Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

Anomaly Detection in IoT Traffic in the Presence of Gaussian Noise Using Deep Neural Networks

Roya Morshedi and S.Mojtaba Matinkhah*

Department of Computer Engineering, *Yazd University, Yazd, Iran.*

Article Info

Article History:
Received 27 January 2025
Revised 28 February 2025
Accepted 15 April 2025

DOI:10.22044/jadm.2025.15672.2684

Keywords:
Anomaly Detection, Gaussian Noise, Hurst Parameter, Intrusion Detection System, LSTM Network.

**Corresponding author: matinkhah@yazd.ac.ir (S. M. Matinkhah).*

Abstract

The Internet of Things (IoT), is one of the fastest-growing domains and plays a pivotal role in modern smart services. However, resource limitations in IoT nodes pose significant security challenges, rendering them vulnerable to cyberattacks. In recent years, the use of deep learning models has emerged as an effective approach for detecting anomalies in IoT traffic. One of the main challenges of these approaches is the detrimental impact of Gaussian noise on the models' detection accuracy. In this study, a deep learning-based intrusion detection system is proposed, leveraging a simple and optimized LSTM architecture with 128 memory units. This model is trained on the CIC-IDS2017 dataset and designed for deployment on edge servers. The performance evaluation results demonstrate the system's high capability in detecting diverse attacks such as DoS, DDoS, and other advanced threats. Key features of this system include a detection rate of 99.90%, an overall accuracy of 99.90%, and an F1 score of 98.93%. One innovation of this research is the examination of the impact of integrating the Hurst parameter with deep learning models. The results indicate that this integration enhances the model's resilience against Gaussian noise and improves threat detection performance in IoT traffic. The findings of this research emphasize the importance of utilizing advanced statistical features and designing noise-resistant models for the cybersecurity of IoT networks. The proposed system, with its precise performance, rapid response time, and unique defensive strategy, represents a significant step toward improving IoT network security and protecting smart infrastructure. This research can be an efficient solution for developing security systems to counter complex cyber threats.

1. Introduction

The Internet of Things (IoT), as one of the most innovative and rapidly growing fields, has had a profound impact on industry and individuals' daily lives. From connected devices in smart homes to complex systems in smart cities, IoT has brought about a transformative change in every aspect of human life. However, these intelligent networks face numerous security threats due to the resource limitations of their nodes and the complexity of communications between them. Cyberattacks, which can compromise IoT data and systems, require advanced security solutions to detect threats in real-time. One of the greatest challenges in this field is combating Gaussian noise and its adverse impact on the accuracy of anomaly detection models. The Internet of Things (IoT), as one of the most innovative and rapidly growing fields, has had a profound impact on industry and individuals' daily lives. From connected devices in smart homes to complex systems in smart cities, IoT has brought about a transformative change in every aspect of human life. However, these intelligent networks face numerous security threats due to the resource limitations of their nodes and the complexity of communications between them. Cyberattacks, which can compromise IoT data and systems, require advanced security solutions to detect threats in real time. One of the greatest challenges in this field is combating Gaussian noise and its adverse impact on the accuracy of anomaly detection models. This paper introduces a deep learning-based intrusion detection system utilizing LSTM networks, capable of identifying diverse threats in IoT traffic with high accuracy. Trained on the CIC-IDS2017 dataset and integrated with the Hurst parameter, the proposed system exhibits exceptional resilience to Gaussian noise. The results demonstrate that this system can play a pivotal role in safeguarding IoT networks and ensuring their security against various threats. For the seamless integration of the proposed intrusion detection system within IoT nodes, a precise understanding of the architecture of IoT nodes is crucial. The architecture of IoT nodes, illustrated in Figure 1, primarily comprises sensor, processing, and communication subsystems. Each IoT node includes sensors that collect environmental data, microcontrollers that manage processing and control operations, and communication subsystems that enable data transmission between nodes and other devices or networks. These nodes are typically connected to edge servers through WiFi routers, establishing their internet connectivity. Compromising these nodes can disrupt their operation and directly affect the efficiency and security of the network. Therefore, a key metric for intrusion detection is evaluating IoT nodes' performance in terms of energy consumption, latency, and throughput. This research addresses one of the most critical security concerns in the digital era by evaluating and enhancing the performance of deep learning models for detecting threats in IoT traffic. In this study, a novel framework for anomaly detection in IoT traffic is presented, which utilizes the Hurst parameter for long-term memory analysis of the data and a hybrid LSTM model for identifying complex patterns. Furthermore, by injecting Gaussian noise into the data, an attempt is made to simulate real network conditions and assess the model's ability to detect anomalies under various scenarios. The results of this study show that using the Hurst parameter as a key feature can enhance the accuracy and reliability of intrusion detection systems. In this research, the Hurst parameter is a powerful tool for identifying evolutionary anomalies in time series data. The Hurst parameter can measure the self-similarity and long-term dependencies of network data, and by analyzing the oscillatory behavior of the data, it can detect unusual patterns. This feature proves particularly useful in detecting IoT traffic anomalies in the presence of Gaussian noise.

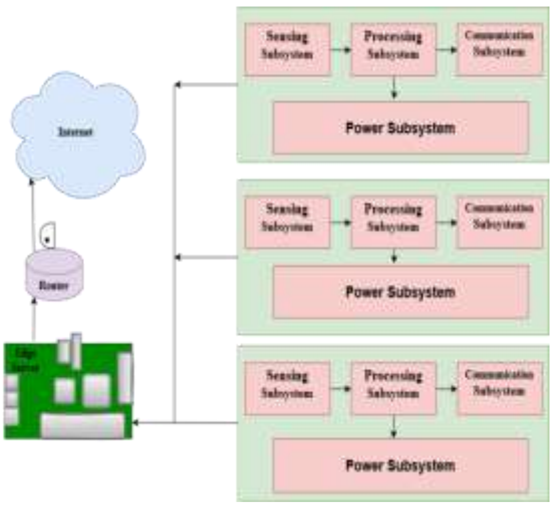

Figure 1. Architecture of IoT,

2. Related Works

Anomaly detection is a key area in data analysis and the security of Internet of Things (IoT) systems, aiming to identify abnormal behaviors in the data collected from IoT devices and networks [1,2]. In essence, anomaly detection is an approach to identifying unexpected or unusual behaviors that may indicate issues such as technical malfunctions, cyberattacks, or network disruptions [3]. Anomalies can significantly impact device performance and data security, making their swift and accurate detection essential for maintaining system integrity and preventing damage.

The rapid growth of IoT in recent years has led to the generation of massive amounts of data. These data, often containing time-series values and location-dependent information, pose challenges in storage, processing, and analysis due to their high volume and complex nature [4]. Among these challenges, Gaussian noise in the data can complicate the anomaly detection process. Thus, robust analytical algorithms and methods are required to effectively handle this noise and accurately detect anomalies [5].

One important approach for analyzing IoT data is the use of the Hurst parameter. This parameter, which represents the self-correlation and scalability characteristics of the data, can provide valuable insights into the long-term behavior of systems. Anomaly detection based on the Hurst parameter involves several operations that classify data into anomalies and normal behavior [5-6].

Anomaly detection is conducted at three levels: network, device, and event. At the network level

[7], the analysis concentrates on assessing overall traffic patterns to identify any unusual or abnormal behavior within the network. The device-level [8] entails a more granular approach, where the behavior of individual devices is scrutinized to detect anomalies specific to each device's activity. At the event level [9], the analysis targets particular events, identifying anomalies within specific time frames or geographic locations.

