Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Next-Generation IDS: Leveraging Transformer and Hybrid Deep Architectures for Robust Cyber Threat Detection

Nour El Houda SENOUSSI

n.senoussi@univ-boumerdes.dz

University of Boumerdes

Cheikh SALMI

University of Boumerdes

Research Article

Keywords: cyberattacks, host-based intrusion detection system, deep learning, transformers

Posted Date: March 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-7715500/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Next-Generation IDS: Leveraging Transformer and Hybrid Deep Architectures for Robust Cyber Threat Detection

Nour El-Houda SENOUSSI[1,2*†] and Cheikh SALMI[1,2†]

[1]Department of Computer Science, Faculty of Science, University of Boumerdes, Boumerdes, 35000, Algeria.
[2]LIMOSE Laboratory, Faculty of Science, University of Boumerdes, Boumerdes, 35000, Algeria.

*Corresponding author(s). E-mail(s): n.senoussi@univ-boumerdes.dz;
Contributing authors: c.salmi@univ-boumerdes;
†These authors contributed equally to this work.

Abstract

As technology advances, so too do the risks posed against cyber security. Furthermore, Host-based Intrusion Detection Systems (HIDS) are vital towards recognizing and controlling cyberattacks and harmful actions within the operating system. Despite traditional Host-based Intrusion Detection Systems having their merits, there are issues related to feature engineering, sequencing noise, and scope of generalization towards new attack patterns. This paper presents a new architecture of Host-based Intrusion Detection Systems that is deep learning-based and uses transformer networks for intrusion detection through system call sequences. We demonstrate the ability of transformers in system call sequences and their recognition of long range dependencies and contextual patterns designed for next generation intrusion detection systems. The proposed approach opens up promising directions for building intelligent, context-aware, and robust host-based IDSs. The models were extensively evaluated on benchmark datasets AWSCTD and NGIDS. The comparison demonstrated the effectiveness of Transformer-based approaches over LSTM, BLSTM, and hybrid CNN-LSTM variants when tested under identical experimental conditions.

Keywords: cyberattacks, host-based intrusion detection system, deep learning, transformers

1 Introduction

Cyberattacks have increased significantly in recent years due to advances in engineering, information technology, and the introduction of computerization in almost every area of social life. The advent of the Internet of Things (IoT), remote working platforms, e-Commerce, cloud computing, and other online activities has greatly expanded the attack surface. These cyberattacks can manifest in a variety of forms, each designed to exploit and damage digital infrastructures and data integrity. These forms include: (1) denial of service (intentionally flooding a system or network to prevent legitimate requests from being processed, i.e., rendering the system inaccessible by flooding it with requests), (2) espionage (using malware to surreptitiously collect sensitive data from the target), data destruction (using malware to erase or disable data on a computer), (3) ransomware (malware that locks down access to resources with the aim of demanding a ransom from the victim to recover their data), and (4) phishing (a fraudulent method used by hackers to obtain sensitive information such as passwords or the target's identity by tricking them into believing they are interacting with a trusted peer).

These cyberattack vectors, among others, pose significant risks to individuals and institutions, potentially leading to substantial financial losses, reputational damage, and compromised security. A defense system called an intrusion detection system (IDS) is designed to identify such malicious behavior. The first generation of IDS, known as traditional IDS [1], was based on attack signatures, making it ineffective against new attacks. Today, this generation has been completely replaced by machine learning and deep learning (ML and DL), two branches of artificial intelligence that give computers the ability to learn from sample data and solve the most complex problems such as cyberattack detection. An IDS consists of three successive segments. First, input data are collected from the network or system to identify potential signs of attack. Then, these data are analyzed to assess and detect suspicious or malicious activity. Finally, identified attacks are described and classified to allow for better understanding of threats and effective incident management. Recent advances in input data analysis have made it possible to leverage machine learning and deep learning techniques not only to anticipate typical and abnormal network activity but also to identify previously unknown attacks. Many intrusion detection systems have been developed using machine learning techniques. These techniques are generally based on supervised, unsupervised, semi-supervised, weakly supervised, and reinforcement learning (RL). In the field of deep learning, numerous studies have been proposed for intrusion detection systems, focusing on feature extraction, classification, or a combination of both. These approaches primarily rely on Deep Neural Networks (DNNs), with significant variations in architectural design. Other research efforts are based on specific models such as autoencoders, recurrent neural networks (RNNs), long-short-term memory networks (LSTMs), convolutional neural networks (CNNs), adaptive networks, and deep reinforcement learning (DRL).

To address the growing complexity and diversity of modern cyber threats, our approach leverages advanced deep learning architectures, with a particular focus on Transformer models and hybrid combinations such as CNN, LSTM, and Bidirectional long-short-term memory networks (BLSTMs). Unlike traditional models that

may struggle to capture long-range dependencies or spatial-temporal patterns in intrusion data, Transformers offer a powerful attention mechanism capable of modeling complex relationships across sequences. Meanwhile, CNNs excel at extracting local features, LSTMs are effective for sequential data, and BLSTMs enhance context understanding by processing information in both directions. By integrating these complementary strengths, our hybrid models aim to improve detection accuracy, generalization to unseen attacks, and robustness against noisy or imbalanced data. This approach is particularly well-suited for host-based intrusion detection, where the subtlety and variability of attack patterns demand more expressive and adaptable learning frameworks.

Our contributions include (1) the integration of Transformer-based attention mechanisms with sequential and convolutional networks to enhance detection performance, (2) the development and comparison of multiple hybrid model architectures, and (3) a comprehensive experimental evaluation demonstrating improved detection accuracy and robustness across diverse intrusion types. This work moves beyond our foundational study [2]—which demonstrated a Transformer-based system for binary intrusion classification on the ADFA-LD dataset—to validate its scalability and advanced capabilities. In this paper, we introduce a comprehensive evaluation across two substantially larger datasets (AWSCTD and NGIDS), including a novel assessment of the model's performance in multi-class classification to distinguish among various specific threat types. The findings of this paper confirm the significant potential of deep and hybrid neural architectures, providing clear evidence and critical insight into their effectiveness for robust host-based intrusion detection. The rest of the paper is organized as follows: Section 2 presents the stat-of-the-art in IDS. Section 3 provides the methodology, including a general overview of the transformers and the proposed model. Section 4 presents the descriptions of the datasets. In Section 5 presents the experimental setup. Section 6 presents the discussion of the results. Finally, the paper concludes in Section 7, by discussing the future directions.

2 Related work

In this section, we present various deep learning algorithms proposed in the context of IDSs, with an emphasis on recent research work.

A scalable hybrid intrusion detection system (SHIA) was proposed in [3]. This approach is based on distributed deep learning, using DNNs to analyze and process large volumes of data in real time. Its main objective is to effectively address the challenges of detecting and classifying unforeseen and difficult-to-predict cyberattacks. Advanced experiments were conducted to tune the hyperparameters of DNNs using the KDD Cup99 dataset, and the models were further validated on other datasets such as NSL-KDD, UNSW-NB15, Kyoto, WSN-DS and CICIDS 2017. The results show that DNNs effectively capture high-dimensional features and outperform traditional classifiers. In [4], the authors present a hybrid intrusion detection model that combines Bidirectional Recurrent Neural Networks (Bi-RNN), LSTM and Gated Recurrent Unit (GRU). This model was designed to overcome the limitations of traditional IDS by exploiting the sequential learning capabilities of deep models. The evaluation on the

CICIDS2017 dataset achieved an accuracy of 99.13%. This model also outperforms conventional methods such as the Naïve Bayes classifier in terms of both accuracy and false positive rate, even when using only 58% of the dataset's attributes. Furthermore, the study provides a comparative analysis of LSTM and GRU in the context of RNNs, confirming the superior performance of the hybrid model in intrusion detection scenarios.

