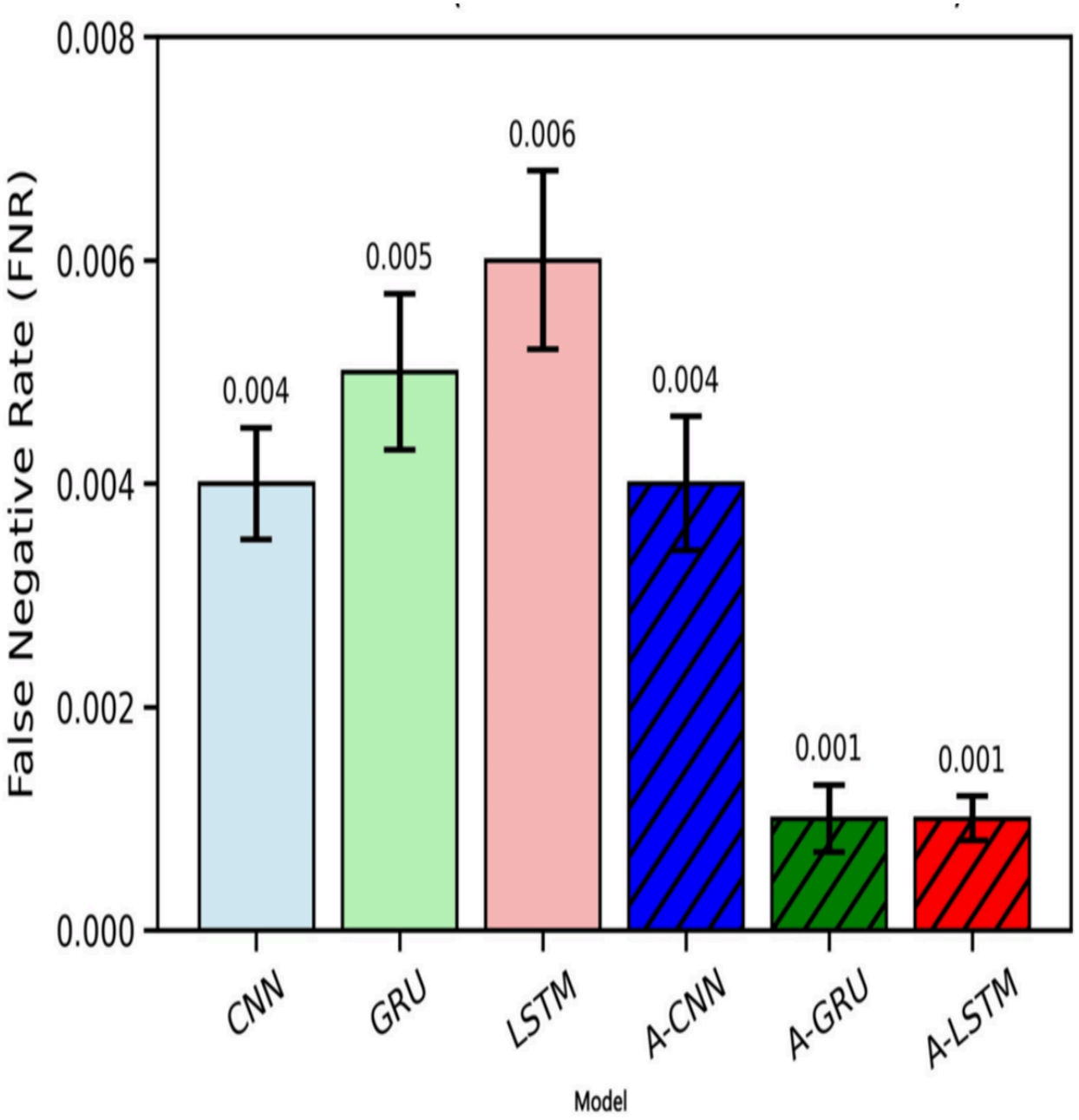

FIGURE 8: Ablation study: FNR comparison with and without attention mechanism (Dataset 1)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

