Computer Fraud and Security
ISSN (online): 1873-7056

A Lightweight Network Intrusion Detection System Based on Temporal Convolutional Networks and Attention Mechanisms

Jianan Li, Luqun Li*

The College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 201418, China

Abstract: The proliferation of Internet of Things (IoT) devices has led to a significant increase in the number and complexity of cyberattacks, posing severe challenges to global cybersecurity and exposing the limitations of traditional signature-based intrusion detection systems (IDS) in addressing unknown or evolving threats. To tackle this issue, we propose TCNSE, a network intrusion detection model that integrates a Temporal Convolutional Network (TCN) with a Squeeze-and- Excitation (SE) module. The TCN effectively captures temporal dependencies in network traffic through dilated convolutions, while the SE module enhances feature representation by modeling inter-channel relationships. Our model maintains high detection accuracy with low computational complexity, making it suitable for deployment on resource-constrained IoT devices. We applied our model to the CIC-IDS 2018 dataset, implementing efficient data preprocessing and feature selection techniques that reduced the number of features from over 80 to 20. We also addressed class imbalance using hybrid sampling methods and Focal Loss. Experimental results demonstrate that the TCNSE model outperforms existing advanced intrusion detection models, achieving key performance metrics of 98.4% accuracy, 99.4% precision, 98.7% recall, and a 99.1% F1 score.

Keywords: Intrusion Detection System; Temporal Convolutional Networks;Attention Mechanisms; CIC-IDS 2018; Feature Selection

INTRODUCTION

The rapid evolution of information technology and the widespread expansion of the Internet have significantly increased both the frequency and complexity of cyberattacks. Network intrusions now present severe challenges to global cybersecurity, often resulting in critical issues such as data breaches, identity theft, and data tampering[1]. For instance, the 2023 FBI Internet Crime Report [2] indicates that cybercrime losses in the United States exceeded $12.5 billion in 2023—a 22% increase from the previous year. Furthermore, with nearly 38 billion IoT devices installed globally by the end of 2023 [3], many of which are equipped with inadequate security measures, the attack surface has expanded considerably. A notable example is the September 20, 2016, Mirai botnet DDoS attack, which exploited vulnerable IoT devices to disrupt the website of a prominent security journalist, causing widespread service interruptions and significant economic losses [4].

Intrusion Detection Systems (IDS) are fundamental to network security defenses, as they are tasked with identifying and mitigating attack activities. Traditionally, IDS have relied on signature-based detection methods that compare network traffic against predefined attack patterns. Although effective for known threats, such approaches often fall short when confronted with novel or evolving attacks, primarily due to the inherent limitations of static signature databases [5]. In contrast, anomaly-based detection systems monitor for deviations from established normal behavior, providing a more flexible means of identifying previously unseen threats. Despite their promise, the increasing complexity of network environments continues to pose significant challenges to both detection strategies.

Recent advances in deep learning have opened new avenues for enhancing intrusion detection capabilities. In this paper, we introduce the TCNSE model—a hybrid architecture that combines a Temporal Convolutional Network (TCN) [6] with a Squeeze-and-Excitation (SE) module [7]. The TCN is adept at modeling long-range temporal dependencies via dilated convolutions, while the SE module refines feature representations by dynamically reweighting channel responses based on inter-channel relationships. This integration not only leverages temporal features more effectively but also enhances the quality of feature representations, thereby bolstering the model's ability to detect diverse and evolving network threats. Moreover, the parallel computation capabilities inherent in the TCN contribute to reduced computational complexity, making the TCNSE model particularly well-suited for deployment on resource-constrained IoT devices and enabling efficient real-time detection.The main contributions of this paper are as follows:

1. We propose an effective data preprocessing and feature selection method tailored for the CICIDS 2018 dataset. Our approach includes comprehensive data cleaning, a combination of various feature selection methods, and the application of hybrid sampling techniques to address class imbalance issues. Experimental results demonstrate that this method significantly enhances the detection performance of the model.

2. We design and implement the TCNSE network intrusion detection model. This model effectively captures the temporal dependencies of network traffic through the integration of the TCN and SE module, while also enhancing feature representation capabilities through the channel attention mechanism, leading to improved detection performance.

3. We conduct thorough experimental evaluations of the TCNSE model on the CIC-IDS 2018 dataset and compare it with various advanced intrusion detection models. The results show that the TCNSE model outperforms existing methods in key metrics such as accuracy, precision, recall, and F1 score.

4. We employ Focal Loss as the loss function to address class imbalance issues, dynamically adjusting sample weights and enhancing the model's ability to learn from difficult-to-classify samples.

RELATED WORK

Convolutional Neural Networks (CNNs) are renowned for their ability to extract spatial features through local receptive fields. In the context of intrusion detection, Kim et al. [8]transformed one-dimensional network data into two-dimensional images (grayscale or RGB), demonstrating that CNN-based approaches can achieve superior accuracy for DoS attack detection compared to RNN-based methods.

To capture both spatial and temporal features, Du et al. [9] proposed a hybrid CNN-LSTM model that leverages CNNs for spatial feature extraction and LSTMs for temporal dependency modeling. Altunay et al.[10] further validated the effectiveness of such hybrid approaches in industrial IoT scenarios, showing that combined CNN+LSTM models outperform standalone models in both binary and multiclass tasks.

Yao et al. [11]introduced a cross-layer feature fusion technique that integrates intermediate CNN features into an LSTM, yielding improved accuracy on datasets such as KDD Cup 99 and NSL-KDD. Similarly, Qazi et al. [12]developed a hybrid CNN-RNN system that achieved high precision and recall on the CIC-IDS 2018 dataset, though its reliance on oversampling to address data imbalance may increase the risk of overfitting.

Addressing low-sample challenges, He et al. [13] designed a Deep Feature-based Autoencoder Network (DFAE) by combining a pre-trained CNN with an autoencoder reconstruction module. This approach significantly enhanced detection metrics on datasets like CIC-IDS 2017 and USTC-TFC2016, despite introducing additional complexity. In the realm of vehicular networks, Kabilan et al. [14] proposed an unsupervised IDS for the CAN protocol by merging autoencoder-based feature extraction with Fuzzy C-Means clustering, though its performance on minority attack types remains suboptimal.