The authors in [5] proposed an innovative method for HIDS using SVM for classification and a short-sequences-based approach for feature extraction. They achieved an accuracy of 0.70 with a FAR of 0.25. The main drawback of the proposed frequency-based techniques is the inability to extract specific features that clearly distinguish attack traces from normal traces. The study in [6] proposed a hybrid anomaly detection system, where CNN with LSTM is employed. A window size is used to define the system call sequence. According to the experimental results, the proposed technique improved the anomaly detection rate to 0.96 while reducing training time. Consequently, this method lowers the false alarm rate.

In [7], the authors developed complex dual-stream architectures such as LSTM-FCN and GRU-FCN on the AWSCTD dataset and compare them with simpler CNNs. The results indicate that the increased complexity of dual-stream models does not improve the detection accuracy of univariate time series data and leads to longer training and detection times. The also show that the simplified AWSCTD-CNN-static model achieves comparable accuracy with significantly faster performance, making it more practical for real-time intrusion containment.

To improve accuracy and reduce training time in complex network environments, the authors in [8] integrated two deep learning models, SDAE-ELM for network intrusion detection and DBN-Softmax for host intrusion detection. They leveraged stacked denoising autoencoders (SDAE), deep belief networks (DBN), and powerful classifiers such as ELM and Softmax to overcome the limitations of traditional DNNs. Mini-batch gradient descent optimization was used to further improve training efficiency. Experimental validation on several benchmark datasets, including KDD Cup99, NSL-KDD, UNSW-NB15, CIDDS-001, and ADFA-LD, showed that the proposed models outperform conventional machine learning approaches in both detection accuracy and response time. A host-based intrusion detection system (HIDS) was developed in [9]. The authors use a C4.5 decision tree classifier based on the consolidated tree construction (CTC) algorithm. To address the class imbalance problem, supervised relative random sampling (SRRS) was introduced during the preprocessing phase, along with a multi-class feature selection mechanism to identify optimal features. Evaluated on the NSL-KDD and CICIDS2017 datasets, this IDS achieved detection accuracies of 99.96% and 99.95%, respectively, demonstrating its robustness and effectiveness compared to existing state-of-the-art systems.

Although intrusion detection systems (IDS), and more specifically HIDS, have been widely studied, few works have explored the use of hybrid deep learning models combined with attention mechanisms. These approaches, while promising for improving both accuracy and interpretability of detection systems, remain largely underexploited in the context of HIDS. To the best of our knowledge, only one work has so far applied the attention mechanism, but this one exclusively concerns the field of network-based

intrusion detection systems (NIDS), thus highlighting a notable gap in the current literature for host-based systems.

3 Methodology

This section presents the design and implementation of the proposed HIDS using a deep learning approach centered around Transformer architectures. As shown in Figure 1, the system is structured into three primary stages: Preprocessing, Feature Encoding and Learning, and Attack Classification.

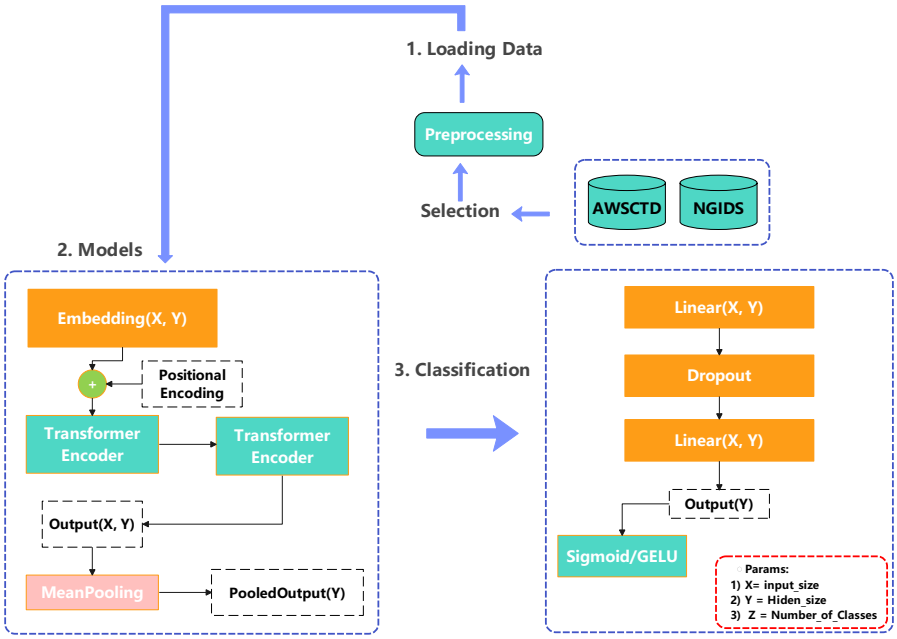


Fig. 1 General approach Pipeline

3.1 General Overview

3.1.1 Preprocessing and Selection

The datasets are presented as flat files distributed across different directories. After reading the data, the preprocessing phase consists of formatting the data by converting each system call sequence and its associated label into the appropriate format (tensor) to ensure compatibility with deep learning models. This phase also removes repetitive system calls (while preserving the semantics of the trace) and resolves ambiguities between attack and non-attack samples. In other words, it identifies sequences with conflicting labels (i.e., the same sequence labeled both as normal and as attack) and resolves the ambiguity by rebalancing the dataset.

3.1.2 Embedding and Feature Encoding

After preprocessing, the selected input features are embedded into a continuous representation to prepare them for sequence modeling. Positional encoding is incorporated to retain the sequence order of events, which is crucial for identifying attack patterns within the system calls (sys-calls). The two Transformer Encoder blocks are used in succession to process these embeddings to capture both local and long-range dependencies in the data. Then, an average pooling layer is applied to aggregate the sequence representations into a fixed-size pooled output, ensuring the model can handle variable-length sequences.

3.1.3 Classification Layer

The selected input features X are clustered and then subjected to a series of linear transformations, incorporating losses for regularization and nonlinear activations such as Sigmoid or GELU functions to improve learning capacity and classification. The final output layer associates the hidden size Y with the number of output classes Z corresponding to different attack types or normal behavior. This structure allows for efficient binary or multiclass classification, depending on the application scenario.