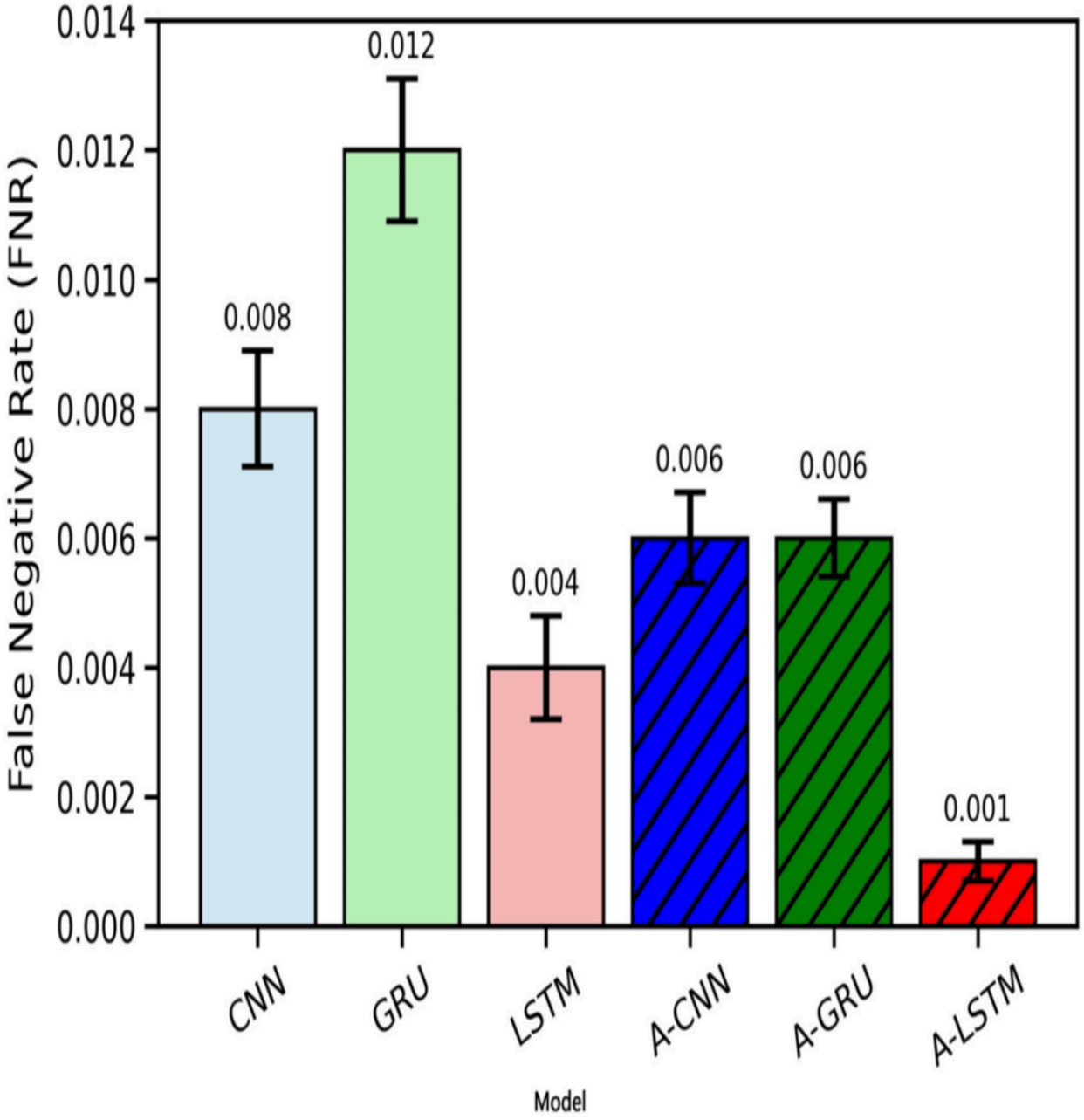

FIGURE 9: Ablation study: FNR comparison with and without attention mechanism (Dataset 2)

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit

The results in Table *6* and Figure *6* demonstrate that ablating the attention mechanism consistently degraded model performance across both datasets. This degradation was statistically significant, as indicated by non-overlapping standard deviations in most metrics. On Dataset 1, the LSTM's mean FNR increased from 0.001 (SD = 0.0002) to 0.006 (SD = 0.0008), a sixfold increase, highlighting the attention layer's crucial role in identifying malicious patterns. On Dataset 2, the trend held firmly: the LSTM's mean F1-score declined from 0.988 (SD = 0.002) to 0.950 (SD = 0.006), and its mean FNR rose from 0.001 (SD = 0.0003) to 0.004 (SD = 0.0008). Similarly, for the GRU on Dataset 2, the mean F1-score dropped from 0.989 (SD = 0.002) to 0.985 (SD = 0.003), while the mean FNR increased from 0.006 (SD = 0.0006) to 0.012 (SD = 0.0011).

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

These consistent increases in FN and decreases in F1-score across architectures confirm that the attention mechanism is a critical component for capturing discriminative sequential patterns and minimizing costly classification errors, particularly in security-sensitive contexts where missed detections are unacceptable.