Alrawashdeh [15] presented an online anomaly detection system that combines Restricted Boltzmann Machines (RBM) with Deep Belief Networks (DBN), achieving 97.9% accuracy on the KDD CUP 99 dataset by effectively reducing feature dimensions through unsupervised learning. Filho et al. [16] introduced "Smart Detection," an ML-based system employing Random Forests for real-time DDoS detection, which bypasses the need for traffic redirection yet raises concerns about privacy and generalization.

More recent studies have explored Transformer-based approaches. Wang et al. [17] proposed RUIDS, a self-supervised IDS that leverages context reconstruction to improve robustness, albeit with increased computational complexity. Ullah et al.[18] developed IDS-INT, a transfer learning method that uses multi-head attention and SMOTE in conjunction with CNN-LSTM layers to effectively manage imbalanced network traffic, though its high resource consumption poses challenges for edge deployment.

Although these methods have achieved notable results in their respective application scenarios, they generally suffer from issues such as complex model architectures, high computational resource consumption, and insufficient real-time performance when deployed on edge devices. In response to these challenges, this paper proposes a lightweight intrusion detection system that, through efficient feature selection and model architecture optimization, significantly reduces model complexity and resource consumption while maintaining high detection performance, thereby providing a superior real-time protection solution for resource-constrained IoT environments.

METHODOLOGY, APPROACH AND PROPOSED MODEL

In the following sections, we outline our methodology and introduce the proposed intrusion detection model. We begin by discussing the datasets utilized in this study, followed by detailed data processing procedures. We then present the architecture of our intrusion detection system, as illustrated in Figure 1, which includes both the data processing workflow and the model training process.

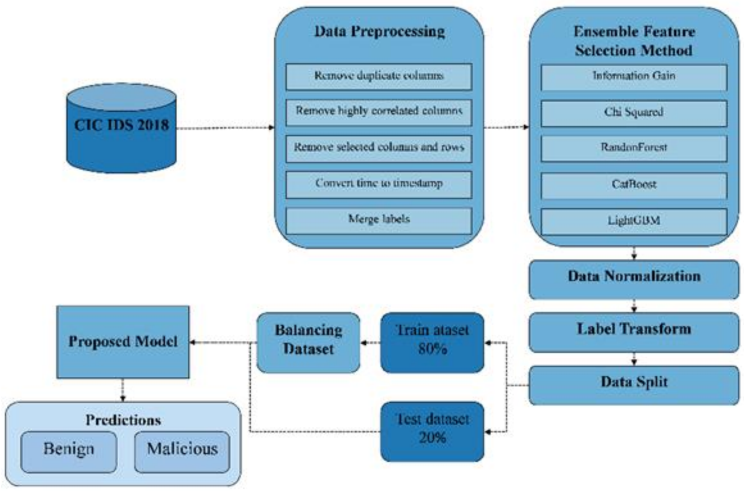


Figure 1 Architecture of the intrusion detection system

Intrusion detection datasets

The effectiveness of IDS relies on high-quality datasets for training and testing models. In recent years, several widely utilized network intrusion detection datasets have been employed to assess IDS performance, the most notable of which include KDD CUP 99, NSLKDD, CIC-IDS 2017, CIC-IDS 2018, and UNSW-NB15.

KDD CUP 99 and NSL-KDD

KDD CUP 99 is one of the earliest benchmark datasets in the field of network intrusion detection, generated from the DARPA 1998 dataset. However, this dataset presents numerous issues, including a significant amount of redundant data and an imbalanced class distribution [19]. Research indicates that approximately 78% of the records in the KDD CUP 99 dataset are duplicates, resulting in biased evaluations, particularly in detecting minority class attacks. About 98% of the training set records and 86% of the test set records can be completely recognized by all 21 classifiers (seven machine learning algorithms, each trained three times), suggesting that many records are overly simplistic for classifiers, leading to an inaccurate reflection of their capability in identifying complex and real-world attacks. To address these shortcomings, the NSL-KDD dataset was introduced, which improved upon KDD CUP 99 by removing redundant records and adjusting class distributions. Nevertheless, the NSL-KDD dataset has a limited range of attack types and does not encompass new attack vectors commonly found in modern networks, making models trained on it less effective against actual network threats.

UNSW-NB15

UNSW-NB15 represents a more contemporary dataset, generated using the IXIA PerfectStorm software in a 31-hour simulated environment. It encompasses nine categories of modern attacks and is based on a more realistic network setting [20]. Despite this, UNSWNB15 still relies on simulated data, which may result in discrepancies with real-world network traffic [21], thereby limiting the generalizability of models trained on it in practical applications.

CIC-IDS 2017 and CIC-IDS 2018

CIC-IDS 2017 and CIC-IDS 2018, published by the Canadian Institute for Cybersecurity (CIC), are modern datasets extensively used in network intrusion detection research. Both datasets are derived from genuine network traffic and encompass various complex attack types while aiming to mimic the characteristics of traffic in actual network environments during generation. The CIC-IDS 2017 dataset [22] contains five days of network traffic data, with approximately 19.7% classified as attack traffic. In contrast, the CIC-IDS 2018 dataset expands the network topology, covering a broader array of attack types and scenarios. This dataset includes 16,233,002 instances, with its volume and diversity of attack types making it an ideal choice for network intrusion detection in the current big data landscape[23].