3.2 Background

The artificial intelligence domain has experienced a revolution due to the rapid evolution of transformers, which are a type of deep learning model used for Natural Language Processing (NLP) tasks. Transformers, introduced by Google in 2017 [10], were developed to handle sequential data in a parallelized manner. To mitigate the problems of RNNs, such as limited memory, transformers introduce an alternative approach. Instead of using recurrence to process long sequences, transformers convert the sequential input data into a representative vector using a mechanism called Attention. This mechanism allows the model to focus on specific words or parts of the input sequence while predicting the output, regardless of how distant those elements are. The attention mechanism computes a weighted sum of input vectors, where the weights represent the importance of each input position in contributing to the representation of another position. It is defined as follows:

$$A(i) = \sum_{j=1}^{T} \alpha(i,j) x_j \tag{1}$$

Here:

- $A(i)$ is the output representation at position i.
- x_j is the input vector at position j.
- $\alpha(i,j)$ is the attention weight, indicating how much position i attends to position j.
- T is the total number of input positions in the sequence.

The attention weight $\alpha(i,j)$ is obtained using the softmax function applied over the dot products of the input vectors:

$$\alpha(i,j) = \frac{\exp\left(x_i^\top x_j\right)}{\sum_{j'=1}^{T} \exp\left(x_i^\top x_{j'}\right)} \tag{2}$$

Where:

- $x_i^\top x_j$ is the dot product (similarity) between the input vectors at positions i and j.
- The numerator computes the unnormalized similarity between x_i and x_j.
- The denominator normalizes the weights across all positions j, ensuring sum to 1.

This mechanism allows the model to focus on different parts of the input sequence when computing the representation for each position, enabling the capture of long-range dependencies and contextual relationships. The process is repeated for each position i in the sequence.

The transformer architecture, while inspired by RNNs, is built on an encoder-decoder structure. Each component is composed of stacked layers. The encoder has two layers: Multi-Head Attention and Feed-Forward Network. The decoder consists of three layers: Multi-Head Attention, Multi-Head Encoder-Decoder Attention, and Feed-Forward Network. The encoder's main role is to transform input tokens into contextualized embedding, with the output of its final layer serving as input to the decoder. Transformers provide several pre-trained models to perform processing tasks on text; such as classification, summarization, and translation. Notable examples include BERT [11], GPTv1-3 [12], RoBERTa [13], T5 [14] and XLNet [15].

3.3 Proposed Models

The proposed models represent different instantiations of the general pipeline presented in Figure 1. These instantiations vary depending on the dataset used, the main difference being the preprocessing, as the datasets are not of the same nature. Moreover, the type of classification, whether binary or multi-class, also influences these variations. These models share a common "Encoder-Only" architecture, suitable for a classification objective. The differences between these models are only in the upper and lower layers, as well as in their respective parameters.

Using the **AWSCTD** dataset, we designed three models: **AWS-BIN** for binary classification (attack vs. clean), **AWS-MULTI-A** for multiclass classification (with only attack types), and **AWS-MULTI-AC** for multiclass classification incorporating both attack and clean samples.

Using the **NGIDS** dataset, we designed two binary models, each based on a distinct sequence construction strategy. The **NGIDS-BIN-SW** model was created by ordering events chronologically and applying a *sliding window* to extract system call subsequences. Each sequence was labeled as an attack if it contained any malicious calls, otherwise as normal. The **NGIDS-BIN-PT** model was developed by grouping events by process ID to form system call traces representing process activity. Each trace was labeled as an attack if any event within it was malicious. This approach enables both precise intrusion detection and proactive prevention by monitoring process-level behavior.

Table 1 summarizes the developed models (Model names are abbreviated for clarity purposes).

Table 1 Summary of Developed Models

Model	Dataset	Type
AWS-BIN	AWSCTD	Binary
AWS-MULTI-A	AWSCTD	Multiclass (attacks)
AWS-MULTI-AC	AWSCTD	Multiclass (attacks+clean)
NGIDS-BIN-SW	NGIDS	Binary (sliding window)
NGIDS-BIN-PT	NGIDS	Binary (process traces)

4 Datasets Descriptions

In this section, we will present the properties and details of the datasets used namely AWSCTD and NGIDS.

4.1 AWSCTD

The AWSCTD dataset [16] is one of the largest of Windows system call collections, containing 112.56 million calls from 12,110 malicious executable samples and 16.3 million calls from 3,145 benign samples. It covers various attack types such as worms, trojans, and Backdoors. This dataset is divided into three subsets: **Allmalware** (malware traces only), **Allmalwareplusclean** (malware and clean behavior), and **Malwareplusclean** (malware vs. benign traces). Each subset is orgonized into two folders; one contains the original sequences, including all consecutive system calls, while the other folder holds sequences where identical consecutive calls are limited to two. Within these folders, CSV files provide sequences of varying lengths (10 to 1000 system calls per line), enabling the evaluation of detection performance at different granularities. In total, 66 files were generated for the training of deep learning models. Table 2 summarizes the dataset content.

Table 2 AWSCTD Dataset Details

Attack Name	Number of Traces
Worm	4,765,030
Trojan	3,502,180
Backdoor	2,688,120
Normal	17,340,354

4.2 NGIDS

The NGIDS (Next-Generation Intrusion Detection System) dataset [17, 18] was generated in an emulated Linux-based environment to provide realistic and comprehensive records of system activities. It contains 99 host log files with millions of system call sequences and detailed event information (system calls, paths, categories, timestamps, etc.) related to events that occurred during normal and abnormal activities, as well as PCAP files capturing network traffic. This dual structure makes NGIDS suitable for both HIDS and NIDS (Hybrid IDS). A summary of the dataset content is provided in Table 3.

Table 3 NGIDS-LD Dataset [17]

File type	Feature	Quantity
ground-truth csv	Total Records	313,926
	Attributes	7
99 csv files of host logs	Total Records	90,054,160
	Attack records	1,262,426
	Normal Records	88,791,734
	Attributes	9
NGIDS.pcap	Total Capture Packets	1,094,231
	Unique IPs	18

5 Experimental Setup

In this section, we present and discuss the various parameters used to optimize models and achieve the best results. These parameters include variables determining the structuring of training, validation, and testing data, the model structure (number of layers, dimensions, etc.), the batch size, the number of iterations (epochs), the learning rate, and the metrics used to evaluate model performance.

5.1 Implementation Details

All models were tested under identical experimental conditions, specifically ensuring: (1) fixed data splits: the same training, validation, and testing sets were used across all models to ensure direct comparison of generalization ability (2) uniform training Regimen: all models were trained for the same number of epochs, utilizing the same batch size and the same optimization algorithm (e.g., Adam), often with the same initial learning rate Schedule (3) controlled Input representation: all models received the same tokenization method and vocabulary, and the input sequences were processed to the same maximum length and (4) equivalent computational resources: the comparison was performed using the same GPU hardware, NVIDIA Tesla P100 with 16Go of HBM2 memory, to ensure a fair measurement of training time, inference speed, and

memory usage. The model consists of two transformer encoder layers with a dimension of 512 and eight multi-head attention mechanisms. It was trained with 11 different sizes, ranging from 30 to 120 iterations, using a learning rate between [1e-5,1e-6], and a batch size of 32.

The AWSCTD model for binary classification was trained with an optimal setup of 60 epochs, a batch size of 32, and Adam as the optimizer. The learning rate and activation function were tuned for each classification type to ensure optimal performance, with Sigmoid selected as the activation function. For multi-class classification with AWSCTD, the best configuration was found with a learning rate of 1×10^{-5}, cross-entropy as the loss function, and GELU as the activation function.