Interpretability and visualization

In addition to quantitative performance, we analyzed the interpretability of the attention-augmented models. Interpretability is particularly important in malware detection, as security analysts must detect threats and understand the rationale behind automated decisions. By visualizing attention weights, we identify which API calls the models prioritize when classifying sequences as benign or malicious. The heatmaps are colored using a purple-green-yellow scale:

· Dark purple regions represent API calls with very low attention weights, meaning the model considered them less relevant.

· Green regions represent moderate attention weights, showing calls that contributed meaningfully but were not decisive.

· Bright yellow regions represent high attention weights, highlighting API calls that the model identified as most influential in the prediction.

This color scale provides an intuitive view of how the models distribute their focus: yellow zones indicate strong signals of malicious behavior, while purple zones indicate background or less critical activity. Furthermore, Figure *10* provides an interpretation of the CNN model.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

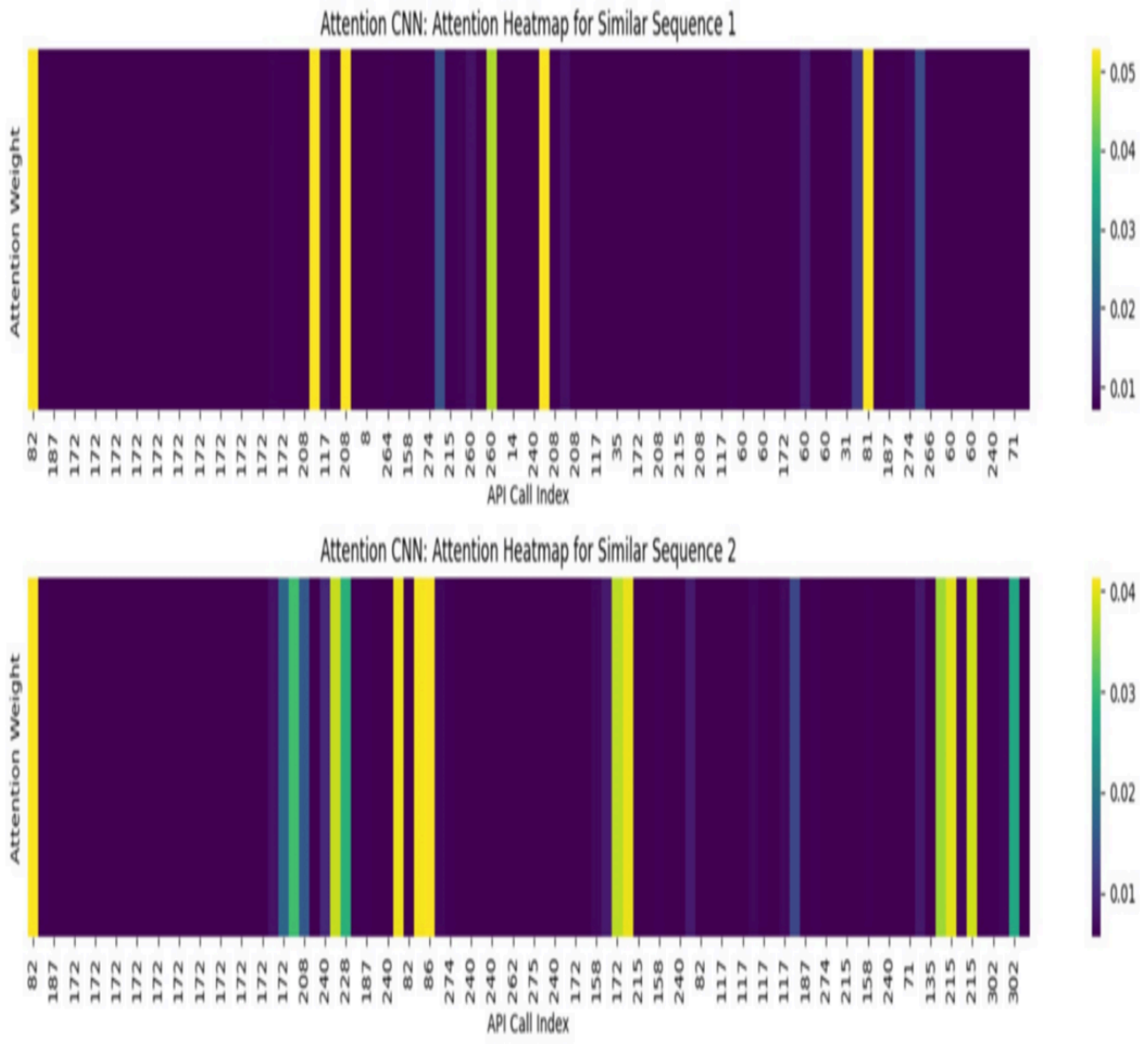

FIGURE 10: Attention CNN heatmaps for two similar malicious sequences. High attention weights are in yellow, with peaks concentrated on discriminative API calls, such as CreateProcess and WritePro-cess-Memory