Data processing

Without systematic data analysis procedures, achieving the desired protective effects in IDS is challenging; thus, the data processing phase is critical. Leevy et al. [24] highlighted in their review that there is a scarcity of literature detailing the data cleaning processes for intrusion detection datasets, which poses numerous challenges for researchers attempting to replicate experimental models due to the lack of clarity regarding original data handling methods. In light of this, this section provides a comprehensive description of the cleaning process for the CIC-IDS 2018 dataset. The CIC-IDS 2018 dataset consists of ten CSV files, with the distribution of attack types illustrated in Figure 2. Notably, the 02-20-2018.csv file contains four additional columns: Flow ID, Src IP, Src Port, and Dst IP, which need to be removed for uniform processing. The data processing steps are as follows:

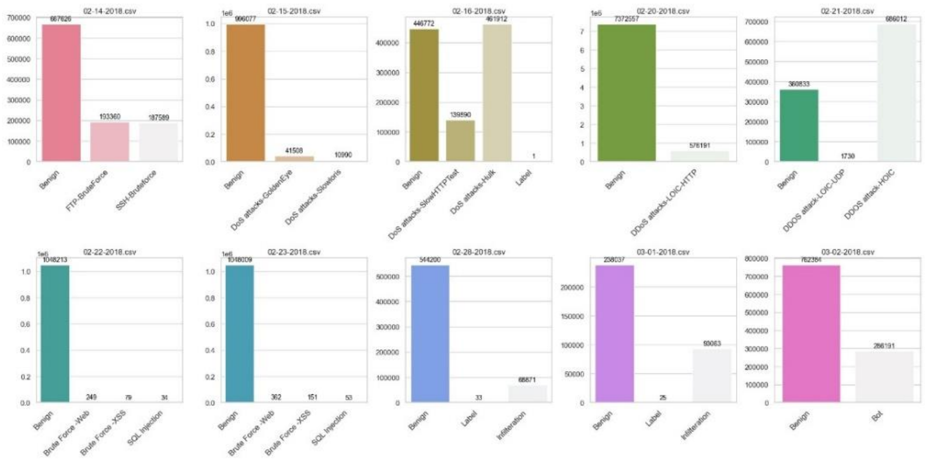


Figure 2 Distribution of attack types in each CSV file of the CIC-IDS 2018 dataset

Data preprocessing

• Remove rows containing , 'infinity' or other invalid values.

• Eliminate duplicate columns.

• Delete constant columns (i.e., columns where all row values are identical, which can be identified quickly by checking if the column variance equals zero).

• Discard highly correlated columns (with a threshold set at 0.95; if the correlation between two columns exceeds this threshold, retain only one).

• Remove specified columns and rows. The 'Protocol' column is omitted since the 'Dst Port' column sufficiently represents the protocol. Rows containing negative values in the 'Fwd Header Len', 'Flow Duration', and 'Flow IAT Min' columns are deleted, as these values should not be negative, indicating erroneous data. Similarly, the 'Init Fwd Win Byts' and 'Init Bwd Win Byts' columns are removed because nearly half of their data consists of negative values

• Convert timestamps (the original dataset's time column contains specific timestamps, while timeseries processing models require epoch time).

• Merge labels (i.e., categorize similar attacks into broader categories, as shown in Table 1).

With the completion of data preprocessing, the overall distribution of categories in the dataset is illustrated in Figure 3, containing 69 columns (including label columns) with a total of 155,722,254 records.

Table 1 Examples of attack type categorization in the CIC-IDS 2018 dataset

Generalized Attack Category	Specific Attack Types
Brute-force	SSH-Brute force
	FTP-Brute Force
Web attack	Brute Force -XSS
	Brute Force -XSS
	SQL Injection
DoS attack	DoS attacks-Hulk
	DoS attacks-Slow HTTP Test
	DoS attacks-Slow loris
	DoS attacks-Golden Eye
DDoS attack	DDOS attack-HOIC

	DDOS attack-LOIC-UDP
	DoS attacks-Slow loris
	DDoS attacks-LOIC-HTTP
Botnet	Bot
Infiltration	Infiltration

In addition to these common data preprocessing steps, it is noteworthy that due to the large size of the CIC-IDS 2018 dataset (approximately 6.41GB), importing it via Pandas may not choose appropriate data types for each column (for instance, if a column's maximum value is less than 255 and its minimum value exceeds 0, Pandas may default to the int type, although uint8 precision would suffice). This can consume substantial memory and slow down subsequent data processing, which can be mitigated through data type compression. The algorithm steps for this process are outlined in Figure 3, after which memory usage can be reduced by approximately 60%.

Figure 3 Reduce memory usage for DataFrame

```
Algorithm 1 Reduce memory usage for DataFrame
Input: DataFrame $props$
Output: Optimized DataFrame with reduced memory usage
for each $col$ in $props.columns$ do
    if $col$ is numeric then
        if values in $col$ can be represented as integers then
            if $col.min() \geq 0$ and $col.max() < 255$ then
                Downcast $col$ to $uint8$
            else if $col.min() \geq 0$ and $col.max() < 65535$ then
                Downcast $col$ to $uint16$        ▷ This logic can be extended to other types like uint32, int, etc., by adding similar conditions
            else
                Convert $col$ to a smaller floating-point type (e.g., $float32$)
            end if
        end if
    end if
end for
```

Feature selection

Feature selection involves selecting a subset of features from the original dataset as model inputs, which can avoid the curse of dimensionality and enhance generalization ability [25], while also reducing model training and inference time [26]. Studies have reported that feature selection methods significantly reduce training and testing times and improve intrusion detection rates. To avoid information loss, feature selection requires data to contain redundant or irrelevant features. Here, a multi-method feature selection algorithm (Figure 4) is adopted. This method integrates multiple approaches (e.g., statistical test-based and model-based feature selection), determining the final feature subset by scoring the importance of each feature across different methods and weighting the cumulative scores. Specifically, this study utilizes common feature selection methods such as information gain, chi-squared test, random forests, and CatBoost, as seen in

Table **2**.

```
Algorithm 2 Weighted feature selection through multiple methods
Data Initialization
$X \leftarrow$ data features
$y \leftarrow$ target variable
feature_scores ← initialize dictionary with zeros for each feature
Define Feature Selection Methods and Weights
methods ← [(method1, weight1, X, y), (method2, weight2, X, y), ...]
Process Each Method
for (method, weight, data, target) in methods do
    $scores \leftarrow$ apply method to data
    $max\_score \leftarrow \max(scores)$
    $normalized\_scores \leftarrow [score/max\_score \times 100$ for score in scores]
    for (score, feature) ∈ zip(normalized_scores, X.columns) do
        feature_scores[feature] ← feature_scores[feature] + score × weight
    end for
end for
Rank Features by Score
$sorted\_features \leftarrow$ sort features by scores in descending order
$selected\_features \leftarrow$ top k features from sorted_features
```

Figure 4 Weighted feature selection through multiple methods

Table 2 The involved feature selection algorithm

Algorithm	Parameter setting	API	Weight
Information Gain	default value	sklearn.feature selection.mutual info classif	1
Chi Squared	default value	sklearn.feature selection.chi2	1
RandomForest	n_estimators=5, random state=42, max depth=5, n_jobs=-1	sklearn.ensemble.RandomForestClassifier	1.2
CatBoost	Learning rate=0.01, iterations=200, depth=10, thread count=-1	catboost.CatBoostClassifier	1
LightGBM	learning rate=0.01	lightgbm.LGBMClassifier	1

Information Gain and the Chi-Squared Test are two commonly used statistical methods for evaluating how strongly a feature correlates with the target. Meanwhile, CatBoost, LightGBM, and Random Forest are tree-based ensemble algorithms, with the first two leveraging efficient base-learner training and the latter aggregating decisions via multiple decision trees.