For the NGIDS dataset, binary classification (attack vs. normal) was carried out with the optimal hyperparameters that yielded the best results: 100 training epochs, a batch size of 32, a learning rate of 1×10^{-6}, binary cross-entropy as the loss function, and Sigmoid as the activation function. Table 4 summarizes the optimal settings of all models (LR indicates learning rate, BCE=Binary Cross Entropy, CE=Cross Entropy).

Table 4 Optimal hyperparameters for each model

Model	**Type**	**Epochs**	**Batch**	**LR**	**Loss**
AWS-BIN	Binary	60	32	1×10^{-5}	BCE
AWS-MULTI-A	Multiclass (attacks)	60	32	1×10^{-5}	CE
AWS-MULTI-AC	Multiclass (attacks+clean)	60	32	1×10^{-5}	CE
NGIDS-BIN-SW	Binary (sliding window)	50	128	1×10^{-5}	BCE
NGIDS-BIN-PT	Binary (process traces)	100	32	1×10^{-6}	BCE

5.2 Evaluation Metrics

To assess the performance and effectiveness of our model, we relied on the confusion matrix [19], which is commonly used in classification tasks to summarize prediction results and identify model errors. The rows of the matrix represent the actual classes, while the columns represent the predicted classes, as presented in Table 5.

Table 5 Confusion Matrix

Actual	**Predicted**	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Where, TP denotes true positives, TN denotes true negatives, FP for false positives (Type I error), and FN for false negatives (Type II error), as commonly defined in

test theory or decision theory. In our case, TP refers to malware samples correctly classified as malware, FN refers to malware samples incorrectly classified as clean, FP refers to clean samples incorrectly classified as malware, and TN refers to clean samples correctly classified as clean.

Using this matrix, several widely adopted metrics can be derived, including accuracy, precision, recall, F1-score, Matthews Correlation Coefficient (MCC), and error rate [20]. These metrics are calculated using the following formulas : (3), (4), (5), (6), (7) and (8), respectively:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{F1-Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{7}$$

$$\text{Error Rate} = 1 - \text{Accuracy} \tag{8}$$

Additionally, the Receiver Operating Characteristic (ROC) curve was used to evaluate the model's performance. It provides a graphical representation of the trade-off between the True Positive Rate (TPR), also known as sensitivity or recall, and the False Positive Rate (FPR), illustrating the model's ability to discriminate between classes across various classification thresholds.

5.3 Baseline Models

In addition to the transformer model, we implemented four baseline deep learning and hybrid algorithms based on LSTM: LSTM, BLSTM, CNN-LSTM, and CNN-BLSTM.

Due to their capability of retaining and updating information over long periods, LSTM and BLSTM networks have been applied successfully for intrusion detection tasks. As in all architectures of RNNs, LSTMs are prone to vanishing and exploding gradients, but they are equipped with mechanisms that alleviate these problems, thereby enabling more stable and effective modeling of long-range dependencies. BLSTMs enhance this feature by allowing forward and backward processing of input sequences, which provides richer contextual information as well as temporal structure to be taken into account. This is particularly useful for sequential data, such as system call traces present in the used dataset. Hybrid architectures, such as CNN-LSTM

and CNN-BLSTM, combine the strengths of CNNs and recurrent structures: CNNs extract local spatial features from input sequences, while LSTM layers capture temporal dynamics. This integration enables more robust and accurate modeling of system behavior for effective intrusion detection.

6 Results and Discussion

The five developed models were executed on the Kaggle platform using a T100 GPU and validated on their respective datasets. To achieve more reliable and robust performance estimates, the entire experiment, including both training and evaluation, was repeated 10 times. The mean values of the performance metrics, such as accuracy and precision, were then calculated across these repetitions.

6.1 AWS-BIN

The obtained results are presented in Tables 6, 7 and 8, and in Figures 2, 3, 4 and 5.

In Table 6, we evaluate the performance of AWS-BIN relative to traditional models with varying input sequence lengths (Counts denote the number of system calls). At sequence length 40, AWS-BIN accuracy is the highest at 94.23%, above LSTM (92.9%), GRU (92.4%), CNN (94), and SVM (92). For longer sequences (e.g. 600, 800, 1000), CNN is better than AWS-BIN. With a sequence length of 1000, CNN outperforms AWS-BIN with the highest accuracy at 99.3%, highlighting its strength in dealing with long sequences. Despite that, AWS-BIN has strong early performance with sequence lengths of 10 - 200 calls, increasing the efficiency of early intrusion detection.

Table 6 AWS-BIN Accuracy rates (%) across varying sequence lengths compared to [21, 22]

Counts	LSTM	GRU	CNN	SVM	AWS-BIN
10	88.6	88.6	88.9	88.4	**89.31**
20	92.0	92.4	93.0	91.8	**93.10**
40	92.9	92.4	94.1	92.0	**94.23**
60	93.2	93.0	93.5	91.9	**93.98**
80	93.0	93.2	**94.5**	91.7	93.41
100	93.0	92.1	**94.5**	91.2	94.08
200	92.4	91.8	**94.9**	88.9	94.40
400	89.2	92.0	**94.8**	88.7	94.52
600	85.8	90.1	**98.6**	87.1	96.92
800	85.1	82.8	**98.9**	87.0	97.39
1000	83.8	72.4	**99.3**	87.0	97.89

In Table 7, we include other hybrid deep learning models based on LSTM with AWS-BIN for comparison across sequence lengths. Up to 400 calls, AWS-BIN maintains a competitive accuracy and bests other models at lower counts (for example, 20 calls at 93.10% versus others sitting around 92-93%). Hybrid models, led by CNN-BLSTM, take command from 600 calls with extreme levels of accuracy, claiming 99% when hitting 1000. Even so, AWS-BIN still maintains a sturdy 97.89% at 1000.

Table 7 AWS-BIN accuracy rates (%) across varying sequence lengths compared to the implemented LSTM-Based approachs

Count	LSTM	BLSTM	CNN-LSTM	CNN-BLSTM	AWS-BIN
10	**90**	89	89	**90**	89.31
20	93	92	93	93	**93.10**
40	94	94	94	94	**94.23**
60	94	94	**95**	94	93.98
80	93	**94**	93	95	93.41
100	92	**95**	94	**95**	94.08
200	92	**95**	93	94	94.40
400	91	93	93	93	**94.52**
600	97	96	**98**	97	96.92
800	**98**	96	**98**	**98**	97.39
1000	98	95	**99**	**99**	97.89

Table 8 AWS-BIN performance metrics on the First 100 system calls -(MalwarePlusClean2)

Acc (%)	Prec (%)	Rec (%)	F1 (%)	TPR (%)	FNR (%)	FPR (%)	TNR (%)	Err (%)	MCC
94.08	98.36	94.29	96.28	94.29	5.71	6.83%	93.17%	1.6	0.82

Acc = Accuracy, Prec = Precision, F1 = F1-Score, Err = Error Rate

Table 8 suggests, from the three prior tables, that the strength of AWS-BIN hinges upon its early stage performance; an arsenal of low false positives, strong metrics, and high precision for real-time intrusion detection appropriation.