API, Application Programming Interface; CNN, Convolutional Neural Network

In Figure *10*, the CNN with Attention assigns sharp peaks of yellow to a small number of API calls. This indicates that CNN isolates localized subsequences that are highly discriminative for malware classification. Most calls appear in purple, meaning the model largely ignored them. Figure *8* further shows the model interpretation for GRU.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

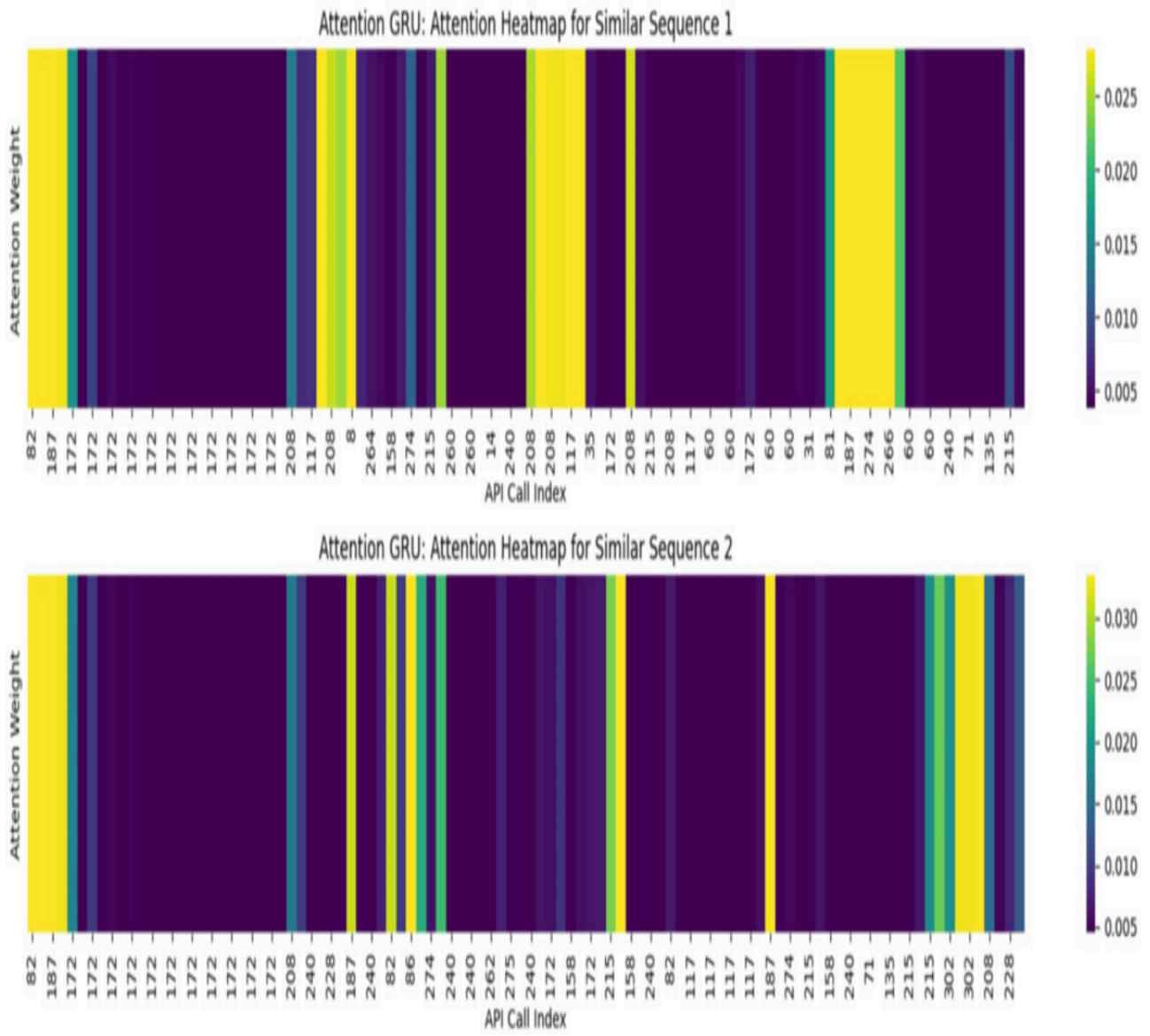

FIGURE 11: Attention GRU heatmaps for two similar malicious sequences. The GRU assigns attention more broadly across the execution sequence, with both green and yellow regions highlighting contextual relevance

GRU, Gated Recurrent Unit

In Figure *11*, the GRU shows a more distributed pattern, with attention spread across multiple calls (green and yellow zones). Unlike CNN, which focuses on isolated peaks, the GRU captures the temporal flow of API calls, recognizing malicious behavior in the context of surrounding calls rather than single events. Figure *9* shows the interpretation of LSTM.