To enhance the effectiveness of different methods, the importance scores from each method are normalized and assigned varying weights based on their reliability in the context (e.g., model-based feature selection may yield better results than statistical test-based methods and thus be given a higher weight). Ultimately, by aggregating the weighted scores, the top k critical features are selected. This multi-method ensemble and weighted scoring strategy for feature selection not only effectively addresses the high dimensionality issue but also ensures the selected features' generalization performance in the model. Table 3 shows the top 20 important features and their descriptions, ranked by Algorithm 2.

Table 3 CIC-IDS 2018 dataset: Top 20 features and their descriptions

Feature	Description
Dst Port	The destination port number of the packet.
Timestamp	The timestamp when the packet is captured.
Fwd IAT Tot	The total forward inter-arrival time (IAT) between packets.
Fwd IAT Max	The maximum forward IAT between packets.
ACK Flag Cnt	The count of ACK flags in the packet headers.
Init Bwd Win Byts	The initial backward window size in bytes.
Fwd IAT Mean	The average forward inter-arrival time (IAT) between packets.
Fwd Pkt Len Max	The maximum length of the forward packets.
TotLen Fwd Pkts	The total length of forward packets.
Fwd Header Len	The length of the forward packet header.
Fwd Seg Size Avg	The average segment size of forward packets.
Fwd Pkt Len Mean	The average length of forward packets.
Flow Pkts/s	The number of packets per second in the flow.
Fwd Seg Size Min	The minimum segment size of forward packets.
Flow IAT Mean	The mean inter-arrival time (IAT) of the flow.
Flow IAT Max	The maximum inter-arrival time (IAT) of the flow.
Fwd IAT Min	The minimum forward inter-arrival time (IAT) between packets.

Subflow Fwd Byts	The number of bytes in the forward subflow.
Bwd Seg Size Avg	The average segment size of backward packets.
Fwd Pkts/s	The number of forward packets per second.

Data normalization

Data normalization plays a crucial role in model performance and training efficiency. It helps align the distributions of different features within similar ranges, preventing issues such as numerical instability and gradient vanishing caused by large discrepancies in feature values. This, in turn, enhances the model's convergence speed and prediction accuracy. Furthermore, normalization improves the model's generalization ability, reduces the risk of overfitting, and boosts detection effectiveness for unknown attacks. The calculation process can be referenced using Equation 1, where the normalized value is denoted as x', and the maximum and minimum values of the dataset are represented as x_{max} and x_{min}, respectively.

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

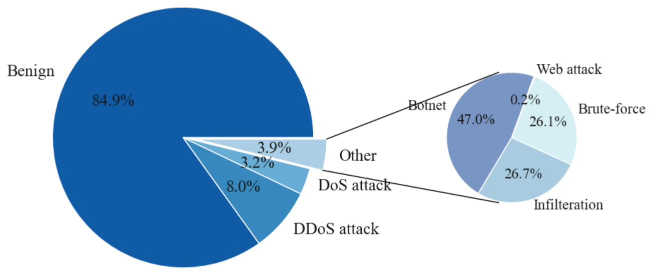
Data split

The dataset is divided into a training set comprising 80% of the data and a test set consisting of the remaining 20% using the train test split method from sklearn.

Data imbalance

Following the above processing, the overall distribution of the dataset is depicted in Figure 5, revealing that the Benign class accounts for 84.9%, indicating a highly imbalanced state. This imbalance typically leads to models becoming overly adapted to the majority class while neglecting the minority class, which may result in a tendency to classify most samples as the majority class [24]. For this dataset, if the model consistently classifies all input data as Benign, it would achieve an accuracy of 84.9% on the test set. In fields like network intrusion detection, this can cause high false positive rates and low detection rates, especially for rare attack types. By applying appropriate balancing techniques, the model's ability to recognize minority classes can be significantly improved. This not only reduces false positives and negatives but also enhances the model's generalization ability, making it more reliable in practical detection scenarios. In this study, a hybrid sampling approach is employed, which first reduces the number of majority samples through random undersampling(RUS), followed by oversampling the minority samples using SMOTE, ultimately achieving a balanced distribution across categories. The hybrid sampling method is applied only to the training set, while the test set remains unchanged.

Figure 5 Distribution of the whole CIC-IDS 2018 dataset



Label transformation

After the sampling process, labels are categorized into two classes: Benign and Malicious.

Proposed model

Figure 6 illustrates the architecture of the proposed intrusion detection model, referred to as the TCNSE network. It primarily comprises two key components: a TCN module and a SE module.

• TCN module: Utilizes causal and dilated convolutions to capture long-range dependencies in time series data, enabling efficient temporal feature extraction.

• SE module: Adapts the importance of each feature channel through a channel attention mechanism, enhancing the model's ability to express and improve accuracy on features.

