Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105 (Online)
Volume **10**, Number **4**, November **2025**

An Enhanced Deep-Learning Model for Cyberattack Identification

Aidong Xu[1,2], Jinran Du[1,2], Tao Dai[1,2], Peiming Xu[1,2], Zhuowei Wang[3], Jiahui Chen[3*]

[1] Electric Power Research Institute, CSG, Guangzhou 510000, China
[2] Guangdong Provincial Key Laboratory of Power System Network Security
China Southern Power Grid Company Ltd, Guangzhou 510000, China
[3] School of Computer Science and Technology
Guangdong University of Technology, Guangzhou 510006, China
xuad@csg.cn, dujr@csg.cn, daitao@csg.cn, xupm@csg.cn
zwwang@gdut.edu.cn, csjhchen@gmail.com

*Corresponding author: Jiahui Chen
Received June 22, 2024, revised December 22, 2024, accepted May 15, 2025.

Abstract.

The identification of cyberattacks is fundamental for maintaining the security, integrity, and resilience of digital assets and systems in an increasingly interconnected and threat-prone environment. Traditional methods like intrusion detection systems (IDS) usually monitor network traffic or system activities for suspicious patterns or anomalies that may indicate a cyber attack. While traditional methods for identifying cyberattacks have proven effective to a certain extent, they also face several challenges in today's rapidly evolving threat landscape. On one side, cyber attackers are constantly evolving their tactics, techniques, and procedures to evade detection by traditional security measures. A new attack can be designed by mutating slightly existing cyberattacks. Traditional methods fall short in detecting these variant attacks. On the other side, the volume of data generated by modern IT environments, including network traffic, system logs, and endpoint telemetry, can overwhelm traditional security tools and processes. To address these challenges, a novel deep-learning architecture has been developed, leveraging a combination of Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and attention mechanisms. According to the experiment results, the proposed model shows a high accuracy and adaptive ability for detecting and mitigating cyber threats.

Keywords: Cyberattack Identification, Deep Learning, Adaptive

1. **Introduction.** Given the rapid increase in cyberattacks recently, cybersecurity risk has been considered as one of the top global concerns [1]. In today's landscape, cybersecurity holds significant relevance, particularly in the realm of modern information systems like cyber-physical systems (CPS) and the Internet of Things (IoT) [2, 3]. Cyberattacks, such as dictionary attacks, attempt to crack passwords using a dictionary or word list, enabling remote system hijacking [4]. Another prevalent attack that can render unavailable to their intended clients is distributed denial of service (DDoS) [5]. The man-in-the-middle attack exploits communication between two endpoints by eavesdropping and intercepting legitimate traffic [6]. Achieving complete exclusion of security risks is nearly impossible within these systems. Therefore, there's an imperative to consistently identify cyberattacks and anomalies, ensuring ongoing monitoring of security risks.

Many methods for detecting cyberattacks have been present. Traditional methods may be categorized as data-driven or model-based schemes [7]. Data-driven schemes like data mining techniques usually try to discover hidden features or patterns in large datasets [8, 9]. Although these techniques have a powerful ability to mine informative knowledge, it is a huge task to design a good data-driven architecture and its statistical meaning may be unclear. Model-based schemes like Kalman filter [10] often construct predictive models by some mathematical processes. Model-based detection algorithms offer a notable advantage over data-driven ones by not relying on historical datasets. However, a significant drawback of model-based approaches is the necessity for both system parameters and a model, making the method vulnerable to performance compromises due to even minor inaccuracies in these parameters.

In recent years, machine learning techniques have drawn great concern from researchers who study cyberattack detection due to their efficiency, flexibility and automation [11, 12, 13]. Machine learning techniques usually develop typical custom models and can be trained on datasets to do complex tasks. Nevertheless, numerous challenges persist. Cyber attackers continually refine their tactics, techniques, and procedures to evade traditional security measures, often developing new attacks through slight mutations of existing ones. Besides, the sheer volume of data produced in contemporary IT environments, encompassing network traffic, system logs, and endpoint telemetry, has the potential to inundate these techniques. Therefore, it is still difficult to identify cyberattacks timely and with high accuracy. Specifically, our contributions are summarized as follows:

- We propose a novel deep-learning model for identifying cyberattacks. The proposed model integrates CNNs, LSTM networks, and multi-head attention mechanisms. By leveraging these advanced deep-learning techniques, we construct an enhanced model that can capture complex patterns and relationships within cyberattack data more effectively.
- Since most new cyberattacks can be constructed from slight variants of existing attacks, attack strategies and tactics are evolving frequently. The proposed model can be adaptive to evolving attacks through continuous learning and refinement and well utilize the huge volume of data generated by modern IT environments.
- Compared with some well-known deep-learning models that use the same dataset to identify cyberattacks, the proposed model achieves a better performance.

The remaining sections are arranged as follows. In Section 2, we will introduce some related research on cyberattack identification. In Section 3, we will explain some preliminaries for constructing the proposed model. In Section 4, we will describe the proposed model in detail. In Section 5, we will give an evaluation of the proposed model. Finally, in Section 6, we will make a summary of the proposed model.

2. **Related work.**

Early in 1987, a real-time intrusion detection system (IDS) was proposed to detect various security violations, from external break-in attempts to internal abuses [14]. IDS is driven by the recognition that existing systems often harbor flaws making them vulnerable to intrusions, while replacing them with more secure alternatives is challenging. The IDS model operates under the premise that abnormal patterns in system usage, gleaned from audit records, can reveal security breaches. It encompasses profiles that encapsulate subject-object interactions through metrics and statistical models, alongside rules for extracting insights from audit data and identifying deviations from typical behaviour. Inspired by this idea, numerous statistical techniques have emerged, particularly within the context of time series analytics [15, 16, 17]. These methods often focus on characterizing the statistical properties of attacks from the captured data.

In recent years, the Internet of Things (IoT) paradigm has seen widespread adoption across various industries, such as the medical sector, vehicle manufacturing, home appliances, and more [18]. Modern systems like this often have some key attributes: One of them is that the nodes in these systems are always active while performing the collection, processing, and transmission of data. Furthermore, the majority of the nodes in these systems possess the capability to gather, analyze, and send data, rendering them vulnerable to certain privacy and security risks. These vulnerabilities could compromise both the integrity of these systems and the associated applications. These systems are vulnerable to a range of attacks and intrusions across multiple layers.