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

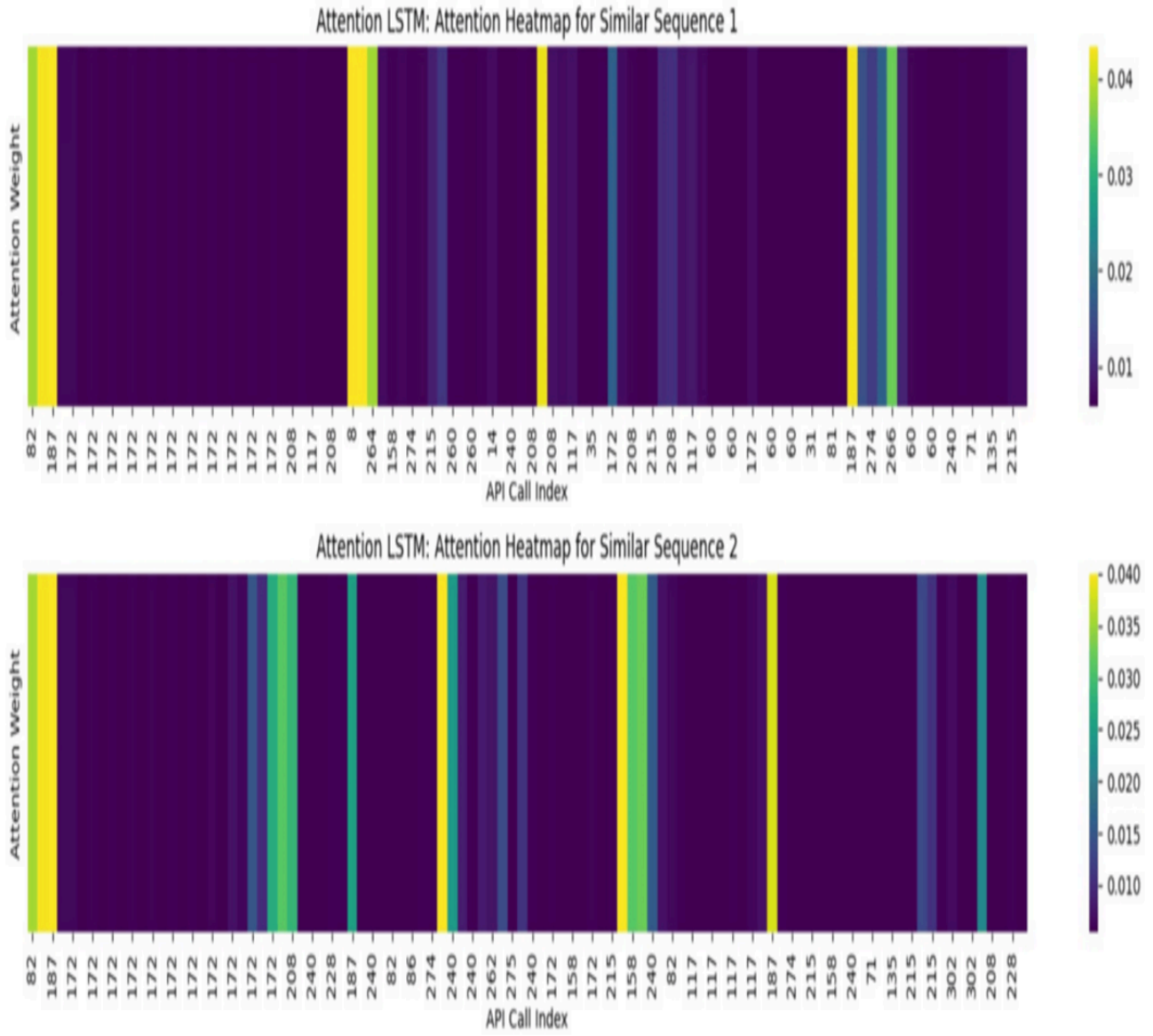

FIGURE 12: Attention LSTM heatmaps for two similar malicious sequences. The LSTM balances localized and distributed attention, with consistent yellow peaks across different samples

LSTM, Long Short-Term Memory

In Figure *12*, the LSTM demonstrates hybrid behavior. It highlights critical calls in yellow, similar to CNN, but also maintains green distributions across broader subsequences, like GRU. The consistency of yellow peaks across both sequences indicates stronger generalization, where the LSTM reliably attends to malicious traits even when sequences vary. Overall, combining color-coded attention heatmaps and model-specific patterns provides actionable interpretability. Security analysts can quickly identify which API calls drove the classification decision. Yellow regions indicate high-risk behaviors, green regions indicate supporting contextual signals, and purple regions indicate background activity. This bridges the gap between automated predictions and human-centered threat analysis. The heatmaps visually demonstrate that the models learned to assign higher attention weights to API calls associated with

malicious behaviors, such as CreateProcess and WriteProcessMemory. This offers a valuable layer of explainability for security analysts. We also confirm the faithfulness of the attention mechanism, as removing highly attended tokens significantly reduces prediction confidence.