The methods for anomaly detection can generally be grouped into three categories: rule-based approaches, deep learning approaches, and hybrid models. Among these, deep learning has gained widespread adoption due to its ability to leverage machine learning algorithms and data-dependent features to classify behaviors as normal or anomalous [10].

Deep learning methods for anomaly detection have proven to be highly effective and powerful, utilizing architectures like recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory networks (LSTMs) to analyze complex data patterns and uncover hidden anomalies [14].

Given the limited research in this area, efforts have been made to utilize studies directly or indirectly related to the subject. Table 1 presents common methods used in previous studies along with their references.

Table1. Comparison of Various Approaches.

Articles	Network	Method	Resource & Implementation Limitation	Scalability	Evaluation & Comparison	Security Focus
[1]	Smart Home IoT	LSTM-based deep learning	Relies on LSTM models for IoT	Suitable for smart home IoT	Performance evaluation using LSTM	Intrusion Detection in Smart Home IoT
[2]	General IoT Networks	Deep neural networks	Relies on deep neural networks for IoT	Scalable for IoT networks	Evaluation using deep neural networks	Intrusion Detection for IoT Security
[3]	5G IoT Networks	Deep learning for IoT security	Needs 5G network infrastructure	Scalable for 5G IoT networks	Using deep learning for IoT network security	Intrusion Detection with 5G IoT
[4]	IoT Networks	Deep learning-based IDS	Relies on deep learning for IoT security	Suitable for IoT networks	Evaluation using deep learning for IoT	Intrusion Detection for IoT Network Security
[5]	IoT Networks	Decision tree classification	Depends on decision tree models	Evaluates noise impact in IoT security	Study of decision tree classification	Impact of Data Noise on IoT Security
[6]	IoT Networks	Machine learning-based feature extraction	Depends on feature extraction	Applicable for IoT environment s	Machine learning for IoT intrusion detection	Feature Extraction for Intrusion Detection
[7]	SDN Networks	Machine learning for SDN	SDN-specific limitations	Scalable for SDN networks	Machine learning for SDN intrusion detection	Intrusion Detection for SDN
[8]	Network IDS	Data resampling with deep learning	Class imbalance issues in datasets	Handles class imbalance in large datasets	Data resampling for class imbalance	Class Imbalance in Network IDS

[9]	IoT and SDN Systems	Deep learning for automatic intrusion detection	Requires IoT and SDN integration	Scalable for IoT and SDN	Deep learning for IoT and SDN systems	Securing IoT and SDN Systems
[10]	Smart Cities	Blockchain and machine learning	Complex privacy solutions	Scalable for smart cities	Blockchain for privacy-preserving security	Privacy-preserving Framework for Smart Cities
[11]	IoT Networks	AI techniques for IoT security	Resource-intensive AI methods	Scalable AI methods for IDS	Review of AI methods for IoT IDS	AI for Intrusion Detection in IoT
[12]	Industrial IoT	Hybrid CNN+LSTM	Requires hybrid CNN+LSTM architecture	Scalable for industrial IoT	Hybrid CNN+LSTM evaluation	Hybrid CNN+LSTM IDS for Industrial IoT
[13]	IoT Networks	Deep learning for network security	Resource and computational challenges	Scalable for IoT networks	Enhancing security through deep learning	Enhancing IoT Network Security
[14]	IoT and SDN Systems	SDN-based deep learning	SDN-specific issues	Scalable for IoT and SDN systems	SDN-enabled IDS for IoT networks	SDN-based IDS for IoT Networks
[15]	General Networks	Literature review of IDS methods	General IDS limitations	Applicable to various network environments	Review of IDS techniques	Systematic Review of IDS
[16]	IoT Networks	Deep learning-based IDS	Requires deep learning for real-time detection	Suitable for real-time IoT nodes	Real-time intrusion detection using deep learning	Deep-IDS for IoT Nodes

3. Data sets and Methodology

3.1. Datasets

The CIC-IDS2017 dataset, one of the most prominent resources for evaluating intrusion detection systems, contains 78 features and a class label for each sample. These features encompass various network traffic attributes and statistics related to packet data. Each sample in the dataset is labeled as either normal or belonging to one of the several types of network attacks. This dataset enables researchers to evaluate intrusion detection systems under various conditions and across different types of attacks, which is crucial for assessing the performance of deep learning algorithms in detecting network threats.

3.2. Hurst Parameter

The Hurst parameter is a statistical measure used to assess the self-similarity of a signal or pattern, typically in the context of time series data [24]. In our research, this parameter is utilized for anomaly detection in Internet of Things (IoT) network traffic using the CICIDS2017 dataset. We calculate the Hurst parameter for each data record to determine the degree of self-similarity in the network traffic.

The calculation process is as follows:

1. First, the time series data is divided into equally sized segments. For example, if the dataset contains 100 data points, it might be divided into segments, each containing 10 data points.
2. For each segment, we compute its range (R), which is the difference between the highest and lowest values, and its standard deviation (S), which measures the variability from the mean.
3. The ratio of R/S is then calculated for each segment, which is obtained by dividing the range by the standard deviation.

4. The logarithm of these R/S values is computed to normalize the data.
5. The average of these logarithmic values across all segments is calculated.
6. The Hurst parameter is then determined using the following formula:

$$H = \frac{E(\log((R/S))}{\log(n)}$$

The Hurst parameter is introduced as a distinctive feature in our dataset to quantify the self-similarity in network traffic patterns, a critical aspect of detecting modern intrusions. By using the Hurst parameter, our proposed model is not only capable of identifying obvious intrusions, but it also has the ability to detect and flag complex and hidden patterns, which are characteristic of advanced network threats. This strategic enhancement plays a vital role in improving the model's detection capabilities and contributes to the advancement of intrusion detection research.

3.3. Methodology

The proposed methodology for detecting and analyzing attacks in Internet of Things (IoT) networks consists of several stages. In the first step, a simulation of an IoT network is created, where each node is associated with sub-nodes. The edge server, which plays a critical role in the network, is defined as part of the system and is equipped with an Intrusion Detection System (IDS) model. During this stage, computations such as energy consumption and throughput for each node are performed and incorporated into the network's graph structure.

In the second stage, input data is loaded from a standardized dataset and undergoes preprocessing. This process includes the removal of invalid values, such as missing and out-of-bounds data, which are subsequently replaced with mean values. Data features are normalized, and labels are converted into a numerical format to prepare them for deep learning models. To address the issue of class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) is employed to balance the number of samples across each class.

In the third stage, to evaluate the system's resilience to noise, Gaussian noise with a normal distribution is added to the data. The added noise was generated randomly with a mean of zero and a standard deviation of 0.1, and was subsequently injected into the original data. By introducing Gaussian noise into the data, an attempt was made to simulate real network conditions and evaluate the model's ability to detect anomalies under various scenarios. The Hurst parameter, which reflects self-similarity, is calculated for both the original and noisy data, and a comparison is made to assess the impact of noise on data stability. This parameter is one of the key metrics in time series analysis, indicating the degree of self-similarity and the long-term dependence behavior of the data. This analysis contributes to improving the model's robustness in real-world conditions.