Subsequently, due to its remarkable efficacy across diverse application domains, machine learning has become increasingly integrated into numerous intrusion detection systems. In 2016, Imamverdiyev et al. applied machine learning techniques to analyze network traffic parameters for indications of potential attacks [19]. This paper explores the utilization of the extreme learning machine method for detecting intrusions within network traffic. Later, Chowdhury et al. [20] devised a few-shot deep learning method to enhance intrusion detection. Firstly, they trained a deep convolutional neural network (CNN) for this purpose and subsequently extracted outputs from various layers within the CNN. Finally, they utilized a linear support vector machine (SVM) and a 1-nearest neighbor (1-NN) classifier for few-shot intrusion detection. In [21], Gao et al. developed a MultiTree algorithm which tried to find new cyberattacks in time. They evaluated recent advancements and current challenges in intrusion detection technology and proposed a MultiTree and adaptive voting algorithm which notably enhance the effectiveness of intrusion detection.

3. **Preliminaries.**

We integrate several well-known deep-learning networks to construct the proposed model. In this section, we will explain the preliminaries of these networks.

3.1. **CNNs.**

A Convolutional Neural Network [22], commonly referred to as CNN or ConvNet, is a neural network architecture designed specifically for analyzing data with a grid-like structure, such as images. A digital image represents visual data in binary form, comprising pixels arranged in a grid pattern. As is shown in Figure 1, a CNN usually consists of three layers: a convolutional layer, a pooling layer, and a fully connected layer.

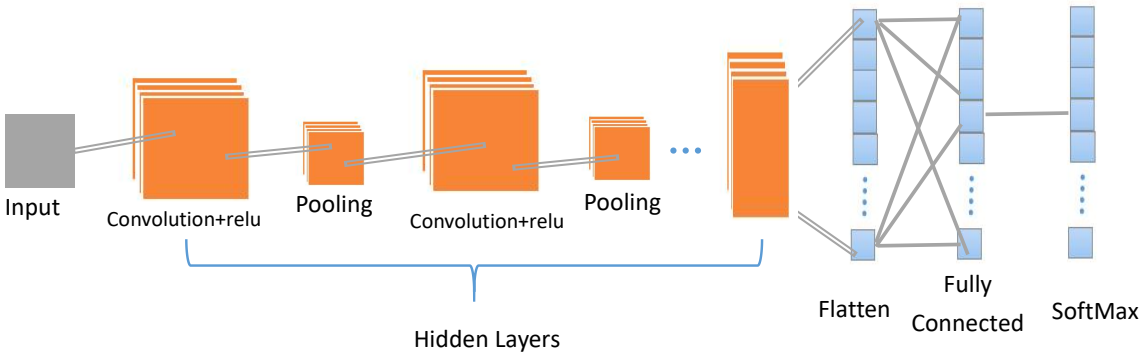

FIGURE 1. Architecture of a CNN

3.1.1. *Convolutional Layer.* The convolution layer serves as the core element of CNNs, carrying the primary computational burden within the network. It conducts a dot product operation between two matrices: the kernel, representing the set of learnable parameters, and the restricted region of the input data. As the kernel traverses the height and width of the image during the forward pass, it generates a two-dimensional activation map

depicting the response of the kernel at each spatial position. The sliding size of the kernel, known as the stride, determines the output volume size. If we have an input of size $W \times W \times D$ and D_{out} number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

This will yield an output volume of size $W_{out} \times W_{out} \times D_{out}$.

3.1.2. *Pooling Layer.* The pooling layer replaces the output of the network at specific locations by computing a summary statistic from nearby outputs, effectively reducing the spatial dimension of the representation and thus lessening computational demands and weights. This pooling operation is conducted independently on each slice of the representation. If we have an activation map of size $W \times W \times D$, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F}{S} + 1$$

This will yield an output volume of size $W_{out} \times W_{out} \times D$.

3.1.3. *Fully Connected Layer.* The neurons within this layer establish full connectivity with all neurons in both the preceding and succeeding layers. Consequently, computations in this layer can proceed conventionally through matrix multiplication followed by the addition of biases. The FC layer facilitates the mapping of representations between the input and output layers.

3.2. **LSTM.** Long Short-Term Memory (LSTM) [23] represents an enhanced iteration of recurrent neural networks, particularly adept at handling sequence prediction assignments. LSTM has a chain of repeating units like recurrent neural networks.

4. The Proposed Model.

4.1. **Overview.** As is shown in 2, the framework of the proposed model integrates CNN, LSTM, and multi-head attention network to build an enhanced deep-learning model. Firstly, the raw dataset is preprocessed to become trainable in the next step. Then, the preprocessed data will be taken as input for the three components (CNN, LSTM, Multi-head Attention) separately. The output of the three networks will be concatenated together and go through a flattened layer. Finally, with a fully connected layer (Softmax), we get the output of the whole model.

4.2. Data Preprocessing.

4.2.1. *The Raw Dataset.* We use the NSL-KDD dataset [24] which is a benchmark dataset for comparing different cyberattack identification methods for researchers. It is an improved version of the KDDCUP'99 [25] dataset which has been widely used for evaluating cyberattack detection systems. The NSL-KDD dataset has eliminated redundant records from the original KDDCUP'99 dataset to avoid biased classification to more frequent records. As is shown in Table 1, the whole dataset contains $148,517$ records and 48.12% of them are cyberattack records. Each record has 43 columns in which 42 are taken as features and 1 column indicates what kind of records it is (normal or the specific attack type). Note that the types of cyberattacks in the training set and testing set are intersected. It means some types of cyberattacks in the testing set are not in the training