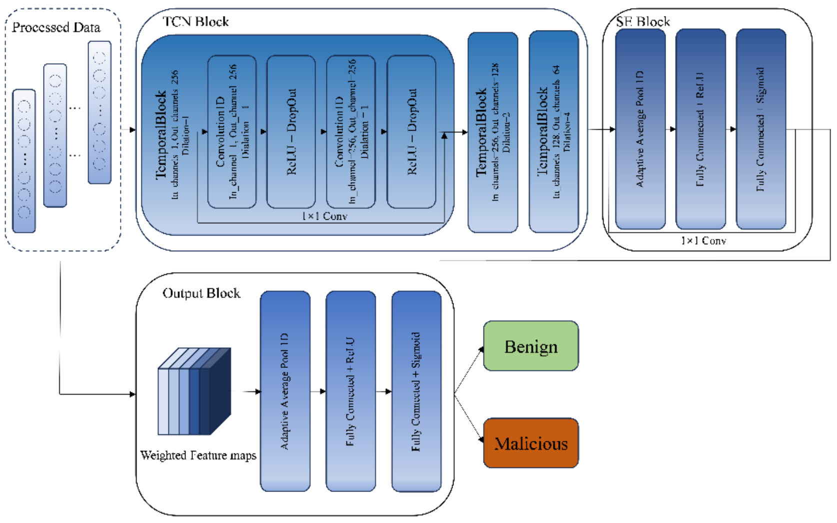

Processed Data
TCN Block
TemporalBlock In_channels 1, Out_channels 256 Dilation=1
Convolution1D In_channel 1, Out_channel 256 Dilation 1
ReLU – DropOut
Convolution1D In_channel=256, Out_channel=256 Dilation = 1
ReLU – DropOut
1×1 Conv
TemporalBlock In_channels=256, Out_channels=128 Dilation=2
TemporalBlock In_channels 128, Out_channels 64 Dilation=4
SE Block
Adaptive Average Pool 1D
Fully Connected + ReLU
Fully Connected + Sigmoid
1×1 Conv
Output Block
Weighted Feature maps
Adaptive Average Pool 1D
Fully Connected + ReLU
Fully Connected + Sigmoid
Benign
Malicious

Figure 6 Architecture of proposed intrusion detection model

Temporal convolutional network

When processing time series data, traditional algorithms like RNNs and LSTMs sequentially handle time points, integrating information from previous time points into subsequent computations. This sequential dependency can hinder model responsiveness and efficiency, particularly in resource-constrained environments like IoT devices. The complexity and memory consumption of these models are critical; an overly complex model can lead to extended inference times, affecting responsiveness. Unlike RNNs, convolutional operations allow for parallel computation, greatly increasing processing speed. Additionally, these traditional algorithms consume substantial memory to store unit gate outputs over long sequences, which is why the TCN module is employed instead of RNNs in this model.

The main characteristics of the TCN module include causal convolutions, dilated convolutions, and residual connections. Causal convolutions are designed to ensure that the output at time t only relies on the current and past inputs, thereby preventing future information leakage. In standard convolution, padding is typically applied to both ends of the input sequence to maintain equal output and input sequence lengths. However, causal convolutions only apply padding to the left side (i.e., in the “past” direction), as shown in Equation 2, ensuring that each output does not involve future input data. Given an ordered input sequence $\mathrm{x} = [\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{T}]$, with kernel size k and weights $\mathrm{w} = [\mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{k}]$, the output at position t in standard one-dimensional convolution is defined by Equation 3:

$$p_l = (k-1) \times d \tag{2}$$

$$y_t = \sum_{i=1}^{k} w_i \cdot x_{t+i-\frac{k+1}{2}} \tag{3}$$

In causal convolutions, the index is adjusted as follows 4:

$$y_t = \sum_{i=1}^{k} w_i \cdot x_{t-i+1} \tag{4}$$

This adjustment ensures that the output y_t depends solely on $\mathrm{x}_\mathrm{t}, \mathrm{x}_{\mathrm{t}-1}, \dots, \mathrm{x}_{\mathrm{t}-\mathrm{k}+1}$, which encompasses the current and prior inputs.

Dilated convolutions introduce gaps between kernel elements, allowing convolution operations to span a broader input range, thus enlarging the receptive field without increasing kernel size or computation, depicted in

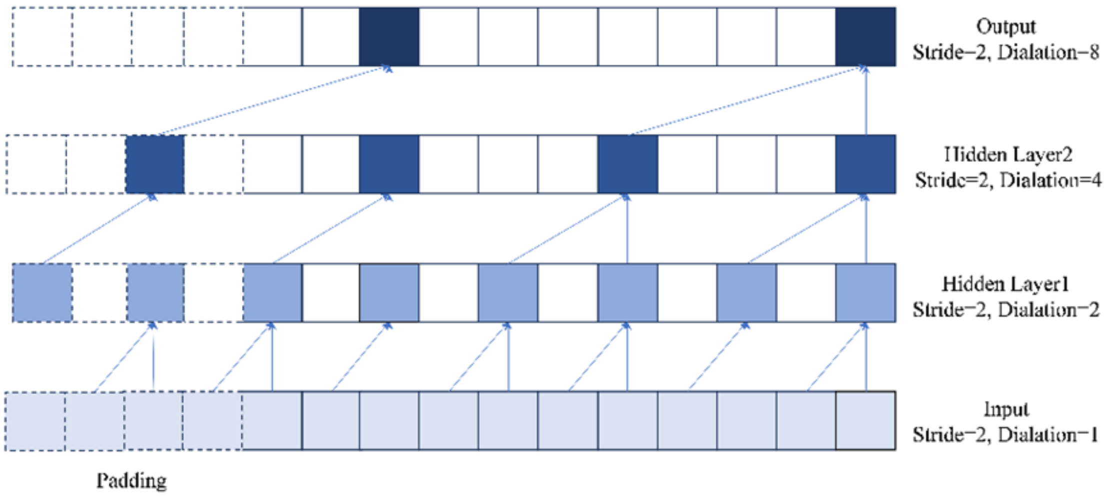


Figure **7**. This enables the model to capture distant relationships among features more effectively. For an input sequence x with kernel size k and dilation factor d, the output at position t is computed as follows 5:

$$y_t = \sum_{i=1}^{k} w_i \cdot x_{t+i\cdot d-d} \tag{5}$$

where:

• w represents the convolution kernel weights.

• d is the dilation factor, which determines the spacing between kernel elements and typically grows exponentially (e.g., [1, 2, 4, ...]).

When d = 1, the dilated convolution reverts to standard convolution.