The proposed deep learning model consists of a Long Short-Term Memory (LSTM) network designed for attack analysis and detection. The input data is reconstructed in a three-dimensional format for use in this model. The network comprises two LSTM layers with different configurations, followed by an output layer using a Softmax activation function. To prevent overfitting, early stopping is applied during training. The model is trained using both training and validation datasets, and its performance is evaluated on test data.

In the next stage, results analysis involves calculating metrics such as accuracy, recall, precision, and the F1 score. Another crucial step is evaluating the model's performance using various metrics, including accuracy and error. Accuracy and error charts are plotted for both training and validation periods, allowing for a detailed examination of how these values evolve throughout the training process. These plots assist in identifying the model's strengths and weaknesses, as well as the impact of various configurations on performance. Confusion matrices are generated for specific attack classes, and Detection Rate (DR) and False Alarm Rate (FAR) are calculated at different threshold levels. The confusion matrix analysis provides detailed insights into the model's ability to identify attacks, helping pinpoint instances where the model failed to detect certain attacks.

Furthermore, the detection rate and response time for different types of attacks are presented in bar charts, and the model's performance in terms of accuracy and error during training and validation is examined. The anomaly detection phase is another key component of the methodology, which involves calculating the absolute error between predicted and actual values of the validation data. Samples with errors exceeding a specified threshold are identified as anomalies. The number and percentage of detected anomalies are calculated, and a distribution chart of these anomalies is plotted in comparison with predicted and actual values. This analysis helps identify abnormal behavioral patterns and their correlation with attack data.

In the subsequent stage, the frequency of attacks and anomalies in training and validation datasets is examined. The number of attacks and normal samples is calculated for each dataset, and the results are presented in comparative bar charts. This analysis provides a deeper understanding of the data distribution and its relationship with the model's detection accuracy.

Another critical step involves evaluating the model's performance through various metrics such as accuracy and error. Accuracy and error charts are plotted for both the training and validation phases and trends in these values are analyzed throughout the training period. These charts aid in identifying strengths and weaknesses in the model, as well as the effects of different configuration settings on its performance.

This methodology, by offering a comprehensive framework, integrates network simulation, data analysis, and deep learning techniques to enhance the detection and analysis of attacks in IoT networks.

The proposed method for identifying and removing noise from the data consists of several key steps that have been proven effective. Initially, statistical analyses, such as identifying outliers and examining data distributions, are used to detect anomalies. Then, invalid values in the data context are converted to NaN and replaced using the mean and statistical methods to improve data quality. Adding controlled noise to the data enables the model to learn more effectively when faced with noise, enhancing its resilience to nonlinear and noisy data. Additionally, calculating the Hurst parameter, as a proven method for trend analysis and accuracy evaluation, helps identify structural changes in noisy data. The use of techniques such as SMOTE to address class imbalance and improve data quality in machine learning models is also well-documented in the literature. Moreover, Dropout layers in neural network architectures, such as LSTM, help reduce noise learning and prevent overfitting. Finally, analyzing predicted errors and establishing a specific threshold for anomaly detection is an effective method for noise identification and removal, which can enhance the accuracy of machine-learning model results (Figure 2).

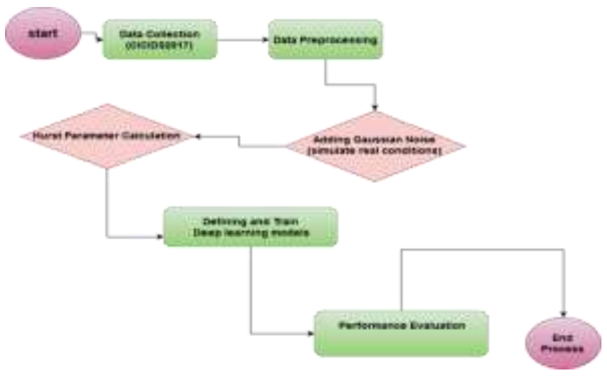
start
Data Collection (CICIDS2017)
Data Preprocessing
Adding Gaussian Noise (simulate real conditions)
Hurst Parameter Calculation
Defining and Train Deep learning models
Performance Evaluation
End Process

Figure 2. Flowchart for the proposed method.

3.4 Implementation of Anomaly Detection in IoT Network Traffic Using the LSTM Model

In this research, the goal is to detect anomalies in the traffic of Internet of Things (IoT) networks. For this purpose, Long Short-Term Memory (LSTM) networks are used to identify normal and abnormal traffic behaviors. This model utilizes real network traffic data to detect attacks and abnormal behaviors in IoT networks, and an IDS model based on LSTM is developed for this task. The implementation steps are as follows:

1. Creation and Definition of the IoT Network

Initially, an IoT network is created using the NetworkX library. In this network, the nodes represent sensors, edge servers, and gateways, each with its own specific features such as energy consumption, communication protocols, and performance characteristics. These nodes are randomly assigned properties like active time, idle time, power consumption, etc.

- **Definition of Nodes**: The nodes include active sensors, microcontroller processors, and various communication protocols such as IEEE802.15.4, Bluetooth LE, and LoRaWAN.
- **Sensor Nodes**: Defined with power consumption and communication protocols such as IEEE802.15.4 and Bluetooth LE.
- **Edge Server**: Acts as the central processing hub, receiving data from sensors and sending it to gateways.
- **Gateways**: Designed to connect the network to the internet and send data from the edge server to the internet network.

Code for creating the IoT network graph:

```
G = nx.Graph()
```

```
# Define IoT nodes with power
consumption and communication
protocols
for i in range(10):
    T_active = np.random.uniform(1, 5)  #
Active time
    T_idle = np.random.uniform(5, 10)  #
Idle time
    P_active = np.random.uniform(0.1, 0.5)
# Active power
    P_idle = np.random.uniform(0.05, 0.1)
# Idle power
    E = T_active * P_active + T_idle *
P_idle  # Energy consumption
    D = np.random.uniform(100, 500)  #
Data size
    T = np.random.uniform(1, 10)  # Task
duration
    TH = D / T  # Power calculation
    G.add_node(f'node_{i+1}',
               type='node',
               sensor_subsystem='active',

processing_subsystem='microcontroller',

communication_subsystem=np.random.ch
oice(['IEEE802.15.4', 'Bluetooth LE',
'LoRaWAN']),
               energy_subsystem='solar_panel',
               energy_usage=E,
               delay=np.random.uniform(0.1,
1.0),
               throughput=TH)
# Add edge server and gateways to the
network
G.add_node('edge_server',
type='edge_server', protocol='Cellular
IoT', ids_model='LSTM IDS Model')
for i in range(5):
    G.add_node(f'gateway_{i+1}',
               type='gateway',

protocol=np.random.choice(['LoRaWAN',
'Cellular IoT']),
               connection='internet')
# Define network connections
for i in range(10):
    G.add_edge(f'node_{i+1}',
'edge_server',
protocol=G.nodes[f'node_{i+1}']['commu
nication_subsystem'])
for i in range(5):
    G.add_edge('edge_server',
f'gateway_{i+1}', protocol='WiFi')
# Calculate energy consumption
energy_consumption =
sum([G.nodes[node]['energy_usage'] for
node in G.nodes if G.nodes[node]['type']
== 'node'])
print(f"Total Energy Consumption:
{energy_consumption:.2f} J")
```

2. Data Loading and Preprocessing

Data is loaded from a CSV file. Initial preprocessing steps are performed to handle missing values, correct erroneous data, and normalize the data. Additionally, SMOTE is used to address class imbalance. Then, the data is reshaped into sequences to serve as input to the LSTM model. The preprocessing steps include:

- **Handling Missing Values (NaN)**: Missing values for features are replaced with the mean of those features.
- **Defining Feature and Label Columns**: Feature and label columns are extracted from the data.
- **Converting Labels to Numerical Format**: Labels are converted to a binary or one-hot encoding format.
- **Normalizing Features**: Features are normalized using StandardScaler to ensure consistent value ranges.