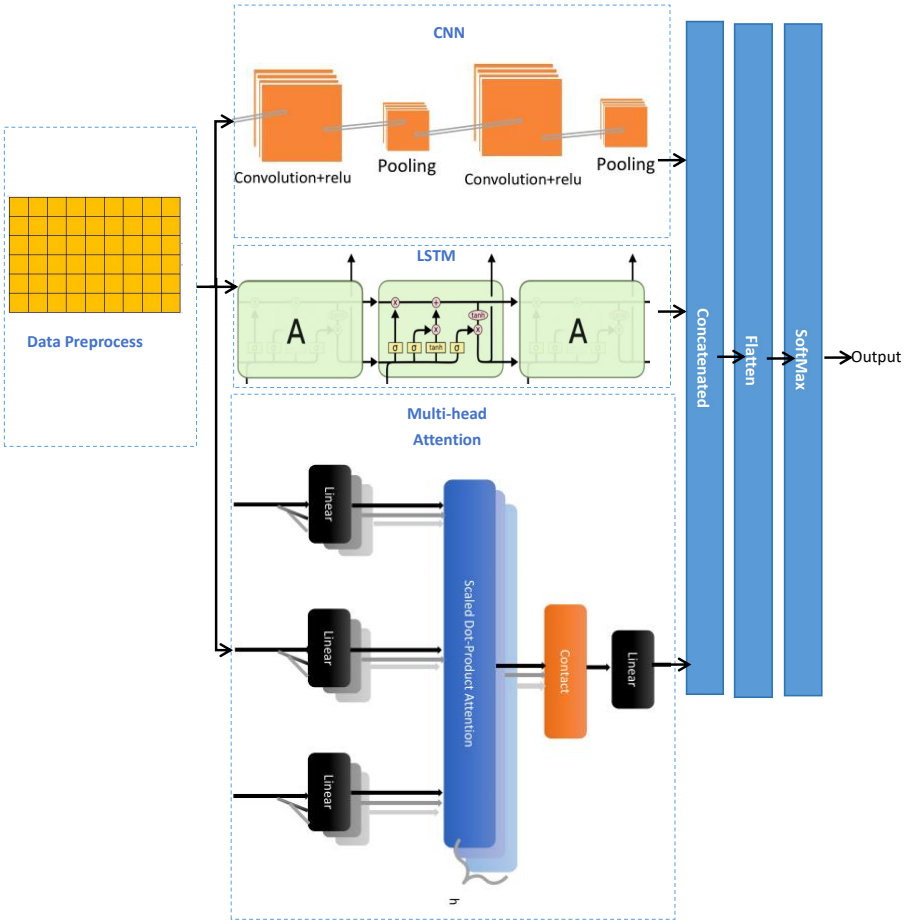

FIGURE 2. Framework of the proposed model

TABLE 1. The Distribution of the Data

Set	Normal	Attack	Numebr of Attack Types
Training	67,341	58,632	22
Testing	9,712	12,832	37
Total	77,053	71,464	39

set. Therefore, it can evaluate the proposed model for its effectiveness in identifying novel attacks.

4.2.2. *Preprocessing.* Since machine learning models do not work with the raw data directly, the raw dataset should be preprocessed before feeding into the deep-learning network for training. The process is shown in Algorithm 1. As is shown in Figure 3, the features of the raw dataset can be divided into categorical features, numerical features and target values. The categorical data are encoded using a technique called one-hot encoding in Line 3 in Algorithm 1. By one-hot encoding, we represent categorical variables as numerical values in a machine-learning model. The categorical parameters will prepare separate columns for all the categorical labels. So, wherever there is a specific categorical label, the value will be 1 in the existing column and 0 in the missing column. The raw numerical data contains features of various dimensions and scales altogether. Different scales of the data features adversely affect the modelling of a dataset. It leads to a biased outcome of predictions in terms of misclassification error and accuracy rates. Thus, we will

use a Python library to standardize the numerical data. The target data which indicates the type of each record is also encoded using a technique called Label Encoding. Suppose the target column has elements as Normal, AttackType1, and AttackType2. After applying label encoding, the target column is converted into a numerical column having elements 0,1, and 2 where 0 is the label for Normal, 1 is the label for AttackType1, and 2 is the label for AttackType2. Finally, all the processed data are concatenated together for training in the next step.

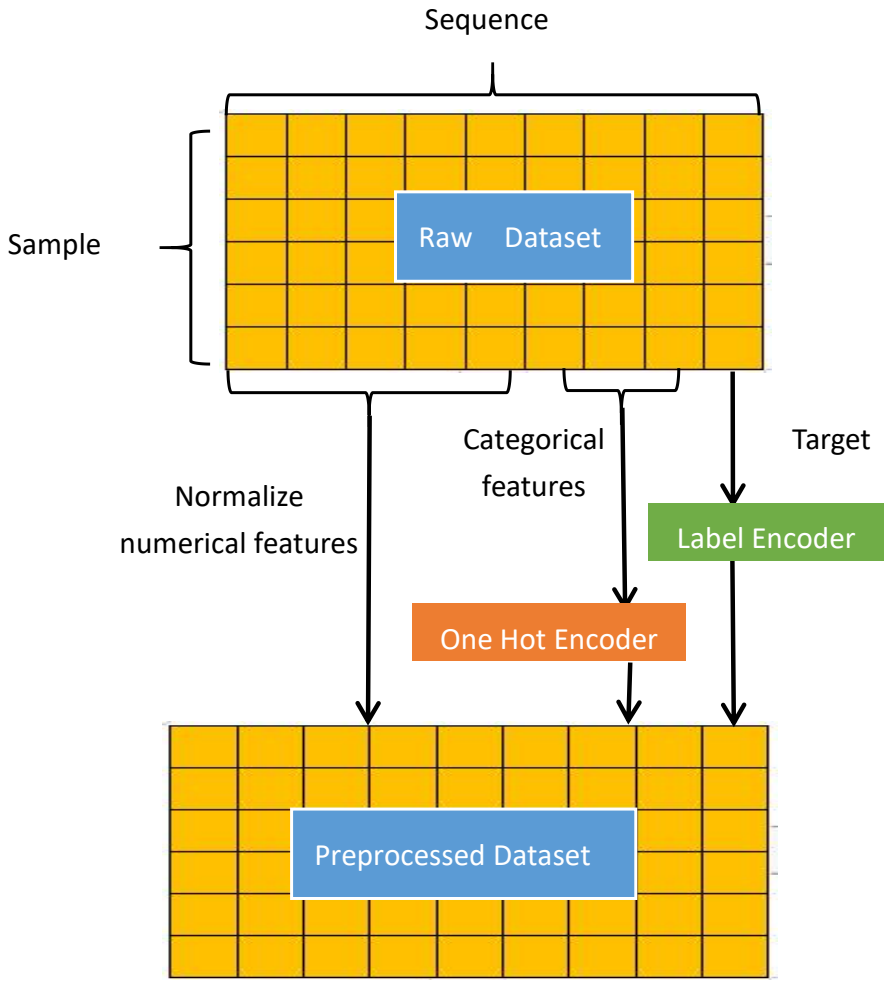

FIGURE 3. Data Preprocessing

Algorithm 1 Data Preprocessing Algorithm

```
data ← read("RawDataSet")
cateData ← getCategoricalData(data)
cateData ← OneHotEncoder(cateData)
numData ← getNumericalData(data)
numData ← StandardScaler(numData)
X ← concatenate(cateData, snumData)
Y ← getTargetColumn(data)
Y ← LabelEncoder(Y)
return (X,Y)
```