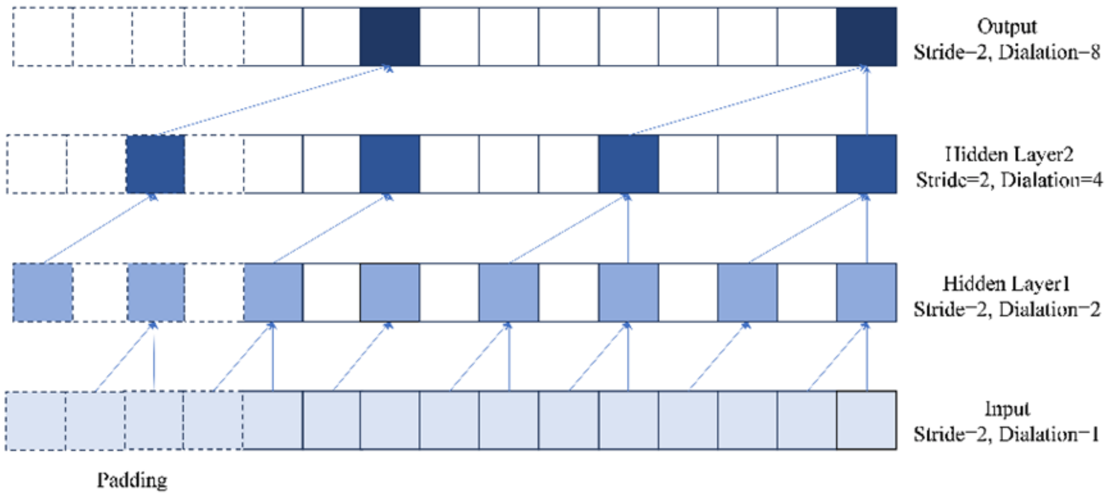


Figure 7 Architecture of proposed intrusion detection model

In deep learning models, as the number of layers increases, the gradients during backpropagation may diminish or explode, complicating network training. Residual connections introduce a direct path bypassing several layers, allowing information to flow to deeper layers. This structure fundamentally alters the transformation function from H(x) to H(x) = F(x) + x, where:

• F(x) represents the learnable residual component.

• x is the input, directly added to the output through the residual connection.

This design allows the network to focus on learning the residual function F(x) rather than the direct mapping H(x). Consequently, even if some layers in a deep network fail to learn effective features (i.e., if F(x) approaches 0), the residual connection ensures that information x can still propagate through the model, thus maintaining performance as the model depth increases.

Squeeze-and-excitation block

Traditional convolutional operations mainly capture spatial features, often overlooking the relationships between different channels. To address this shortcoming, the SE (Squeeze-and-Excitation) module introduces an attention mechanism that dynamically reassigns weights to feature channels, enabling the network to focus on the most informative ones. As illustrated in Figure 8, the SE module operates in three stages: Squeeze, Excitation, and Scale. In the Squeeze step, global average pooling is applied to each channel to generate a compact vector summarizing the global channel statistics. Next, during the Excitation step, fully connected layers along with non-linear activations are used to learn channel-specific weights—this effectively acts as an attention mechanism by emphasizing the channels that are most relevant. Finally, the Scale step involves reweighting the original feature maps with these learned weights, thus enhancing critical channels while suppressing less significant ones and ultimately improving the network's representational capacity.

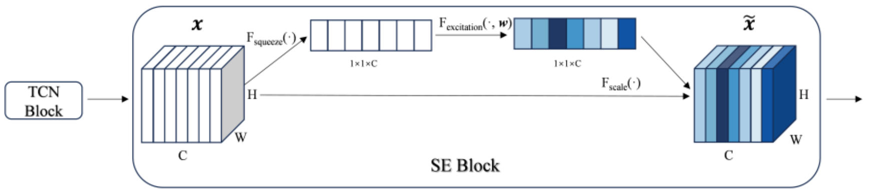


Figure 8 Detailed Interaction between the TCN Block and the SE Block

Loss function and optimizer

Given the extreme class imbalance in the CICIDS 2018 dataset, the Binary Focal Loss [27] is utilized to enhance the model's focus on difficult to- classify samples during training. In scenarios with class imbalance or a preponderance of easily classified samples, standard cross-entropy loss often leads models to concentrate excessively on the majority class, overlooking the minority class or challenging samples. Focal Loss dynamically adjusts sample weights, effectively enhancing the model's learning capabilities for minority classes and difficult samples, thereby improving overall performance. The calculation for Focal Loss is provided in Equation 6:

$$FL(p_t) = -\alpha_t(1 - p_t)^{\gamma}\log(p_t) \tag{6}$$

where:

• p_t denotes the predicted probability for the true class.

• α_t is the weight applied to the true class, aiming to address class imbalance.

• γ is a focusing parameter, determining the level of focus on hard-to-classify samples. A larger γ value places greater emphasis on these samples.

Experimental analysis and results

Performance metrics

In addition to the model's accuracy on the test set, there are many metrics used to assess the performance of an intrusion detection model. Here, we use the 10 most commonly used metrics in the field of intrusion detection[28], which are: Accuracy, Recall (True Positive Rate), Precision, F1-Score, False Positive Rate (FPR), ROC Curve and AUC, True Negative Rate (TNR), False Negative Rate (FNR), Confusion Matrix, Training Time

Model training and performance analysis

This section details the training process of the intrusion detection model, as well as the evaluation and comparison of model metrics. Table 4 shows the hardware configurations and software versions of the training platform, and

Table 5 lists the optimal hyperparameters obtained through experimental comparisons.

Table 4 Platform specifications for model training

Component	Specification
CPU	Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz
Graphics card	3080 10GB

Memory	40G
Python version	3.8
Pytorch version	2
Cuda version	11.8

Table 5 Hyperparameters for the model training

Hyperparameter	Value
Batch size	2048
Loss function	BinaryFocalLoss
Number of epochs	50
Optimizer	AdamW
Weight decay	1.00E-05
Learning rate	0.001
Learning rate decay	CosineAnnealingLR, T max=50
Temporal convolutional network structure	[128, 64, 32]

In this study, all features in the CIC-IDS 2018 dataset were ranked according to importance using an integrated feature selection algorithm, and the model's performance was evaluated with different numbers of features, as shown in Table 6. Experimental results indicate that when the number of features is set to 20, the model performs best across all performance metrics, achieving an accuracy of 98.4%, a recall of 98.9%, precision and F1-score both at 99.1%, the lowest false positive rate of 3.3%, the highest true negative rate of 96.7%, and a training time per epoch of only 74.4 seconds. This result clearly demonstrates that appropriately reducing the number of features not only maintains the model's classification ability but also enhances the overall performance in multiple aspects.