Code for data preprocessing:

```
data_path =
'/content/drive/MyDrive/data.csv'
data = pd.read_csv(data_path)
# Preprocess data
data.columns = data.columns.str.strip()  #
Remove extra spaces from column names
label_column = 'Label'  # Label column
name
feature_columns = [col for col in
data.columns if col != label_column]  #
Extract features
features = data[feature_columns]
labels = data[label_column]
# Replace incorrect values with column
mean
features.replace([np.inf, -np.inf], np.nan,
inplace=True)
features.fillna(features.mean(),
inplace=True)
# Convert labels to numerical format
labels = pd.get_dummies(labels).values
# Normalize features
scaler = StandardScaler()
features = scaler.fit_transform(features)
```

3. Adding Noise to the Data

To improve the model's generalization capability and simulate real-world conditions where data may have noise, noise is added to the features. This noise is generated from a normal distribution with a mean of 0 and a standard deviation of 0.1.

- **Noise**: Gaussian noise is added to the features to simulate real-world data conditions.

Code for adding noise:

```
def AddNoise(data):
   mu, sigma = 0, 0.1  # Mean and standard deviation of noise
   noise = np.random.normal(mu, sigma, np.shape(data))   # Generate noise from normal distribution
   noised_data = data + noise  # Add noise to the data
   return noised_data
X_noised = AddNoise(features)
```

4. Calculation of the Hurst Parameter

The Hurst exponent is used to model the self-similarity behavior of the data and detect long-term trends. This parameter is especially useful in time-series analysis and can also be beneficial in traffic data analysis.

- **Raw and Noisy Data**: The Hurst parameter is computed for both the raw and noisy data.

Therefore, the Hurst parameter is calculated for each numerical feature in the dataset. This is done using the hurst_exponent function, which computes the Hurst exponent for each feature and stores the result in the hurst_features list. If any NaN values arise during the calculation, they are replaced with 0 using the np.nan_to_num function to prevent errors. Next, the data is standardized using the StandardScaler to ensure that the features are on the same scale, which is particularly important for machine learning models like LSTM that are sensitive to the scale of the data. After the data is scaled, the calculated Hurst parameters need to be added to the scaled data. To achieve this, the Hurst values are first reshaped into the appropriate dimensions. Then, the Hurst values are repeated for each data sample to match the number of samples in the feature set. If the number of samples in the Hurst values and the feature data does not align, np.tile is used to repeat the Hurst values until they match the number of feature samples. Finally, the scaled data and the Hurst parameters are concatenated together, creating a final dataset (X_scaled_hurst) that contains the original features along with the Hurst parameters. This combined dataset is then ready for use in machine learning models, such as LSTM, to leverage the additional feature of the Hurst parameter.

Code for Hurst parameter calculation:

```
H_raw = nolds.hurst_rs(features.flatten())
# Calculate Hurst for raw data
print(f"Hurst Parameter (Original Data): {H_raw:.4f}")

H_noised = nolds.hurst_rs(X_noised.flatten())          # Calculate Hurst for noisy data
print(f"Hurst Parameter (Noised Data): {H_noised:.4f}")
# --- Calculate Hurst Exponent for each feature ---
def hurst_exponent(ts):
   ts = np.array(ts)
   ts = ts[~np.isnan(ts)]
   if len(ts) <= 1 or np.std(ts) == 0:
      return np.nan
   l = len(ts)
   T = np.arange(1, l + 1)
   R = np.cumsum(ts - np.mean(ts))
   S = np.std(ts)
   try:
      Y = np.log10(R / S)
      X = np.log10(T)
   except ValueError:
      return np.nan
   fit = np.polyfit(X, Y, 1)
   return fit[0]

# Calculate Hurst Exponent for each feature
hurst_features = []
for col in features.select_dtypes(include=[np.number]).columns:
   hurst_value = hurst_exponent(features[col].values)
   hurst_features.append(hurst_value)

# Replace NaN values with 0
hurst_features = np.nan_to_num(np.array(hurst_features), nan=0.0)

# --- Preprocess the data ---
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)

# --- Add Hurst Exponent to features ---
hurst_features = hurst_features.reshape(-1,
1)
hurst_features_repeated =
np.tile(hurst_features, (X_scaled.shape[0]
// len(hurst_features), 1))

if len(hurst_features_repeated) >
X_scaled.shape[0]:
   hurst_features_repeated =
hurst_features_repeated[:X_scaled.shape[
0]]
elif len(hurst_features_repeated) <
X_scaled.shape[0]:
   hurst_features_repeated =
np.tile(hurst_features_repeated,
(X_scaled.shape[0] //
len(hurst_features_repeated) + 1, 1))
   hurst_features_repeated =
hurst_features_repeated[:X_scaled.shape[
0]]

X_scaled_hurst =
np.concatenate([X_scaled,
hurst_features_repeated], axis=1)
```

5. Building and Training the LSTM Model
After preprocessing, an LSTM model is used for anomaly detection in network traffic. LSTM is ideal for learning long-term dependencies in sequential data, making it suitable for traffic data with temporal features.

Model Architecture:

- **LSTM Layers**: The model consists of LSTM layers for learning time-dependent patterns, with Dropout layers to prevent overfitting.
- **Input Shape**: Data needs to be reshaped into sequences, i.e., timesteps and features.
- **Layers**:
  - First LSTM layer with 128 units.
  - Second LSTM layer with 64 units.
  - Dropout layers for regularization.
  - A final Dense layer with a softmax activation function for multi-class classification.

Code for LSTM model definition and training:

```
model = Sequential()
model.add(LSTM(128,
input_shape=(X_train.shape[1],
X_train.shape[2]),
return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64,
return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(y_train.shape[1],
activation='softmax'))
# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
# EarlyStopping for overfitting prevention
early_stopping =
EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)
# Train the model
history = model.fit(X_train, y_train,
epochs=50, batch_size=64,
validation_split=0.2,
callbacks=[early_stopping], verbose=1)
```

6. Model Evaluation
After training, the model is evaluated using test data to compute the overall accuracy. Additionally, recall is calculated to evaluate the model's performance in detecting anomalies.
Code for model evaluation:

```
loss, accuracy = model.evaluate(X_test,
y_test)
print(f"Test Accuracy: {accuracy *
100:.2f}%")

y_pred = model.predict(Test)
y_pred_classes = np.argmax(y_pred,
axis=1)
y_true = np.argmax(y_test, axis=1)
# Calculate accuracy
test accuracy = accuracy_score(y_true,
y_pred_classes)
print(f"Test Accuracy: {test_accuracy *
100:.2f}%")
# Calculate Recall
recall = recall_score(y_true,
y_pred_classes, average='macro')
print(f'Accuracy: {test_accuracy *
100:.2f}%')
print(f'Recall: {recall * 100:.2f}%')
```

7. Anomaly Detection
At this stage, the model is used to detect anomalies. Anomalies are defined as samples that are incorrectly classified and likely exhibit abnormal behavior.