4.3. **CNN Component.** The CNN part of the model is responsible for extracting local patterns within the input data. The input is first passed through a convolutional layer with some filters and kernels, applying the ReLU activation function. This is represented by the formula:

$$\text{output} = \text{Pooling}(\text{Conv}(a = relu)(\text{Pooling}(\text{Conv}(a = relu))))$$

After the convolutional operations, a max pooling layer reduces the dimensionality of the data, followed by another convolutional layer with n filters. Finally, a global max pooling layer is applied to reduce the output to a single vector that captures the most significant features from the CNN layers.

4.4. **LSTM Component.** The LSTM part is designed to capture long-term dependencies in sequential data.

As is shown in Figure 4, the LSTM Component of the repeating unit contains four neural networks and different memory blocks called cells. Information is retained by the cells, and memory manipulations are performed by the following gates.

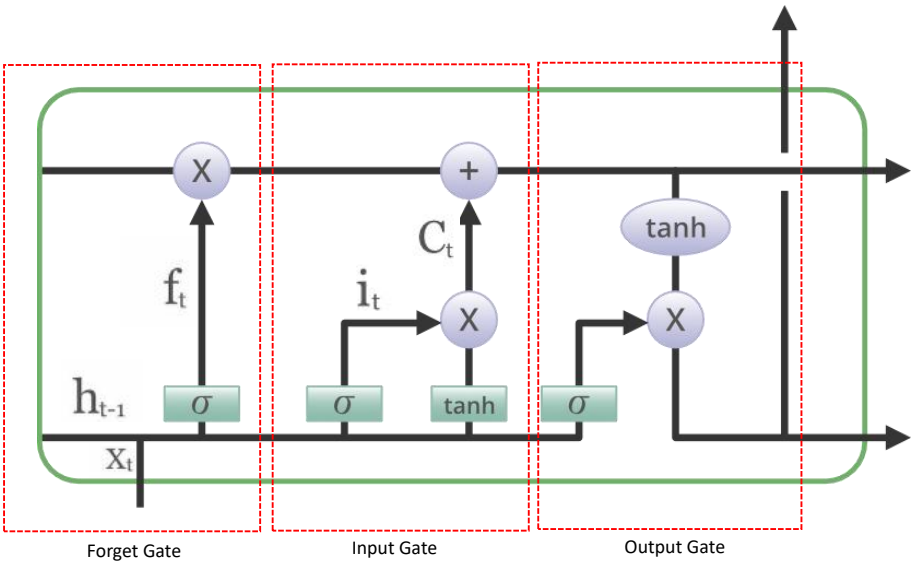

FIGURE 4. Workflow of our LSTM Component

The forget gate serves to eliminate irrelevant information from the cell state. It takes two inputs, X_t (the input at the current time step) and h_{t-1} (the previous cell output), which are then subjected to multiplication by weight matrices and the addition of biases. The resulting values undergo an activation function, yielding a binary output. If the output is 0 for a given cell state, the corresponding information is discarded, while an output of 1 signifies retention for future utilization. The equation governing the forget gate is as follows:

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f)$$

where W_f indicates the weight matrix, b_f is the bias with the forget gate and σ is the activation function.

The input gate facilitates the integration of useful information into the cell state. Initially, information undergoes modulation via the sigmoid function, akin to the forget gate, utilizing inputs h_{t-1} and X_t to filter the values to be retained. Subsequently, a vector is generated using the $tanh$ function, producing an output ranging from -1 to $+1$, encompassing all potential values from h_{t-1} and X_t. Finally, the values of this vector and the modulated values are multiplied to extract the valuable information. We disregard

the information previously selected for exclusion by multiplying the previous state by f_t. Then, we incorporate $i_t * C_t$, which denotes the revised candidate values, adjusted based on the degree to which each state value was chosen for updating. The equation governing the input gate is outlined as follows:

$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i)$$

$$\hat{C}_t = tanh(W_c[h_{t-1}, X_t] + b_c)$$

$$C_t = f_t C_{t-1} + i_t \hat{C}_t$$

The output gate is responsible for extracting valuable information from the current cell state. Initially, a vector is created by subjecting the cell to the *tanh* function. Subsequently, the information undergoes modulation via the sigmoid function with inputs h_{t-1} and X_t. Finally, the values of this vector and the modulated values are multiplied to serve as both output and input for the next cell. The equation governing the output gate is detailed as follows:

$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o)$$

It processes the input sequence one element at a time and maintains a hidden state that effectively carries information across the sequence. The LSTM output is given by:

$$\text{output} = \text{LSTM}(n)$$

This component can be particularly useful for tasks like time-series forecasting or language modeling, where the context provided by previous elements is crucial for understanding the current element.

4.5. **Attention Component.** As is shown in Figure 5, the module splits its Query, Key, and Value parameters N-ways and passes each split independently through a separate head. All of these similar calculations are then combined together to produce a final Attention score. The equation governing the module is detailed as follows:

$$MultiHead(Q, K, V) = [head_1, ..., head_h]W_0$$

$$head_i = Attention(QW_i{}^Q, KW_i{}^K, VW_i{}^V)$$

where W are all learnable parameter matrices.

The attention mechanism allows the model to focus on different parts of the input sequence when making predictions, similar to how humans pay attention to specific parts of a scene or conversation. The multi-head attention provides multiple "views" of the input, enabling the model to capture various aspects of the data. The attention output is described as:

$$\text{output} = \text{MultiHeadAttention}(\text{num\_heads} = n, \text{key\_dim} = m)$$

This component is particularly useful for tasks that require understanding complex relationships within the data, such as machine translation or document summarization.