From Figures 2 and 3, it can be seen that the model's accuracy incrementally improves over the epochs, with the curves being very close to each other, demonstrating effective generalization and little overfitting. Towards the end of the epochs, the accuracy curves for both models level off at around 96%, demonstrating excellent classification performance. Furthermore, the gradual and steady decline in both training and validation loss demonstrates the effectiveness of the learning process. In addition, the small gap from training to validation loss indicates that the model does not suffer from overfitting and generalizes well to unseen data, which means that the loss values dropped from roughly 0.82 to 0.22 depicting a considerable reduction in errors.

The remaining models—AWS-MULTI-A, AWS-MULTI-AC, NGIDS-BIN-SW, and NGIDS-BIN-PT—exhibited similar trends during training and validation, with consistent convergence of accuracy and loss curves, and no significant overfitting. Their detailed training curves are omitted from this paper for the sake of brevity.

In Figure 4, the receiver operating characteristic (ROC) curve illustrates that the model performs well when the False Positive Rate (FPR) is relatively low (near 0). When True Positive Rate (TPR) is high (close to 1) while also being lower than FPR, it shows that the model has strong discrimination power. The model achieves

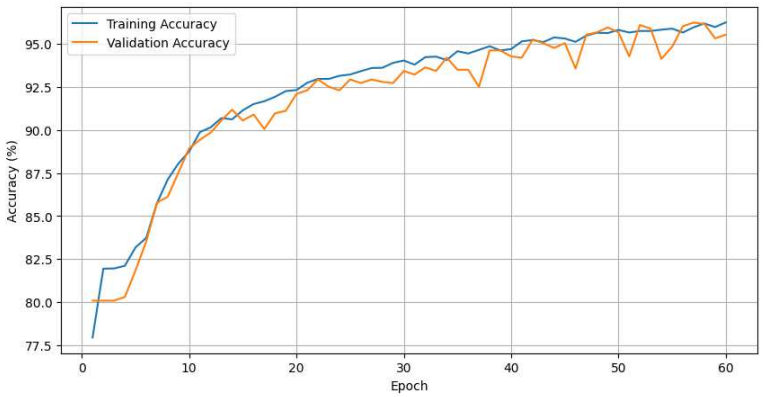


Fig. 2 AW-BIN training Accuracy over epochs

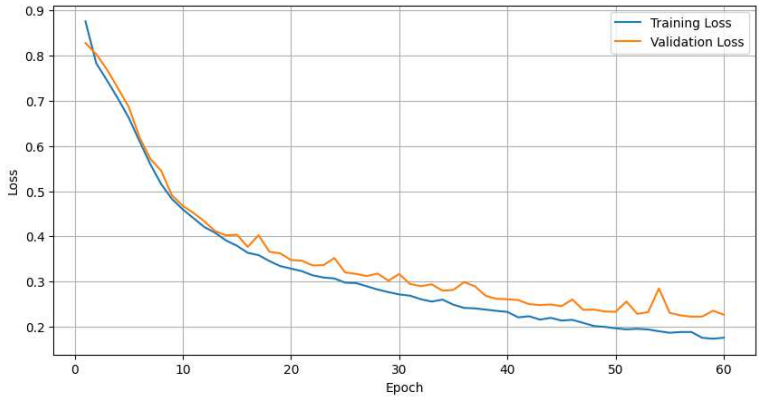


Fig. 3 AW-BIN training accuracy loss epochs

a sensitivity (TPR) of 94.29% and a specificity of 93.17%, confirming its effectiveness in correctly identifying both positive and negative cases.

The confusion matrix in Figure 5 confirms that the model is highly effective in detecting malware, with very few missed threats (false negatives). Although its precision is slightly impacted by a moderate number of false positives, the overall accuracy and recall remain excellent, making the model well-suited for malware detection tasks where minimizing undetected threats is critical.

6.2 AWS-MULTI-A

The obtained results are depicted in Tables 9 and 10, and the confusion matrix is shown in Figure 6.

Table 9 presents the accuracy rates obtained for classifying attack types in different models. None of the models achieved an accuracy of 90% for the first two system call sequences lengths (10 and 20 syscalls). However, all models except LSTM, exceed and

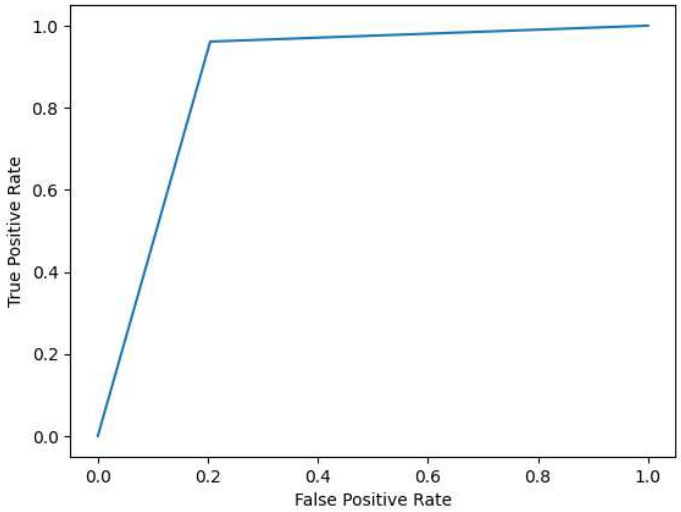


Fig. 4 AW-BIN Receiver operating characteristic

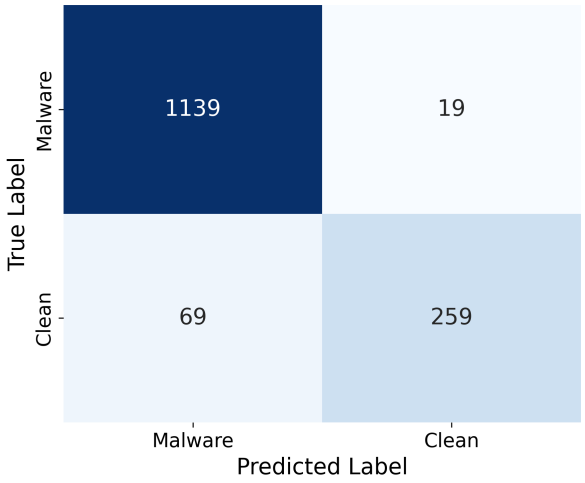


Fig. 5 AWS-BIN confusion matrix

surpass this threshold for sequences of the first 40 system calls. The SVM method achieved the best results for the 60 syscall sequences, with an accuracy close to that of AWS-MULTI-A. Our model reached a maximum accuracy of 93.17% for the 100 syscall sequences and achieved the second-best performance for longer sequences, after the CNN model. Sequence-based models, such as LSTM and GRU, produced poor results on long sequences (400, 600, 800, and 1000 syscalls), unlike AWS-MULTI-A and CNN.