Code for error calculation and anomaly detection:

```
error = np.abs(y_pred.flatten() - y_test.flatten())  # Absolute error
threshold = 0.5  # Threshold for anomaly detection

# Detect anomalies
anomalies = error > threshold

# Display anomaly results
print(f"Number of anomalies detected: {np.sum(anomalies)}")
print(f"Percentage of anomalies: {np.mean(anomalies) * 100:.2f}%")
```

8. Displaying Anomalies and Comparison with Attacks

To observe and compare the behavior of anomalies and attacks in the dataset, various plots and matrices are used.

1. **Displaying Anomalies**:
   - Anomalies are displayed as red dots on the plot of predicted and true labels.
2. **Comparing the Frequency of Anomalies and Attacks**:
   - The number of attack samples and anomalies in the training and test datasets are counted and displayed.

```
# Plot anomalies

plt.figure(figsize=(10, 6))

plt.plot(y_val.flatten(), label='True Labels', color='blue', alpha=0.5)  # True labels

plt.plot(predicted.flatten(), label='Predicted Labels', color='orange', alpha=0.5)  # Predicted labels

plt.scatter(np.where(anomalies)[0], predicted.flatten()[anomalies], color='red', label='Anomalies', zorder=5)  # Anomalies

plt.title('Anomaly Detection in Traffic Behavior')

plt.xlabel ('Sample Index')

plt.ylabel('Prediction Value')

plt.legend()

plt.show()

# Step 6: Comparison of Anomaly and Attack Frequency

train_attack_counts = np.sum(y_train.argmax(axis=1) == 1)

train_benign_counts = np.sum(y_train.argmax(axis=1) == 0)

val_attack_counts = np.sum(y_val.argmax(axis=1) == 1)

val_benign_counts = np.sum(y_val.argmax(axis=1) == 0)

# Print attack and anomaly counts

print(f"Training set - Attack samples: {train_attack_counts}, Benign samples: {train_benign_counts}")

print(f"Validation set - Attack samples: {val_attack_counts}, Benign samples: {val_benign_counts}")

# Plot comparison of anomaly and attack frequency

labels = ['Benign', 'Attack']

train_counts = [train_benign_counts, train_attack_counts]

val_counts = [val_benign_counts, val_attack_counts]

x = np.arange(len(labels))  # Number of categories

width = 0.35  # Bar width

fig, ax = plt.subplots(figsize=(8, 6))

# Plot training and validation bars

rects1 = ax.bar(x - width/2, train_counts, width, label='Training', color='blue')
```

```
rects2 = ax.bar(x + width/2, val_counts, width, label='Validation', color='red')

# Add labels and titles

ax.set_ylabel('Counts')

ax.set_title('Anomaly and Attack Frequency Comparison')

ax.set_xticks(x)

ax.set_xticklabels(labels)

ax.legend()

# Add labels on bars

def add_labels(rects):

    for rect in rects:

        height = rect.get_height()

        ax.annotate('{}'.format(height),

                    xy=(rect.get_x() + rect.get_width() / 2, height),

                    xytext=(0, 3),  # Offset label

                    textcoords="offset points",

                    ha='center', va='bottom')

add_labels(rects1)

add_labels(rects2)

plt.show()
```

9. Confusion Matrix

For a more detailed evaluation of the model, a confusion matrix is calculated for the predictions and true labels. This matrix shows how many samples were correctly classified and how many were misclassified.

10. Accuracy and Loss Plots

Finally, to visualize the training and evaluation trends of the model, accuracy (Figure 3) and loss (Figure 4) plots are generated for the training and validation data. The data is divided into two sections: 80% of the data is used for training the model and is allocated to the training set, while the remaining 20% is designated for evaluating the model's performance on unseen data and is assigned to the testing set.

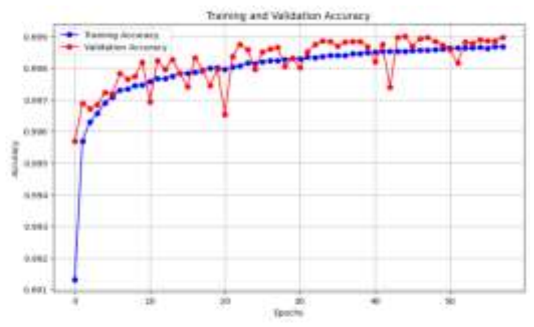

Figure 3. Accuracy graph of LSTM with CIC-IDS2017 dataset.

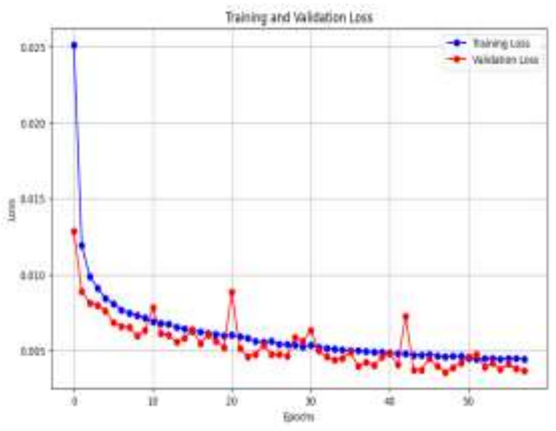

Figure 4. Loss graph of LSTM with CIC-IDS2017 dataset.

Table 3 presents a comparison of the performance between the proposed method and other similar approaches. This comparison highlights the significant superiority of the proposed method in intrusion detection over several existing systems, which is clearly demonstrated through a comprehensive evaluation of model performance. The proposed method, with an accuracy of 99.90%, has made a substantial improvement over other similar studies. Specifically, none of the other methods have been able to achieve the balanced performance across all metrics that our proposed method has demonstrated. This comparison clearly showcases the power and efficiency of the proposed method in accurately detecting and classifying attacks and intrusions, setting a new standard for evaluating cybersecurity systems in the field of IoT.

Table 3. Comparison with other similar articles.

Article/Year	Accuracy	Precision	Reacall	F1-Score
5/2023	90%	60%	90%	70%
4/2024	99/7	99	99/7	98
Proposed method	99/90%	99%	99/90%	98/93%

4. Results

4.1 Evaluation

The model evaluation is conducted using several different metrics that thoroughly examine and analyze the model's performance. Initially, the model's accuracy is calculated using the test data, with the final output presented as a percentage. The Hurst parameter value for the LSTM model is measured at 0.8253 on the original data and 0.8228 on the noisy data.

Next, the model's performance in identifying various attacks is assessed using the Confusion Matrix. To perform a more detailed analysis, the Confusion Matrix is visualized, breaking down the number of correct and incorrect samples for each class, Benign and Attack, providing a clearer understanding of the model's performance. For a more precise evaluation, confusion matrices are specifically plotted for different types of attacks. This helps in accurately identifying attacks and comparing predictions with true labels for each attack type. In the Confusion Matrix for performance evaluation, attacks are precisely categorized into various types, including BRF (Brute Force Attack), MITM (Man-in-the-Middle Attack), DDoS (Distributed Denial of Service Attack), RP (Replay Attack), Port Scan, Botnet, Heartbleed, Infiltration, Spoofing, SQL Injection, and AML (Adversarial Machine Learning Attack). This categorization aids the model in assessing its accuracy in detecting different types of attacks and allows for the measurement of true and false performance rates in each class. The confusion matrices shown in Figures 5, 6, and 7 illustrate the performance of the proposed method in classifying network traffic into one of six classes, demonstrating the system's efficacy in identifying these categories.