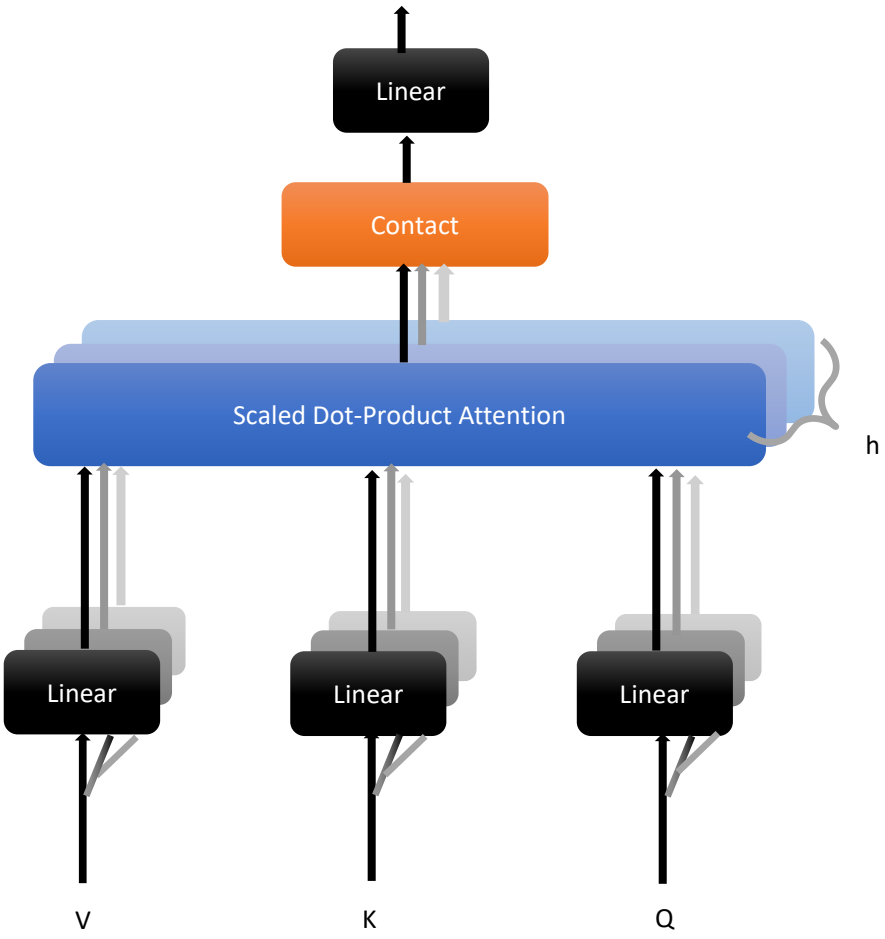

FIGURE 5. Workflow of the multi-head attention component

4.6. **Dta Outputs.** The outputs of the CNN, LSTM, and attention components are concatenated into a single vector. This combined vector is then flattened and passed through a dense layer with a softmax activation function to produce the final output, which represents the probability distribution over the target classes. The output formula is:

$$\text{Output} = \text{SoftMax}(\text{Concatenate}([\text{CNN}, \text{LSTM}, \text{Attention}]))$$

4.7. **The Final Algorithm.** As is shown in Algorithm 2, we create a deep learning model that combines convolutional neural networks (CNN), long short-term memory networks (LSTM), and multi-head attention mechanisms. This model is designed to process sequential data, such as time series or text, by leveraging the strengths of each component to capture different types of patterns and dependencies in the data.

In summary, the algorithm constructs a model that integrates different neural network architectures to handle complex patterns in sequential data, making it robust for a variety of tasks that require nuanced understanding and processing of sequences. The model is finalized by defining the input and output layers, and then it's ready to be compiled and trained on a specific task.

5. Experiment Results.

5.1. **Experimental Environment.** The proposed model was evaluated on a public computing platform which may be running many other models at the same time. The platform has 32 CPUs of Intel(R) Xeon(R) Gold 6130 at 2.10GHz, 2 NVIDIA Tesla P40, and about

Algorithm 2 Create the Proposed Model

Require: *input_shape*, *num_classes*
inputs ← Input(shape=*input_shape*)
cnn ← Conv(inputs)
cnn ← MaxPooling(cnn)
cnn ← Conv(cnn)
cnn ← GlobalMaxPooling(cnn)
lstm ← LSTM(inputs)
attention ← MultiHeadAttention(inputs, inputs)
attention ← Reshape((-1,))(attention)
concatenated ← Concatenate([cnn, lstm, attention])
flatten ← Flatten(concatenated)
outputs ← SoftMax(*num_classes*)(concatenated)
model ← Model(inputs=inputs, outputs=outputs)
return model

81.8*G* free memory. The OS version is Linux 3.10.0-1160.el7.x86_64. The proposed model is implemented by Python and several libraries such as sklearn, keras, tensorflow are employed.

5.2. **Evaluation Of Different Hyperparameters.** To evaluate the performance of the proposed model, we test the model on different hyperparameters. These hyperparameters include the number of epochs, batch size, learning rate, the number of LSTM units, the number of CNN filters, and the number of multi-head attention heads. When we evaluate the proposed model, we will first choose and tune one of these hyperparameters for observation and keep the values of the other hyperparameters invariant at the same time.

Firstly, we evaluate different numbers of epochs. Here, the number of epochs is a hyperparameter that defines how many times the learning algorithm will work through the entire training dataset. As is shown in Figure 6, there are two peaks of the training accuracy and testing accuracy curve. Since the testing accuracy is a more valuable indicator than the training accuracy when evaluating the model, 40 is a recommended value for this hyperparameter. Batch size refers to the number of training samples used in one iteration. As is shown in Figure 7, the value of this hyperparameter with which the model reaches the best performance is about 64. Learning rate is a crucial hyperparameter that influences the size of the steps taken during the optimization process. As is shown in Figure 8, a reasonable learning rate is around 0.001.

Figures 9, 10 and 11 show the evaluation of the hyperparameters of the integrated networks in the proposed model. Each LSTM unit can be thought of as a memory cell capable of remembering information for a certain amount of time during the training process. The number of units in an LSTM layer represents the size of the hidden state, which is the output dimensionality of the layer. In Convolutional Neural Networks (CNNs), Each filter is applied to the input data to produce a feature map, which represents certain features detected in the input. More filters can allow a network to capture more detailed features but also increase computational complexity and the risk of overfitting. The number of multi-head attention heads defines how many sets of attention mechanisms are used in parallel. According to the experiment results, we suggest 16, 16, 2 for the number of LSTM units, the number of CNN filters and the number of attention heads separately.