Faithfulness and diagnosticity of attention mechanisms

Beyond visualization, we quantitatively assessed the faithfulness and diagnosticity of the attention mechanisms in our augmented models (A-CNN, A-GRU, A-LSTM) to validate their reliability as explanations. Faithfulness was measured by progressively masking the top-k% most highly attended API calls (k = 10, 20, ..., 50) in malicious test sequences from the Mal-API-2019 dataset and observing the degradation in model confidence, as indicated by the softmax probability for the malware class and the F1-score. A faithful mechanism should show a substantial drop when critical tokens are removed. On average across 500 randomly selected malicious samples: Removing the top-20% attended tokens caused a 42% drop in prediction confidence from ~0.98 to ~0.57 and a 0.28 drop in F1-score for the A-LSTM.

In contrast, removing randomly selected tokens resulted in only a 12% decrease in confidence and a 0.09 decrease in F1 score, confirming that high-attention tokens are indeed more impactful. Similar trends were observed for A-GRU, with a 38% confidence drop, and A-CNN, with a 35% drop; all attention-augmented models outperformed random baselines by more than 25% in degradation magnitude. These results indicate strong faithfulness, as perturbing attended regions significantly impairs performance, aligning with prior findings that attention weights correlate with feature importance in sequence classification tasks.

Diagnosticity was evaluated through counterfactual testing: for pairs of similar sequences (original malicious and minimally perturbed benign-like variants created by substituting or removing 5-10 non-critical APIs while preserving length), we measured the KL divergence between attention distributions. A diagnostic mechanism should exhibit a marked shift toward the altered regions. Average KL divergence was 0.82 for A-LSTM (high shift), compared to 0.31 for random perturbations. This shift consistently highlighted the modified APIs as gaining attention in the counterfactual benign case, supporting the notion that attention distributions are diagnostic of the model's reasoning rather than superficial artifacts. Overall, these metrics confirm that the integrated attention mechanisms provide not only plausible explanations, as shown in the visualizations, but also faithful and diagnostic explanations, strengthening their utility for security analysts. The quantitative faithfulness and diagnosticity scores are presented in Table *11*, and a comparison of our proposed models against prior literature is provided in Table *12*.

Model	Confidence Drop (Top-20% Removal)	F1 Drop (Top 20% Removal)	KL Divergence (Counterfactual Pairs)
A-LSTM	42%	0.28	0.83
A-GRU	38%	0.25	0.76
A-CNN	35%	0.22	0.71
Random Baseline	12%	0.09	0.31

TABLE 11: Faithfulness and diagnosticity metrics

A-LSTM, Attention-Long Short-Term Memory; A-CNN, Attention-Convolutional Neural Network; A-GRU, Attention-Gated Recurrent Unit; KL, Kullback-Leibler

Authors	Method	Dataset	Accuracy (%)	F1-Score (%)
Qian and Cong [10]	CAFTrans (Transformer-based)	Mal-api-2019	64.60	65.30
Agrawal, Stokes, Selvaraj and Marinescu [11]	ARI-LSTM	Mal-api-2019	93.00	NR
Zhao, Liu, Zhang and Zheng [14]	CNN-AttBiLSTM	CIC-IDS2017 & CIC-DDoS2019	95.80	95.86
Jo, Cho and Moon [15]	ViT-based	Androzoo	80.20	77.40
Huang, Du, Wang, Li and Yuan [16]	CoAtNet	Malicious code detection	96.60	96.57
Alshomrani, Albeshri, Alturki, Alallah and Alsulami [17]	TabNet with Attention	CIC UNSW-NBIS	74.00	NR
Our Work—CNN	Non-attention CNN	Oliveira (2019), Mal-api-2019	99.40	99.40
Our Work— A-CNN	Attention CNN	Oliveira (2019), Mal-api-2019	99.20	99.60
Our Work—GRU	Non-attention GRU	Oliveira (2019), Mal-api-2019	98.90	99.20
Our Work— A-GRU	Attention GRU	Oliveira (2019), Mal-api-2019	99.10	99.60
Our Work—LSTM	Non-attention LSTM	Oliveira (2019), Mal-api-2019	98.60	99.30
Our Work—A-LSTM	Attention LSTM	Oliveira (2019), Mal-api-2019	99.20	99.70
Our Work—Transformer	Transformer	Oliveira (2019), Mal-api-2019	98.50	99.20