Table 10 represents the performance metrics of AWS-MULTI-A on the first 100 system calls. We remark that the classes AdWare, Trojan and WebToolbar are well

Table 9 AWS-MULTI-A accuracy rates (%) across varying sequence lengths compared to [21, 22] – (AllMalware)

Count	LSTM	GRU	CNN	SVM	AWS-MULTI-A
10	88.1	88.3	87.3	89.4	**89.45**
20	88.8	88.1	89	89.4	**89.45**
40	89.1	90.6	91.2	91.6	**91.67**
60	88.2	90.5	91.2	**91.9**	91.84
80	91.8	91.6	92.3	92.7	**93.09**
100	91.6	91.9	92.8	92.4	**93.17**
200	90.4	91.5	92.7	89.6	**92.82**
400	87.6	90.3	**93**	87.3	92.20
600	87.4	91.4	**93.1**	86.1	92.29
800	82.1	88.5	**93**	84.7	92.55
1000	75.5	89.6	**93.1**	83.2	92.55

Table 10 AWS-MULTI-A performance metrics on the first 100 system calls-(AllMalware)

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
AdWare	95	96	96
Trojan	90	95	92
DangerousObject	0	0	0
Downloader	89	77	82
WebToolbar	97	97	97

detected with a precision that exceeds 90%. In addition, the Downloader class shows a lower precision detection compared to previous classes, with a recall of 77%. However, the DangerousObject class was not detected at all (the metrics are all 0%), and this is due to significant class imbalance.

As depicted in Figure 6, the model shows its performance in classifying the most types of malware, notably Adware and WebToolbartest malware, where most instances are correctly classified, as seen in the diagonal values of the confusion matrix.

6.3 AWS-MULTI-AC

The obtained results are presented in Tables 11, 12 and 13, as well as in Figure 7.

Table 11 represents the accuracy results of the different models trained on the AWSCTD, compared to AWS-MULTI-AC. We can observe that our model achieved the highest accuracy rate with the sequences of the first 10, 60, 400 system calls. Moreover, AWS-MULTI-AC maintains strong performance, with accuracy values close to the highest achieved by CNN. Furthermore, AWS-MULTI-AC performs very well for longer sequence lengths like 600, 800, and 1000 system calls, with accuracy of 92.93%, 93.31% and 93.15%, respectively, while the models LSTM and SVM show a significant drop. For example; in sequence 800, the performance of LSTM is reduced to 41.5%, clearly showing that LSTM cannot handle long sequences.

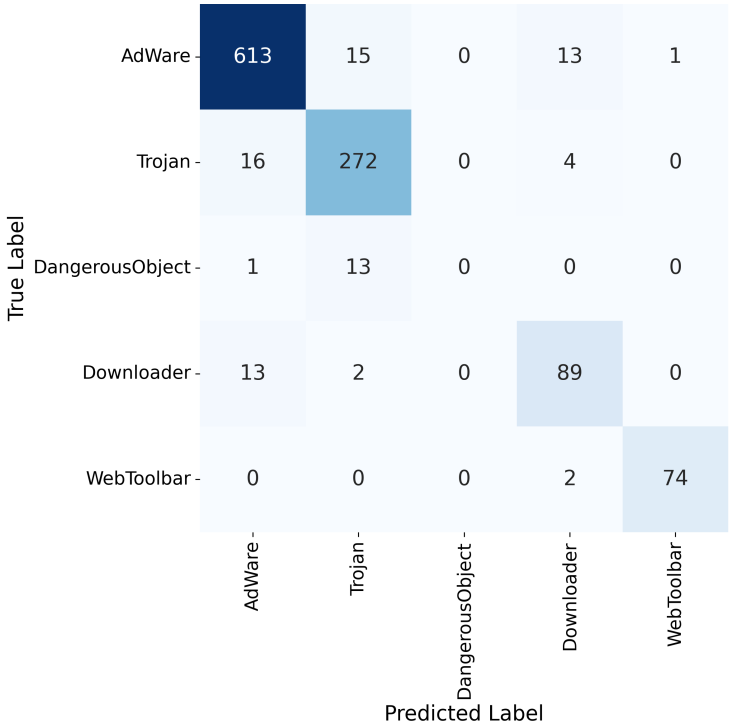


Fig. 6 AWS-MULTI-A Confusion Matrix

In addition, the accuracy rates obtained from the LSTM based approaches presented in Table 12 show that the developed models achieve the highest accuracy compared to AWS-MULTI-AC for short sequences of 10 and 20 system calls. However, from 60 system calls to 1000, AWS-MULTI-AC generally performs well, with a little difference in values. For example, 600, 800, and 1000 syscalls sequences, AWS-MULTI-AC achieves the highest accuracy compared to the models, with an accuracy rates above 92%, where AWS-MULTI-AC highlighting its strength especially on longer sequences. Thus, AWS-MULTI-AC is suitable for capturing complex patterns in both short and long system call sequences.

In Table 13, we present the precision, recall, and f1-score of the first 100 sequence, where we can remark that the model demonstrates high detection of the clean and the different malware classes. WebToolbar class gets the highest precision of 99%, and recall of 95% which results in the highest f1-score of 97%. Also, Adware class can be detected with high precision of 88%, recall of 94% and f1-score of 91%. However, the DangerousObject class cannot be detected, as previously explained in section 6.2, is due to a very low number of samples. The model achieves a strong performance of clean samples, with a precision of 94%.

Figure 7 represents the confusion matrix of AWS-MULTI-AC, where the model proves its feasibility in correctly classifying the most malware types, as well as the clean samples, and this is clearly reflected in the high diagonal value of the matrix.

Table 11 AWS-MULTI-AC Accuracy rates across varying sequence lengths compared to [21, 22]-(AllMalwarePlusClean2)

Counts	LSTM	GRU	CNN	SVM	AWS-MULTI-AC
10	80.1	79.4	80.1	81.2	**81.69**
20	85.8	85.6	**86.9**	86.7	86.54
40	87.3	87.6	**88.8**	88.0	88.46
60	87.5	86.4	88.5	88.2	**88.60**
80	87.5	88.0	**89.2**	87.9	88.43
100	87.7	87.9	**88.8**	87.6	87.79
200	86.2	87.3	**89.6**	86.3	87.95
400	80.3	87.3	89.6	78.1	**89.69**
600	77.6	89.4	**93.5**	74.8	92.93
800	41.5	92.3	**93.6**	73.3	93.31
1000	64.0	84.1	**93.9**	72.3	93.15

Table 12 AWS-MULTI-AC accuracy rates across varying sequence lengths compared to the implemented LSTM-Based approachs -(AllMalwarePlusClean2)

Counts	LSTM	BLSTM	CNN-LSTM	CNN-BLSTM	AWS-MULTI-AC
10	74	**82**	**82**	81	81.69
20	86	**88**	**88**	**88**	86.45
40	89	89	89	**90**	88.46
60	86	85	87	87	**88.60**
80	88	87	87	88	**88.43**
100	87	87	**88**	**88**	87.79
200	87	87	**88**	**88**	87.95
400	88	**90**	89	**90**	89.96
600	89	92	91	92	**92.93**
800	88	90	91	89	**93.31**
1000	86	88	91	89	**93.15**

Table 13 AWS-MULTI-AC performance metrics on the first 100 system calls-(AllMalwarePlusClean2)

Class	Precision (%)	Recall (%)	F1-score (%)
Clean	94	78	85
AdWare	88	94	91
Trojan	81	88	84
DangerousObject	0	0	0
Downloader	83	85	84
WebToolbar	99	**95**	**97**

6.4 NGIDS-BIN Models

As mentioned previously in section 3.3, and in order to circumvent the well-known sequencing problem inherent in the NGIDS dataset [23], we proposed two distinct