Table 6 Performance metrics across different numbers of features

Number of Features	Accuracy (%)	Recall (%)	Precision (%)	F1-score (%)	False Positive Rate (%)	True Negative Rate (%)	Training Time(s)
15	97.8	97.5	98.2	97.8	4.5	95.5	**60.1**
20	**98.4**	**98.9**	**99.2**	**99.1**	**3.3**	**96.7**	74.4
30	98.2	98.6	99.1	99.0	3.6	96.4	111.6
40	98.3	98.7	99.1	98.2	3.7	96.3	131.4
50	98.3	98.7	98.8	98.7	3.7	96.3	153.4
60	98.1	98.6	99.4	98.6	4.7	95.3	162.5
68(all features)	98.1	98.7	98.9	98.9	4.8	95.2	184.9

As the number of features increases, although the model's precision and F1-score remain high, other metrics like the false positive rate increase, the true negative rate slightly decreases, and the training time extends significantly. For example, when the number of features increases to 68, the training time nearly doubles to 184.9 seconds, while the false positive rate rises to 4.8%, and the true negative rate drops to 95.2%. This phenomenon can be explained by the curse of dimensionality [29]. High-dimensional feature spaces cause data to become sparse, making it difficult for the model to effectively capture the intrinsic structure of the data, leading to decreased generalization ability. Moreover, more features mean more parameters to learn, increasing computational complexity and extending training time. Especially in resource-constrained IoT device environments, such computational overhead negatively impacts system response time and user experience.

Feature selection played a critical role in this process. By comprehensively utilizing multiple feature selection algorithms to select the 20 most discriminative features, the model maintains high classification accuracy while significantly reducing computational overhead and training time. This not only enhances the model's real-time responsiveness but also reduces data storage and transmission costs, improving the system's feasibility and efficiency in practical applications. Additionally, an appropriate number of features helps reduce interference from redundant and noisy features, enhancing the model's robustness and stability, thereby improving its generalization performance on unseen data.

Figure 9 shows the changes in various metrics during model training with learning rates of 0.01 and 0.001, respectively. It can be seen that when the learning rate is 0.001, the model's Precision, FPR, and TNR are better than those with a learning rate of 0.01. This may be attributed to the lower learning rate making parameter updates more stable during training, allowing the model to more closely approach the optimal solution of the loss function [30], and reducing parameter oscillations or overfitting caused by an excessive learning rate. In contrast, a learning rate of 0.01 may cause the model to skip over optimal regions during training, failing to fully learn the deep features of the data, thereby affecting the model's classification performance.

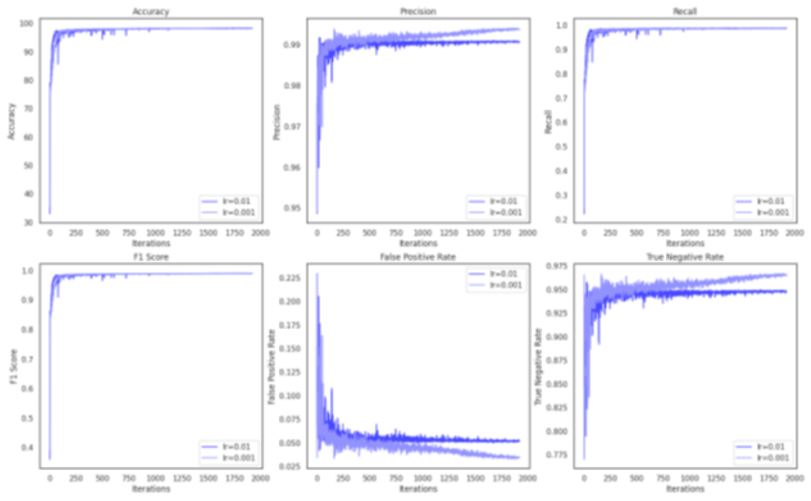


Figure 9 Comparison of metrics across different learning rates

From the trend curves of various performance metrics over training iterations, depicted in Figure 10, it is evident that although different batch sizes exhibit slightly different convergence speeds during the early stages, the final performance is optimal when the batch size is set to 2048, achieving higher accuracy, precision and TNR, and a lower FPR. However, when the batch size is further increased to 4096, all metrics noticeably deteriorate. This phenomenon may be attributed to the tendency of large-batch training to converge to sharp minima that weaken generalization ability, as noted by Zhang et al. [31], where excessively large batch sizes lead the model to converge to inferior minima, ultimately degrading the final detection performance.