TABLE 12: Comparison of the proposed model with existing research

NR = Not Reported

Future work

While our study demonstrates the effectiveness of attention-augmented deep learning models for malware detection via API call sequences, it is important to acknowledge its current limitations. First, the evaluation did not include adversarial robustness testing. In real-world cybersecurity, attackers often employ evasion techniques specifically designed to fool machine learning models. The absence of such testing means we have not assessed the models' resilience against sophisticated, adaptive threats. Second, our experiments relied on publicly available datasets from 2019. Although these datasets are well-established and contain a diverse set of malware families, they may not fully represent the characteristics of the most recent, evolving threats, such as fileless malware or AI-aided attacks. Third, the study focused solely on API call sequences as the input modality. In practice, integrating complementary data sources, such as system logs, network traffic flows, or memory dumps, can provide a more comprehensive and robust view of malicious behavior. Finally, while we reported computational efficiency metrics (training time, inference latency, memory usage), the models were not deployed and evaluated in a true resource-constrained edge environment. Consequently, their real-time performance and operational overhead in settings like IoT gateways or mobile devices remain to be validated.

To address the limitations above and extend the contributions of this work, several research directions are promising. First, future work should focus on improving the scalability and efficiency of attention-augmented models for deployment in resource-constrained environments. Techniques such as model pruning, quantization, or lightweight attention variants like Linformer and Performer can reduce computational overhead without compromising accuracy. Second, expanding the scope of input data to include multimodal sources such as system logs, network flows, and memory sequences would provide a richer context for anomaly detection and enhance real-world applicability. Third, adversarial robustness evaluation must be incorporated to ensure that models are resilient against evasion attacks. This could involve generating adversarial API sequences or testing against known adversarial samples. Finally, advancing interpretability remains a critical avenue. Future research could explore hybrid explainability approaches that combine attention visualizations with post hoc interpretability methods, such as SHAP and LIME, to provide security analysts with deeper insights into model decisions. By pursuing these directions, the gap between high-performance malware detection models and practical, trustworthy cybersecurity tools can be further narrowed.

Conclusions

This study presented a systematic and controlled evaluation of attention-augmented deep learning models for malware detection via API call sequence analysis. Our principal contribution is a rigorous, ablation-style comparison of CNN, LSTM, and GRU architectures with and without attention mechanisms under identical experimental conditions, with a Transformer serving as a pure attention-based baseline. This controlled design directly isolates the contribution of the attention mechanism from other architectural factors, addressing a key methodological gap in the existing literature. The results confirm that attention augmentation consistently improves detection performance, reduces false negative rates, and enhances model interpretability across all evaluated architectures, with the Attention-LSTM achieving the strongest overall performance on both the training-domain and independent test sets. The ablation study validates that these gains are attributable specifically to the attention mechanism rather than increased model complexity. The interpretability analysis further demonstrates that attention weights are both faithful and diagnostic, providing a reliable basis for human-interpretable threat analysis in operational cybersecurity environments. Collectively, these contributions establish attention augmentation as a principled and deployable enhancement for sequence-based malware detection systems.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Acquisition, analysis, or interpretation of data: John Adejoh

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6

Drafting of the manuscript: John Adejoh

Concept and design: Nsikak Owoh, Felix Ale, Moses Ashawa, Salaheddin Hosseinzadeh, Prasad Rajesh

Critical review of the manuscript for important intellectual content: Nsikak Owoh, Felix Ale, Moses Ashawa, Salaheddin Hosseinzadeh, Prasad Rajesh

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Data Availability Statements

The data supporting this study are openly available as follows: Dataset 1 (Training & Validation): The Malware Analysis Datasets: API Call Sequences is openly available in IEEE DataPort at 10.21227/tqqm-aq14. Dataset 2 (Independent Testing): The Mal-API-2019 dataset is openly available in PeerJ Computer Science at 10.7717/peerj-cs.346. Both datasets were used without modification beyond the preprocessing steps (tokenisation, sequence padding, and SMOTE balancing) described in Materials & Methods section. No new primary datasets were generated during this study. Formal dataset citations are provided in the References section.