Advanced evaluation metrics used in this paper for assessing the Intrusion Detection System (IDS) performance are calculated from the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) of the Confusion Matrix presented in Figures 4, 5, and 6. Accuracy, Precision, Recall (Sensitivity), and F1 score are defined in Equations 1, 2, 3, and 4, respectively, in Table 2. Additionally, Precision, Recall, and F1 scores are calculated separately to comprehensively evaluate the model's performance in detecting various attacks. These metrics are used to ensure a balance between precision and recall and to prevent potential errors in the model. The evaluation results show that the model has achieved an intrusion detection rate of 99.90%, overall accuracy of 99.90%, and an F1 score of 98.93%, demonstrating good performance in detecting various types of attacks. Furthermore, techniques such as EarlyStopping have been used to prevent overfitting, ensuring reliable results in the final evaluation.

In the next step, graphs are plotted to display the Detection Rate and False Alarm Rate at different thresholds, which helps to better understand the model's efficiency under various conditions. Additionally, the detection rate and response time for different attacks are displayed in bar charts to highlight the model's performance in identifying various attacks and its speed. The detection rates for various attacks, including BRF, MITM, DDoS, RP, Port Scan, Botnet, Heartbleed, Infiltration, Replay, Spoofing, SQL Injection, and AML, are as follows: 100%, 100%, 100%, 98%, 98%, 98%, 100%, 98%, 100%, 98%, 100%, and 100%, respectively. The corresponding response times for these attacks are 1.46, 1.05, 2.02, 0.92, 1.20, 1.30, 1.75, 1.60, 1.10, 1.25, 1.85, and 1.50 seconds, as shown in Figures 8 and 9. These results indicate the model's high accuracy in detecting various attacks and its good performance in terms of detection rate and response time.

Next, anomaly detection is performed. To achieve this, the absolute error between predicted values and true values is first calculated. Then, using a predefined threshold (set to 0.5 in this case), samples with an error greater than this threshold are identified as anomalies (Figure 10). These anomalies are plotted on a graph that displays both predicted and true labels, providing a clear visualization of the detected anomalies (Figure 11). This information helps in identifying abnormal behaviors or potential attacks. Finally, a comparison is made between the number of attack samples and benign samples in the training and validation sets. This comparison provides insight into the distribution of data across different sets and aids in a more detailed analysis of how attacks and benign samples are distributed. The bar chart created visually represents this comparison, with labels added to indicate the exact number of samples in each category. These evaluations help in a better understanding of the model's performance and its ability to detect various attacks (Figure 12).

Table 2. Equations Used.

Equations	number
$\text{Recall} / TPR = \frac{TP}{TP + FN}$	(1)
$\text{Precision} / PPV = \frac{TP}{TP + FP}$	(2)
$ACC = \frac{TP + TN}{TP + TN + FP + FN}$	(3)
$F1score = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$	(4)

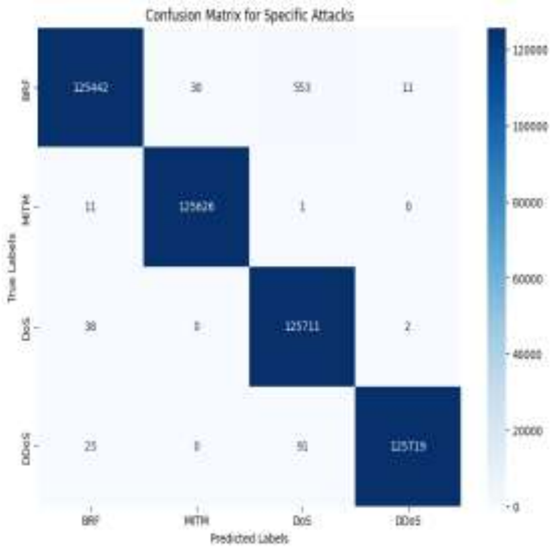

Figure 5. Confusion matrix.

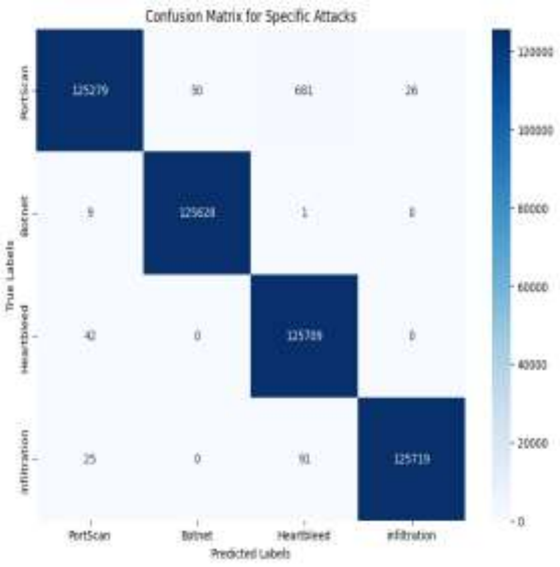

Figure 6. Confusion matrix.

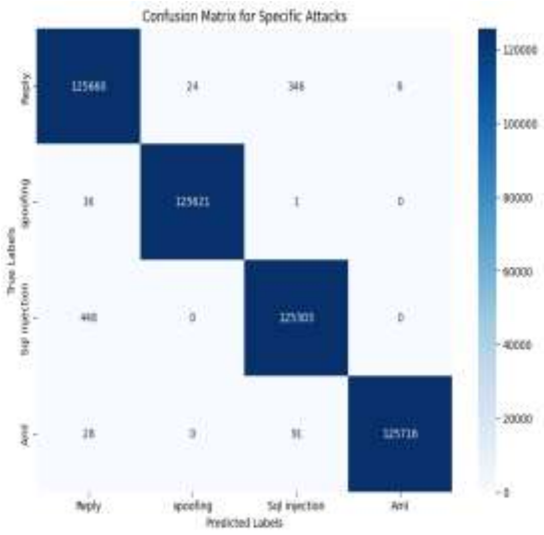

Figure 7. Confusion matrix.

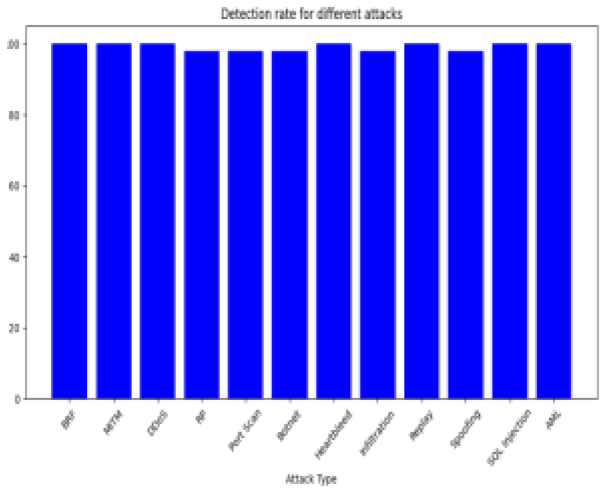

Figure 8. Detector rate different attacks.