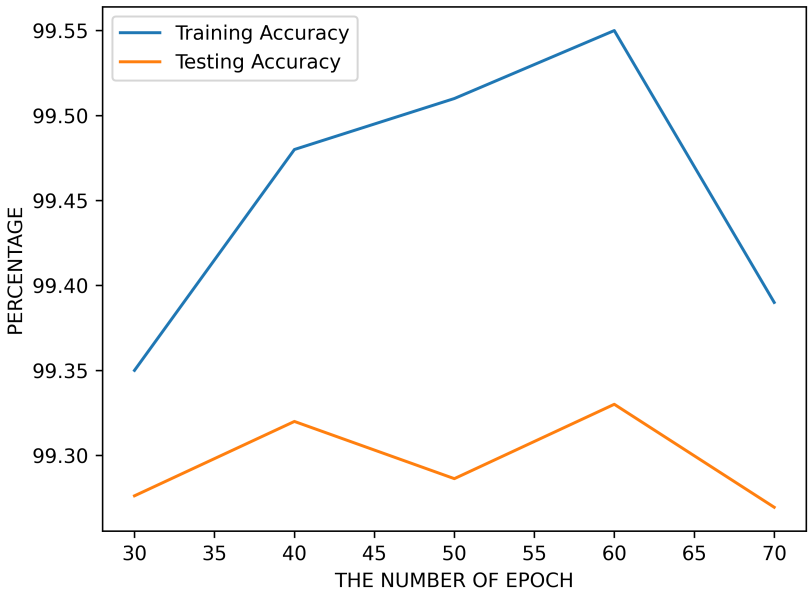

FIGURE 6. Evaluation for different numbers of epochs

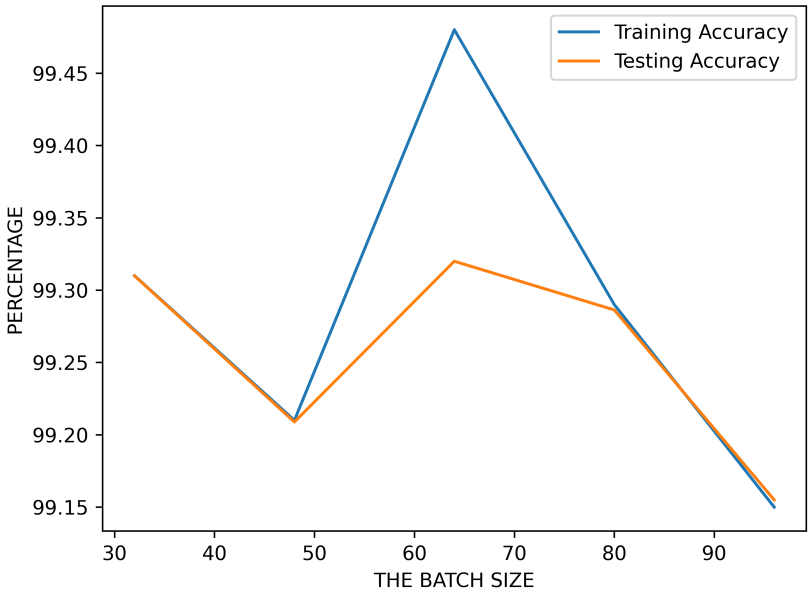

FIGURE 7. Evaluation for different batch size

5.3. **Comparison.** To give a comparison with the existing deep-learning model, we chose two well-known methods (LSTM and CNN) which run on the same raw dataset as the proposed model. In the experiment, we not only predict whether a cyber behavior is an attack but also identify what kind of attack is for an attack behavior. Thus, this is a multiclass classification problem and we compare the prediction methods using the following metrics:

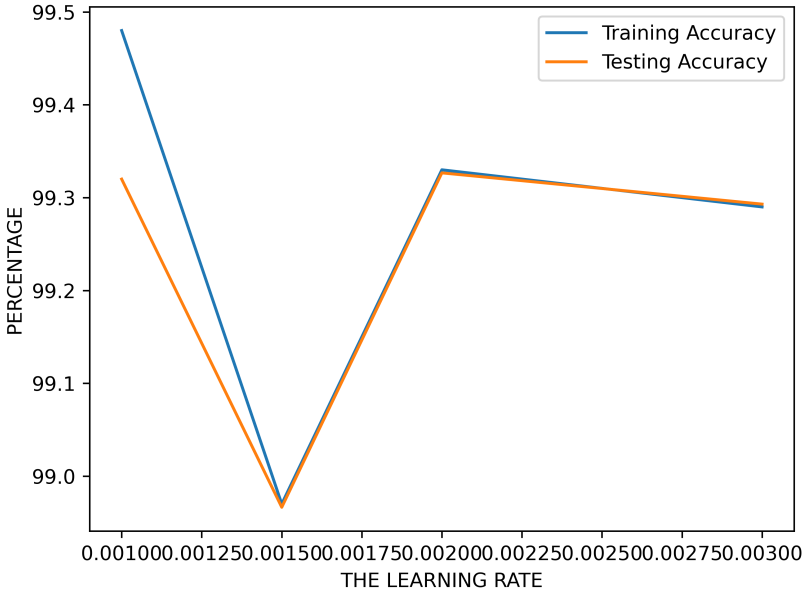

FIGURE 8. Evaluation for different learning rate

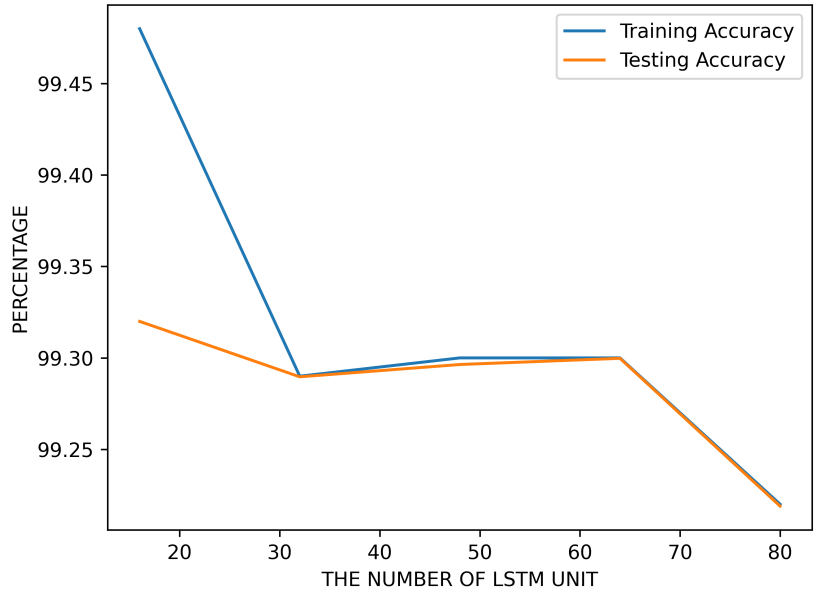

FIGURE 9. Evaluation for different numbers of LSTM units

Accuracy: Accuracy is the most common metric to be used in everyday talk. Accuracy shows how many predictions out of all we made were true. The formula is as follows:

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

Where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

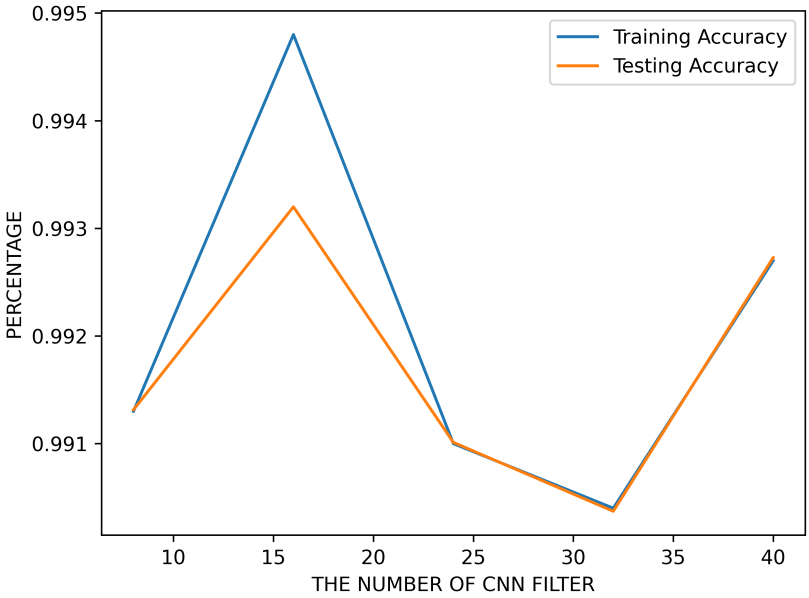

FIGURE 10. Evaluation for different numbers of CNN filters

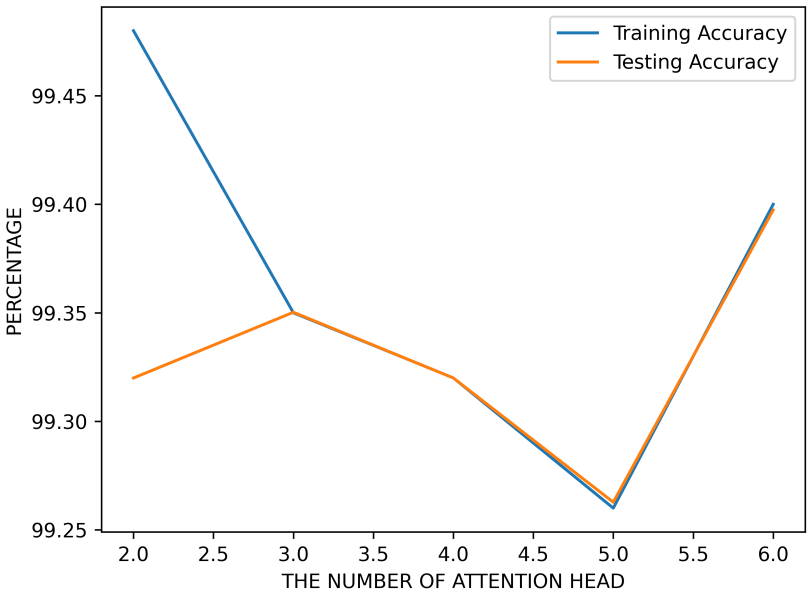

FIGURE 11. Evaluation for different numbers of attention heads

Precision: For a specific class, it is the number of true positives (the number of items correctly labeled as belonging to the positive in that class) divided by the total number of elements labeled as belonging to the positive class (true positives + false positives). The formula is as follows:

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

Recall: For a specific class, it is the number of true positives divided by the total number of elements that actually belong to the positive class (true positives + false

TABLE 2. COMPARISON WITH SIZE (133666, 64292)

Models	Accuracy(%)	Precision(%)	Recall(%)	F1 Score(%)
Our Method	**99.49**	**85.01**	**79.96**	**82.41**
CNN	98.90	77.09	63.76	69.80
LSTM	98.12	50.11	48.63	49.36

TABLE 3. COMPARISON WITH SIZE (103962, 50043)

Models	Accuracy(%)	Precision(%)	Recall(%)	F1 Score(%)
Our Method	**99.47**	**81.00**	**77.78**	**79.36**
CNN	98.85	75.22	62.82	68.47
LSTM	98.08	50.00	48.42	49.20

TABLE 4. COMPARISON WITH SIZE (103962, 50043)

Models	Accuracy(%)	Precision(%)	Recall(%)	F1 Score(%)
Our Method	**99.45**	**81.74**	**77.04**	**79.32**
CNN	98.85	74.22	62.59	67.91
LSTM	98.04	52.66	51.03	51.83

TABLE 5. COMPARISON WITH SIZE (44556, 21454)

Models	Accuracy(%)	Precision(%)	Recall(%)	F1 Score(%)
The proposed model	**99.40**	**76.76**	**72.51**	**74.57**
CNN	98.80	68.44	61.75	64.92
LSTM	97.98	53.99	50.72	52.30

negatives). The formula is as follows:

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

F1 Score: It is a measure that combines recall and precision. As we have seen there is a trade-off between precision and recall, F1 can therefore be used to measure how effectively our models make that trade-off. The formula is as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

In multiclass classification, we first calculate the metrics for each class individually and then take the average. This method treats all classes equally, regardless of their size. To make a comprehensive comparison, we chose 4 different sizes of test data to conduct the evaluation for these models. We sampled 90%, 70%, 50%, and 30% data from the whole dataset separately for evaluation. The total size of the data and the count of true attacks in each test dataset are (133666, 64292), (103962, 50043), (74259, 35711), and (44556, 21454). As is shown in Table 2 to Table 5, the proposed model enjoys a higher performance on all the testing datasets than the other two models.