References

1. Alsubaei FS, Almazroi AA, Atwa WS, Almazroi AA, Ayub N, Jhanjhi NZ: Adaptive malware identification via integrated SimCLR and GRU networks. Scientific Reports. 2025, 15:25309. 10.1038/s41598-025-08556-4
2. Owoh N, Adejoh J, Hosseinzadeh S, Ashawa M, Osamor J, Qureshi A: Malware detection based on API call sequence analysis: a gated recurrent unit-generative adversarial network model approach. Future Internet. 2024, 16:369. 10.3390/fi16100369
3. Gasmi H, Laval J, Bouras A: LSTM recurrent neural networks for cybersecurity named entity recognition. arXiv. 2024, 10.48550/arXiv.2409.10521
4. Alsuhaimi A, Janbi J: Attention mechanism for attacks and intrusion detection. International Journal of Computer Science and Information Technology. 2024, 16:1-16. 10.5121/ijcsit.2024.16601
5. Ahmed AA, Aliyu AA, Ibrahim M, Abdulkadir S, Ahmad MA, Tanko SA, Umaru IA: Enhancing network security through integrated deep learning architectures and attention mechanisms. FUDMA Journal of Sciences. 2024, 8:407-415. 10.33003/fjs-2024-0806-3010
6. Bhattacharya S, Khanna A, Ganapaneni S, Najana M: Attention-based deep learning frameworks for network intrusion detection: an empirical study. International Journal of Global Innovations and Solutions. 2024, 10.21428/e90189c8.eb32676c
7. Borré A, Seman LO, Camponogara E, Stefenon SF, Mariani VC, Coelho LdS: Machine fault detection using a hybrid CNN-LSTM attention-based model. Sensors. 2023, 23:4512. 10.3390/s23094512
8. Ye X, Ye Q, Yan X, Wang T, Chen J, Li S: Demand forecasting of online car-hailing with combining LSTM + attention approaches. Electronics. 2021, 10:2480. 10.3390/electronics10202480
9. Ye Y, Li T, Adjeroh D, Iyengar SS: A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR). 2017, 50:1-40. 10.1145/3073559.
10. Qian L, Cong L: Channel features and API frequency-based transformer model for malware identification. Sensors. 2024, 24:580. 10.3390/s24020580

11. Agrawal R, Stokes JW, Selvaraj K, Marinescu M: Attention in recurrent neural networks for ransomware detection. ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK. 2019, 3222-3226. 10.1109/ICASSP.2019.8682899

12. AlSobeh AMR, Gaber K, Hammad MM, Nuser M, Shatnawi A: Android malware detection using time-aware machine learning approach. Cluster Computing. 2024, 27:12627-12648. 10.1007/s10586-024-04484-6

13. Terawi N, Ashqar HI, Darwish O, Alsobeh A, Zahariev P, Tashtoush Y: Enhanced detection of intrusion detection systems in cloud networks using time-aware and deep learning techniques. Computers. 2025, 14:282. 10.3390/computers14070282

14. Zhao J, Liu Y, Zhang Q, Zheng X: CNN-AttBiLSTM mechanism: a DDoS attack detection method based on attention mechanism and CNN-BiLSTM. IEEE Access. 2023, 11:136308-136317. 10.1109/access.2023.3334916

15. Jo J, Cho J, Moon J: A malware detection and extraction method for the related information using the ViT attention mechanism on Android operating system. Applied Sciences. 2023, 13:6839. 10.3390/app13116839

16. Huang H, Du R, Wang Z, Li X, Yuan G: A malicious code detection method based on stacked depthwise separable convolutions and attention mechanism. Sensors. 2023, 23:7084. 10.3390/s23167084

17. Alshomrani M, Albeshri A, Alturki B, Alallah FS, Alsulami AA: Survey of transformer-based malicious software detection systems. Electronics. 2024, 13:4677. 10.3390/electronics13234677

18. Oliveira A: Malware Analysis Datasets: API Call Sequences. IEEE Dataport. (2019). https://doi.org/10.21227/tqqm-aq14..

19. Catak FO, Ahmed J, Sahinbas K, Khand ZH: Data augmentation based malware detection using convolutional neural networks. PeerJ Computer Science. 2021, 7:e346. 10.7717/peerj-cs.346

20. Bahdanau D, Cho K, Bengio Y: Neural machine translation by jointly learning to align and translate. arXiv. 2014, 10.48550/arXiv.1409.0473

21. Vaswani A, Shazeer N, Parmar N, et al.: Attention is all you need. arXiv. 2023, 10.48550/arXiv.1706.03762

How to cite this article:
Adejoh J, Owoh N, Ale F, et al. (March 30, 2026) Attention-Augmented Deep Learning Architectures for Malware Detection Through System Call Sequence Analysis. Cureus J Comput Sci 3 : es44389-026-00046-6. DOI https://doi.org/10.7759/s44389-026-00046-6