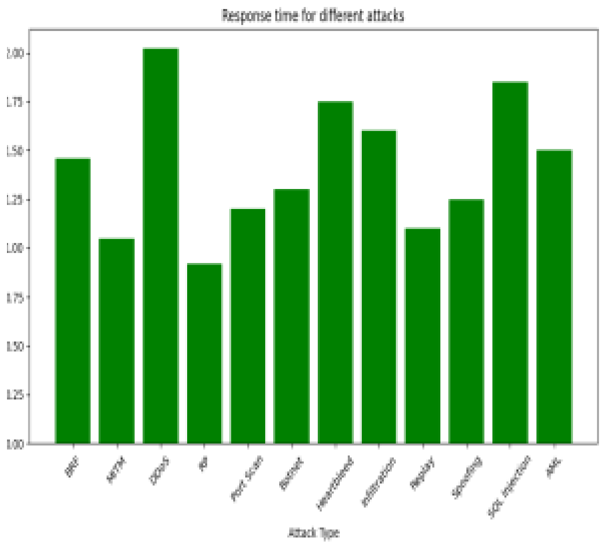

Figure 9. Response time different attacks.

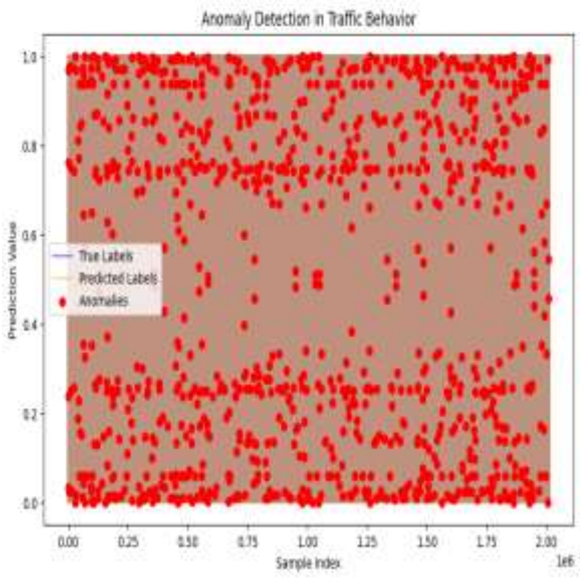

Figure 10. Anomaly Detection in Traffic Behavior Using LSTM in the Presence of Gaussian Noise and the Hurst Parameter.

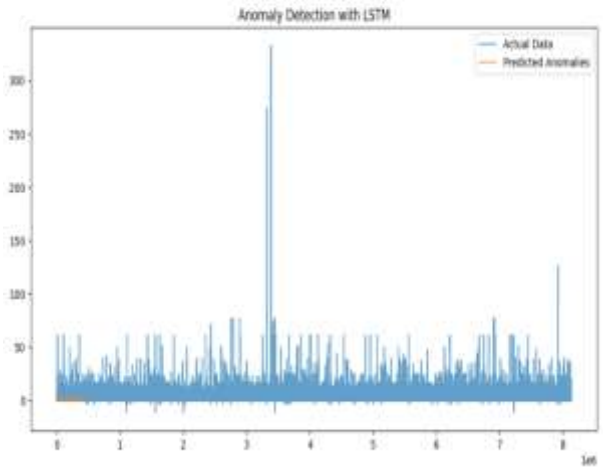

Figure 11. Real Data and Anomaly Predictions by the LSTM Model in the Presence of Gaussian Noise and the Hurst Parameter.

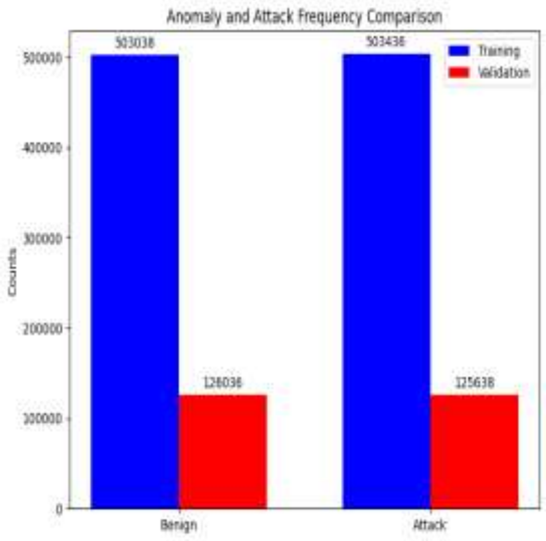

Figure 12. Comparison of the Frequency of Normal and Attack Samples Using LSTM in the Presence of Gaussian Noise and the Hurst Parameter.

4.2 Discussion and Comparison

In this study, the proposed model, which combines LSTM networks and the Hurst Exponent parameter for anomaly detection in IoT traffic, has achieved significant results, outperforming related works. One of the key innovations of this research is the use of the combination of LSTM and the Hurst parameter as a tool for simulating self-similarity and dynamic data analysis of network traffic. This combination allows the model to simulate and analyze more complex features of the data, which proves to be highly effective in noisy conditions. In this context, the Hurst parameter helps simulate long-term self-similarity trends in traffic data, a feature that is highly beneficial for detecting anomalies in complex network conditions. Moreover, the process of adding Gaussian noise to the training data improves the model's generalizability when facing unfamiliar and real-world data. This technique enables the model to detect attacks and anomalies in real-world conditions, which are often noisy, as many IoT data in real-world environments are influenced by unpredictable factors. The use of precise preprocessing techniques, such as feature standardization and replacing missing values with the mean of the features, has significantly improved the quality of the input data. This, in turn, enhances the model's accuracy during the learning process and results in more stable outcomes. In terms of model optimization, the use of the Adam optimization algorithm alongside EarlyStopping to prevent overfitting and accelerate model convergence has played a critical role in improving its performance. These methods ensure that the model halts before overfitting occurs and that the best weights are retained throughout the training process. Ultimately, the proposed model has shown a significant improvement in identifying more complex attacks compared to similar models, demonstrating better performance in anomaly and attack detection. Particularly in IoT networks, which are typically exposed to high noise and unstable data, this model has been able to simulate and detect anomalies in real-world conditions with greater accuracy. The performance improvements of the proposed model over previous methods are attributed to the combination of innovative data processing techniques and the use of advanced optimization algorithms. This combination not only provides higher accuracy in detecting attacks and anomalies but also makes the model applicable in complex, real-world environments.

5. Conclusion

In this study, a deep learning-based intrusion detection system utilizing an optimized LSTM architecture is introduced, demonstrating exceptional performance in detecting various attacks within Internet of Things (IoT) networks. The system achieves an accuracy of 99.90% and an F1 score of 98.93%, effectively identifying complex threats such as DoS and DDoS attacks. One of the key innovations of this research is the integration of the Hurst parameter with deep learning models, which enhances the model's resilience to Gaussian noise and improves its performance in detecting threats within IoT traffic. The findings underscore the importance of employing advanced statistical features and designing noise-resistant models, with the proposed solutions representing a significant step forward in enhancing the security of IoT networks and safeguarding smart infrastructures.

References

[1] S. W. Azumah, N. Elsayed, V. Adewopo, Z. S. Zaghloul, and C. Li, "A deep LSTM based approach for intrusion Detector IoT devices network in smart home," in *Proc. IEEE 7th World Forum Internet Things* (WF-IoT), Jun. 2021, pp. 836–841.