6. **Conclusion.** The emergence of this enhanced deep-learning architecture, seamlessly integrating CNNs, LSTM networks, and attention mechanisms, marks a profound leap forward in the domain of cyberattack identification. By leveraging the capabilities of these advanced techniques, the architecture offers a robust and adaptive solution for detecting and mitigating cyber threats in real-time. From the experiment results, we can see that the

proposed model outperforms similar methods when using the same dataset. As cybersecurity continues to be a critical concern in the digital age, the adoption of such innovative approaches holds the potential to strengthen defenses, safeguard sensitive information, and preserve the integrity of digital infrastructures. While the presented deep-learning architecture marks a significant advancement in cyberattack identification, the journey toward robust and comprehensive cybersecurity solutions is an ongoing endeavor. As we continue to advance in the field of cyberattack identification, future research should focus on enhancing deep-learning architectures, exploring data augmentation techniques, integrating adversarial defense mechanisms, improving model explainability, enabling real-time deployment, facilitating continuous learning, and fostering interdisciplinary collaboration. By addressing these key areas, we can further strengthen our cybersecurity defenses and create more robust and adaptive solutions to combat evolving cyber threats.

Acknowledgment. This research was supported by the Guangdong Provincial Key Laboratory of Power System Network Security (No.GPKLPSNS-2022-KF-01).

REFERENCES

[1] C. Florackis, C. Louca, R. Michaely, and M. Weber, "Cybersecurity risk," *The Review of Financial Studies*, vol. 36, no. 1, pp. 351–407, 2023.

[2] O. Tushkanova, D. Levshun, A. Branitskiy, E. Fedorchenko, E. Novikova, and I. Kotenko, "Detection of cyberattacks and anomalies in cyber-physical systems: Approaches, data sources, evaluation," *Algorithms*, vol. 16, no. 2, p. 85, 2023.

[3] R. Alanazi and A. Aljuhani, "Anomaly detection for industrial internet of things cyberattacks." *Computer Systems Science & Engineering*, vol. 44, no. 3, 2023.

[4] A. Ustundag, E. Cevikcan, B. C. Ervural, and B. Ervural, "Overview of cyber security in the industry 4.0 era," *Industry 4.0: managing the digital transformation*, pp. 267–284, 2018.

[5] M. M. Salim, S. Rathore, and J. H. Park, "Distributed denial of service attacks and its defenses in iot: a survey," *The Journal of Supercomputing*, vol. 76, pp. 5320–5363, 2020.

[6] N. Sivasankari and S. Kamalakkannan, "Detection and prevention of man-in-the-middle attack in iot network using regression modeling," *Advances in Engineering software*, vol. 169, p. 103126, 2022.

[7] N. Mtukushe, A. K. Onaolapo, A. Aluko, and D. G. Dorrell, "Review of cyberattack implementation, detection, and mitigation methods in cyber-physical systems," *Energies*, vol. 16, no. 13, p. 5206, 2023.

[8] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, "Data mining techniques in intrusion detection systems: A systematic literature review," *IEEE Access*, vol. 6, pp. 56 046–56 058, 2018.

[9] J. R. Saura, D. Palacios-Marqués, and D. Ribeiro-Soriano, "Using data mining techniques to explore security issues in smart living environments in twitter," *Computer Communications*, vol. 179, pp. 285–295, 2021.

[10] L. Daria, Z. Dmitry, and Y. Anastasiia, "Predicting cyber attacks on industrial systems using the kalman filter," in *2019 Third World Conference on Smart Trends in Systems Security and Sustainablity (WorldS4)*. IEEE, 2019, pp. 317–321.

[11] M. Bagaa, T. Taleb, J. B. Bernabe, and A. Skarmeta, "A machine learning security framework for iot systems," *IEEE Access*, vol. 8, pp. 114 066–114 077, 2020.

[12] R. V. Mendonça, A. A. Teodoro, R. L. Rosa, M. Saadi, D. C. Melgarejo, P. H. Nardelli, and D. Z. Rodríguez, "Intrusion detection system based on fast hierarchical deep convolutional neural network," *IEEE Access*, vol. 9, pp. 61 024–61 034, 2021.

[13] I. Ullah and Q. H. Mahmoud, "Design and development of a deep learning-based model for anomaly detection in iot networks," *IEEE Access*, vol. 9, pp. 103 906–103 926, 2021.

[14] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[15] Z. Zhan, M. Xu, and S. Xu, "Characterizing honeypot-captured cyber attacks: Statistical framework and case study," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1775–1789, 2013.

[16] Z. Zhenxin, M. Xu, and S. Xu, "Predicting cyber attack rates with extreme values," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1666–1677, 2015.

[17] J. D. Cryer, K.-S. Chan, and K.-S. Chan, *Time series analysis: with applications in R*. Springer, 2008, vol. 2.

[18] S. M. Kasongo, "An advanced intrusion detection system for iiot based on ga and tree based algorithms," *IEEE Access*, vol. 9, pp. 113 199–113 212, 2021.

[19] Y. Imamverdiyev and L. Sukhostat, "Anomaly detection in network traffic using extreme learning machine," in *2016 IEEE 10th international conference on application of information and communication technologies (AICT)*. IEEE, 2016, pp. 1–4.

[20] M. M. U. Chowdhury, F. Hammond, G. Konowicz, C. Xin, H. Wu, and J. Li, "A few-shot deep learning approach for improved intrusion detection," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. IEEE, 2017, pp. 456–462.

[21] X. Gao, C. Shan, C. Hu, Z. Niu, and Z. Liu, "An adaptive ensemble machine learning model for intrusion detection," *Ieee Access*, vol. 7, pp. 82 512–82 521, 2019.

[22] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[23] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[24] M. Hassan, "Nsl-kdd dataset," https://www.kaggle.com/datasets/hassan06/nslkdd, 2020, accessed: 2024-04-28.

[25] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, 2009, pp. 1–6.