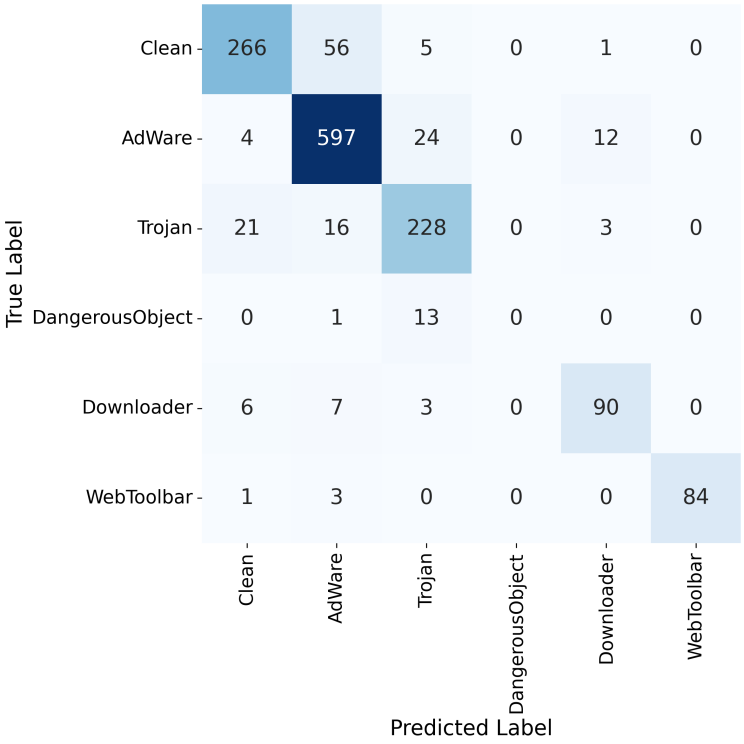


Fig. 7 AWS-MULTI-AC Confusion Matrix

approaches, resulting in the creation of the NGIDS-BIN-PT and NGIDS-BIN-SW models.

The first approach is based on the construction of system call traces representing individual process activity. It begins with a temporal sorting of the records, followed by a grouping (group by) based on the process ID. This operation generates, for each process, a sequence of system calls constituting its trace, accompanied by a set of labels. Labeling is performed via a logical disjunction (OR) operation applied to the process's system call labels: if a single call is malicious, the entire process is considered as such.

Figure 8 shows the confusion matrix for NGIDS-BIN-PT, which shows a strong performance in detecting both malware and clean samples, Where, the model can correctly classify 740 out of 744 from the clean test dataset and correctly classify 719 out of 741 from the malware test dataset. Notice that the false negative is only equal to 4, and the false positive is 22, which is fairly good.

Figure 9 shows the ROC curve of NGIDS-BIN-PT, which confirms that the model performs well when the False Positive Rate (FPR) is low (close to 0). A high True Positive Rate (TPR) close to 1, especially when the FPR remains low, indicates that the model has strong discriminative power.

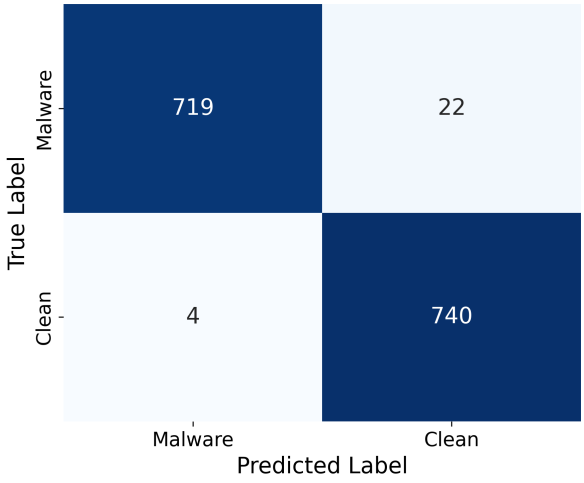


Fig. 8 NGIDS-BIN-PT Confusion Matrix

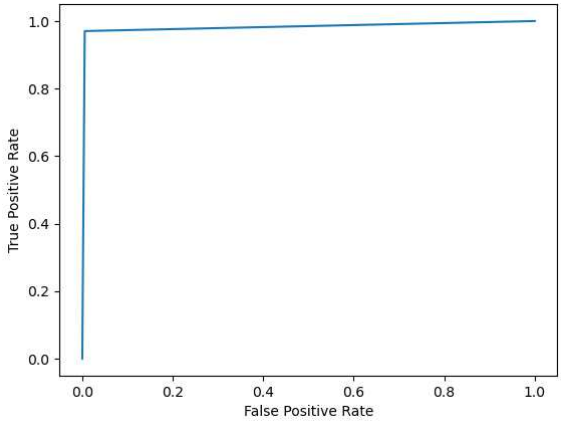


Fig. 9 NGIDS-BIN-PT ROC

The second approach leverages the date and timestamp to organize system calls at the global operating system level. The data is first sorted chronologically and then divided into fixed-size sliding windows. Each window provides a sequence of system calls, which is labeled according to the same logical principle: if the sequence contains at least one call marked as malicious (label 1), then the entire sequence is labeled 1; otherwise, it receives the label 0.

Figure 10 shows the confusion matrix for both the two versions of the sliding window technique. This figure clearly shows that increasing the model window size from 32 to 64 improves the predictive ability of the model and suggests that longer context helps provide more distinctive features for intrusion detection. The obtained

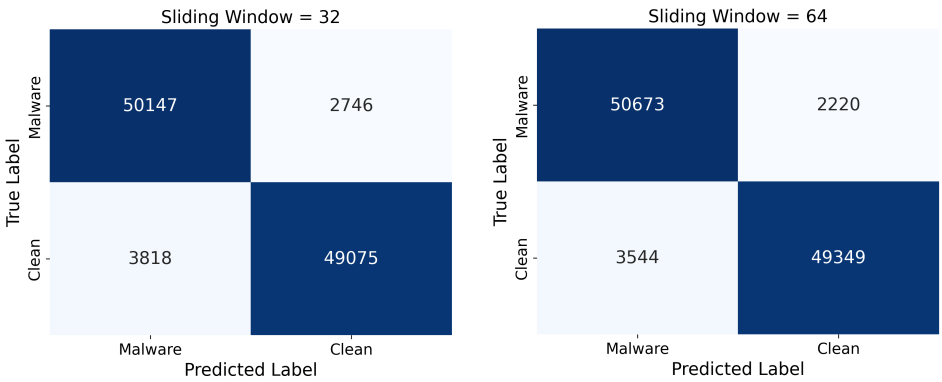


Fig. 10 NGIDS-BIN-SW confusion matrix

Table 14 NGIDS-BIN-PT and NGIDS-BIN-SW Performance metrics

Model	**TPR** (%)	**TNR** (%)	**FPR** (%)	**FNR** (%)	**Acc** (%)	**Pr** (%)	**Rec** (%)	**F1** (%)	**MCC**	**ER** (%)
NGIDS-BIN-PT	97	99	1	3	98	99	97	98	0.97	2
NGIDS-BIN-SW(32)	95	93	7	5	94	93	95	94	0,88	6
NGIDS-BIN-SW(64)	96	93	7	4	95	93	96	95	0,89	5

results for both NGIDS-BIN-PT and NGID-BIN-SW (sliding window size 32 and 64) are depicted in Table 14.