[2] M. Ahsan, N. Rifat, M. Chowdhury, and R. Gomes, "Intrusion Detector for IoT network security with deep neural network," in *Proc. IEEE Int. Conf. Electro Inf. Technol. (eIT)*, May 2022, pp. 467–472.

[3] N. Yadav, S. Pande, A. Khamparia, and D. Gupta, "Intrusion Detector system on IoT with 5G network using deep learning," *Wireless Commun. Mobile Comput.*, vol. 2022, pp. 1–13, Mar. 2022.

[4] R. Morshedi, S. M. Matinkhah, and M. T. Sadeghi, "Intrusion detection for IoT network security with deep learning," Journal of Artificial Intelligence and Data Mining 12, no. 1 (2024): 37–55.

[5] S. M. Matinkhah, R. Morshedi, and Seyed Akbar Mostafavi, "Exploring Impact of Data Noise on IoT Security: a Study using Decision Tree Classification in Intrusion Detection Systems," Journal of Artificial Intelligence and Data Mining (JAIDM), vol. 11, no. 4, pp. 609-626, 2023.

[6] D. Musleh, M. Alotaibi, F. Alhaidari, A. Rahman, and R. M. Mohammad, "Intrusion Detector system using feature extraction with machine learning algorithms in IoT," *J. Sensor Actuator Netw.*, vol. 12, no. 2, p. 29, Mar. 2023.

[7] G. Logeswari, S. Bose, and T. Anitha, "An intrusion Detector system for SDN using machine learning," *Intell. Autom. Soft Comput.*, vol. 35, no. 1, pp. 867–880, 2023.

[8] A. Abdelkhalek and M. Mashaly, "Addressing the class imbalance problem in network intrusion Detector systems using data resampling and deep learning," *J. Supercomput.*, vol. 79, no. 10, pp. 10611–10644, Jul. 2023.

[9] R. A. Elsayed, R. A. Hamada, M. I. Abdalla, and S. A. Elsaid, "Securing IoT and SDN systems using deep-learning based automatic intrusion Detector," *Ain Shams Eng. J.*, vol. 14, no. 10, Oct. 2023, Art. no. 102211.

[10] P. Kumar, G. P. Gupta, and R. Tripathi, "TP2SF: A trustworthy privacy-preserving secured framework for sustainable smart cities by leveraging blockchain and machine learning," *J. Syst. Archit.*, vol. 115, May 2021, Art. no. 101954.

[11] M. Saied, S. Guirguis, and M. Madbouly, "Review of artificial intelligence for enhancing intrusion Detector in the Internet of Things," *Eng. Appl. Artif. Intell.*, vol. 127, Jan. 2024, Art. no. 107231.

[12] H. C. Altunay and Z. Albayrak, "A hybrid CNN+LSTM-based intrusion Detector system for industrial IoT networks," *Eng. Sci. Technol., Int. J.*, vol. 38, Feb. 2023, Art. no. 101322.

[13] S. A. Bakhsh, M. A. Khan, F. Ahmed, M. S. Alshehri, H. Ali, and J. Ahmad, "Enhancing IoT network security through deep learning-powered intrusion Detector system," *Internet Things*, vol. 24, Dec. 2023, Art. no. 100936.

[14] R. Chaganti, W. Suliman, V. Ravi, and A. Dua, "Deep learning approach for SDN-enabled intrusion Detector system in IoT networks," *Information*, vol. 14, no. 1, p. 41, 2023.

[15] O. H. Abdulganiyu, T. Ait Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion Detector system (IDS)," *Int. J. Inf. Secur.*, vol. 22, no. 5, pp. 1125–1162, Oct. 2023.

[16] S. Racherla, P. Sripathi, N. Faruqui, M. A. Kabir, M. Whaiduzzaman, and S. A. Shah, "*Deep-IDS: A Real-Time Intrusion Detector for IoT Nodes Using Deep Learning*," *IEEE Access*, 2024.

تشخیص ناهنجاری در ترافیک اینترنت اشیاء در حضور نویز گوسی با استفاده از شبکه‌های عصبی عمیق

رؤیا مرشدی و سید مجتبی متین خواه*

گروه مهندسی کامپیوتر، دانشگاه یزد، یزد، ایران.

ارسال ۲۰۲۵/۰۱/۲۷ بازنگری ۲۰۲۵/۰۲/۲۸؛ پذیرش ۲۰۲۵/۰۴/۱۵

چکیده:

اینترنت اشیاء (IoT) یکی از حوزه‌های در حال رشد سریع به شمار می‌رود و نقش محوری در ارائه خدمات هوشمند مدرن ایفا می‌کند. با این حال، محدودیت‌های منابع در گره‌های IoT چالش‌های امنیتی قابل‌توجهی را به همراه دارد و آن‌ها را در برابر حملات سایبری آسیب‌پذیر می‌سازد. در سال‌های اخیر، استفاده از مدل‌های یادگیری عمیق به‌عنوان رویکردی مؤثر برای شناسایی ناهنجاری‌ها در ترافیک IoT مطرح شده است. یکی از چالش‌های اصلی این رویکردها، تأثیر منفی نویز گوسی بر دقت شناسایی مدل‌ها است. در این مطالعه، یک سامانه تشخیص نفوذ مبتنی بر یادگیری عمیق پیشنهاد شده است که از معماری ساده و بهینه‌شده LSTM با ۱۲۸ واحد حافظه بهره می‌برد. این مدل با استفاده از مجموعه‌داده CIC-IDS2017 آموزش داده شده و برای پیاده‌سازی در سرورهای لبه‌ای طراحی گردیده است. نتایج ارزیابی عملکرد نشان می‌دهد که این سامانه توانایی بالایی در شناسایی حملات متنوعی همچون DoS، DDoS و سایر تهدیدات پیشرفته دارد. از ویژگی‌های برجسته این سیستم می‌توان به نرخ شناسایی ۹۹٫۹۰٪، دقت کلی ۹۹٫۹۰٪ و امتیاز F1 معادل ۹۸٫۹۳٪ اشاره کرد. یکی از نوآوری‌های این پژوهش بررسی تأثیر ادغام پارامتر هرست با مدل‌های یادگیری عمیق است. نتایج حاکی از آن است که این ادغام، مقاومت مدل را در برابر نویز گوسی افزایش داده و عملکرد آن را در شناسایی تهدیدات در ترافیک IoT بهبود می‌بخشد. یافته‌های این تحقیق بر اهمیت بهره‌گیری از ویژگی‌های آماری پیشرفته و طراحی مدل‌های مقاوم در برابر نویز برای تأمین امنیت سایبری شبکه‌های IoT تأکید دارد. سامانه پیشنهادی با عملکرد دقیق، زمان پاسخ‌دهی سریع و راهبرد دفاعی منحصربه‌فرد خود، گامی مؤثر در جهت ارتقای امنیت شبکه‌های IoT و حفاظت از زیرساخت‌های هوشمند محسوب می‌شود. این پژوهش می‌تواند راهکاری کارآمد برای توسعه سامانه‌های امنیتی در مقابله با تهدیدات پیچیده سایبری باشد.

کلمات کلیدی: پارامتر هرست، تشخیص نفوذ، تشخیص ناهنجاری، شبکه LSTM، نویز گوسی.