The NGIDS-BIN-PT model, based on the aggregation of system calls by process, displays remarkable performance across all metrics: 97% TPR, 99% TNR, precision of 99%, F1 score 98%, and MCC of 0.97. These results underline the fact that this approach is exceptionally good at capturing hostile activity at the process level, because of the inner temporal coherence of each trace, and the clear meaning of generated sequences. Grouping by processes keeps a high correlation between events, which helps detect intrusions.

However, the NGIDS-BIN-SW(32) and NGIDS-BIN-SW(64) models, which use a sliding window approach for temporal segmentation of system calls, perform slightly worse. The first achieves 95% in TPR, 93% in TNR, and 94% in F1-score, while the second slightly improves these Figures with 96% TPR, 95% F1 score, and 0.89 MCC. It seems that increasing the window size improves the ability to capture wider context, thus improving performance. Also, these two variants suffer from a higher false positive

rate (FPR = 7%) than NGIDS-BIN-PT (FPR = 1%), which is likely explained by the nature of sliding windows, which can contain system calls from heterogeneous contexts, resulting in noisier sequences.

7 Conclusion and Future Work

With the emergence of the Internet and the Internet of Things, everything we use every day is becoming connected to the internet: from mobile phones and wearable devices to household appliances and digital assistants. The more interconnected our IT systems become, the more vulnerable they are to cyberattacks, such as data theft, software corruption, operational disruption, and even physical damage to hardware and network infrastructure. Cybersecurity aims to meet the challenge of understanding these threats and effectively protecting against them. Among the most important are IDS. An intrusion detection system is a device, whether software or hardware, designed to analyze host machine data or network traffic to identify potential suspicious activity or known threats, and to generate an alert if detected. Machine learning and deep learning are among the aspects of artificial intelligence most relevant to cybersecurity. Following the considerable success of Transformers in the development of Large Language Models (LLMs) and NLP in general, as well as in computer vision, researchers have begun to adopt this basic methodology for work in various other fields. In this work, we will leverage this innovative technology, the Transformer architecture, to develop efficient and robust models dedicated to cybersecurity. More specifically, we apply it to the construction of IDS capable of accurately and rapidly identifying and classifying threats, using data from real and varied environments.

In this paper, we propose a complete framework for host-based intrusion detection using Transformers. Based on a template model, we developed five models: AWS-BIN, AWS-MULTI-A, AWS-MULTI-AC, NGIDS-BIN-SW, and NGIDS-BIN-PT. With these models, we attempt to solve binary and multiclass threat classification problems while considering two datasets, AWSCTD and NGIDS, which were used in this research. Through system call traces, each model is designed to capture complex dependencies to facilitate accurate intrusion detection. For a fair comparison, we developed and tested other deep learning models like LSTM, BLSTM, CNN-LSTM, and CNN-BLSTM, all trained under identical experimental conditions. Our Transformer models outperformed the others for all but one instance – CNN-LSTM, which outperformed our models in some cases. This indicates that there is potential in merging CNN with Transformers for superior results.

To begin with, we aim to create lightweight versions of both CNN and Transformer models, providing an optimal balance between resource expenditure and detection performance. Adding convolutional layers to our transformer model for feature extraction for large sequences involves combining CNNs as a feature extractor and transformers using their attention mechanisms to build more precise contexts. This will showcase the promise of using Transformer architectures in the cybersecurity and intrusion detection domains, while setting the stage for hybrid deep learning dominations that seek to capitalize on the strengths of disparate model types.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.
Financial interest The authors have no relevant financial or non-financial interests to disclose.

References

[1] Sy, B.K.: Signature-based approach for intrusion detection. In: Perner, P., Imiya, A. (eds.) Machine Learning and Data Mining in Pattern Recognition, pp. 526–536. Springer, Berlin, Heidelberg (2005)

[2] Senoussi, N.E.H., Salmi, C., Banouh, N., Khalfi, A.: Transformer-based approach for intrusion detection system. In: 2025 International Symposium on iNnovative Informatics of Biskra (ISNIB), pp. 1–6 (2025). https://doi.org/10.1109/ISNIB64820.2025.10983583

[3] Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A., Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019) https://doi.org/10.1109/ACCESS.2019.2895334

[4] Gautam, S., Henry, A., Zuhair, M., Rashid, M., Javed, A.R., Maddikunta, P.K.R.: A composite approach of intrusion detection systems: Hybrid rnn and correlation-based feature optimization. Electronics **11**(21) (2022) https://doi.org/10.3390/electronics11213529

[5] Kim, J., Kim, J., Thu, H.L.T., Kim, H.: Long short-term memory recurrent neural network classifier for intrusion detection. In: 2016 International Conference on Platform Technology and Service (PlatCon), pp. 1–5 (2016). IEEE

[6] Chawla, A., Lee, B.: Host-based intrusion detection system with combined cnn/rnn model. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 149–158 (2018). Springer

[7] Ceponis, D., Goranin, N.: Investigation of dual-flow deep learning models lstm-fcn and gru-fcn efficiency against single-flow cnn models for the host-based intrusion and malware detection task on univariate times series data. Journal of Cybersecurity and Information Management **15**(2), 112–125 (2020)

[8] Wang, Z., Liu, Y., He, D., Chan, S.: Intrusion detection methods based on integrated deep learning model. computers & security **103**, 102177 (2021)

[9] Panigrahi, R., Borah, S., Bhoi, A.K., Ijaz, M.F., Pramanik, M., Kumar, Y., Jhaveri, R.H.: A consolidated decision tree-based intrusion detection system for binary and multiclass imbalanced datasets. Mathematics **9**(7), 751 (2021)

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

[11] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (long and Short Papers), pp. 4171–4186 (2019)

[12] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)

[13] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)

[14] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research **21**(140), 1–67 (2020)

[15] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems **32** (2019)

[16] Čeponis, D., Goranin, N.: Towards a robust method of dataset generation of malicious activity for anomaly-based hids training and presentation of awsctd dataset. Baltic Journal of Modern Computing **6**(3), 217–234 (2018)

[17] Haider, W., Hu, J., Slay, J., Turnbull, B.P., Xie, Y.: Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. Journal of Network and Computer Applications **87**, 185–192 (2017)

[18] Haider, W.: Developing reliable anomaly detection system for critical hosts: a proactive defense paradigm. PhD thesis, UNSW Sydney (2018)

[19] Fawcett, T.: An introduction to roc analysis. Pattern recognition letters **27**(8), 861–874 (2006)

[20] Thakkar, A., Lohiya, R.: A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. Artificial Intelligence Review **55**(1), 453–563 (2022)

[21] Goranin, N., Čeponis, D.: Investigation of awsctdc dataset applicability for malware type classification. Security & Future **2**(2), 83–86 (2018)

[22] Čeponis, D., Goranin, N.: Evaluation of deep learning methods efficiency for

malicious and benign system calls classification on the awsctd. Security and Communication Networks **2019**(1), 2317976 (2019)

[23] Grimmer, M., Kaelble, T., Rahm, E.: Improving host-based intrusion detection using thread information. In: Meng, W., Katsikas, S.K. (eds.) Emerging Information Security and Applications, pp. 159–177. Springer, Cham (2022)