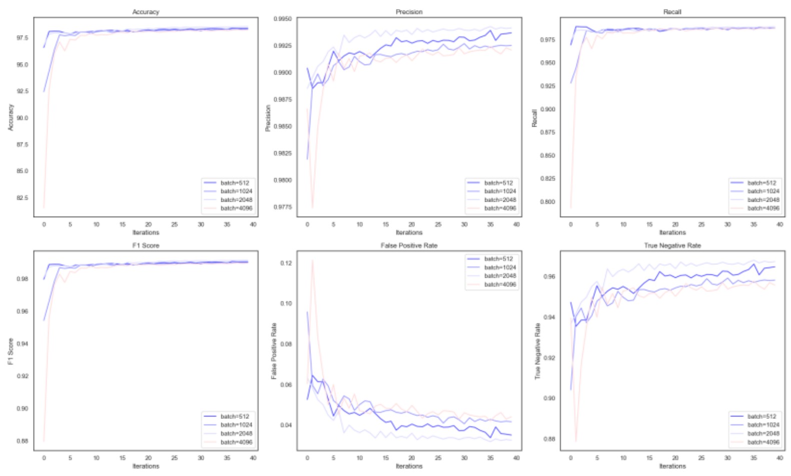


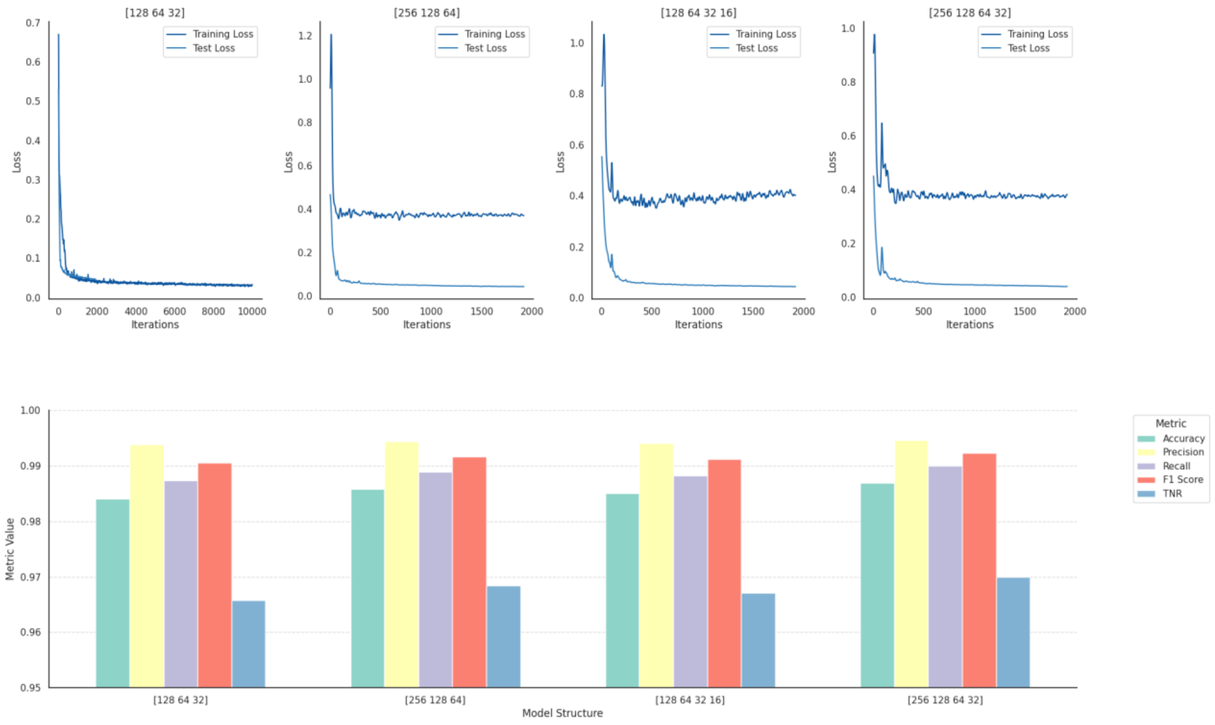
Figure 10 Comparison of metrics across different batch sizes

Table 7 clearly demonstrates that while both random under-sampling and SMOTE individually enhance model performance, the hybrid approach (RUS+SMOTE) outperforms both. It achieves the highest accuracy, recall, precision, and F1-score, along with the lowest false positive rate, confirming the effectiveness of combining both methods to address class imbalance.

Table 7 Comparison of performance metrics for different imbalance handling techniques

Method	Accuracy (%)	Recall (%)	Precision (%)	F1-score (%)	FPR (%)	TNR (%)
Random Undersampling	97.8	97.5	98.3	97.7	4.5	95.5
SMOTE Alone	97.9	98.0	98.1	98.0	4.2	95.8
Hybrid Sampling(RUS+SMOTE)	98.4	98.9	99.0	99.1	3.3	96.7

Figure 11 compares the training error, test error, and various performance metrics under different temporal convolutional block structures of the TCNSE model, with a learning rate of 0.001 and using 20 selected features. Table 8 summarizes the key performance metrics and training times for these structures. As model complexity increases, Accuracy, Precision, Recall, F1 Score, and TNR all improve. Notably, the [256, 128, 64, 32] structure achieves the best performance, with 98.6% accuracy, 99.4% precision, 98.9% recall, 99.2% F1 score, and 96.9% TNR. This enhancement is mainly due to its deeper architecture and greater parameter capacity, enabling the capture of more complex features and temporal dependencies. The introduction of residual connections also mitigates gradient vanishing, ensuring effective information flow. However, this performance gain comes at the cost of increased training time: one epoch for [256, 128, 64, 32] takes 89.6 seconds, nearly twice that of the



[128, 64, 32] structure at 46.1 seconds. This trade-off, driven by the higher number of layers and parameters, may affect deployment on IoT devices with limited computational resources.

Figure 11 Comparison of metrics across different model structures

Table 8 Summary of different model structure comparison

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	FPR (%)	TNR (%)	Training Time (s)
[128 64 32]	98.4	99.3	98.7	99.1	3.3	96.7	46.1

[256 128 64]	98.5	99.4	98.9	99.1	3.2	96.8	83.1
[128 64 32 16]	98.5	99.4	98.8	99.1	3.3	96.7	51.6
[256 128 64 32]	98.6	99.4	98.9	99.2	3.1	96.9	89.6

Therefore, in model design, a balance needs to be struck between performance improvement and computational overhead. Although more complex models can provide better performance metrics, training time and inference efficiency are equally critical in practical applications. For scenarios requiring real-time response, appropriately reducing model complexity and choosing a lighter structure like [128, 64, 32] may be a more practical choice.

Figure 12 shows the confusion matrix on the test set under the conditions of using the model structure [128, 64, 32], the optimal learning rate, and the selected number of features. Figure 13 presents the ROC curves for detecting various types of attacks. Table 9 illustrate the detection performance of our model across various attack types. Notably, the Infiltration attack type constitutes only about 1% of the overall dataset; nevertheless, the model achieves a classification accuracy of 98.2% and a corresponding true positive rate of 96% for these scarce samples. This attack category is, however, the most prone to false positives, an issue that appears to be currently unavoidable. In contrast, for more prevalent attack types such as DDoS, the model attains a true positive rate of 100% and an accuracy of 98.8%, indicating that false positives and false negatives are virtually eliminated

Table 9 Performance metrics for different attack types

Attack Type	Accuracy	TPR
DDoS Attack	98.8%	100.0%
DoS Attack	98.8%	100.0%
Botnet	98.7%	100.0%
Infiltration	98.2%	96.0%
Brute-force	98.7%	100.0%
Web Attack	98.7%	99.0%

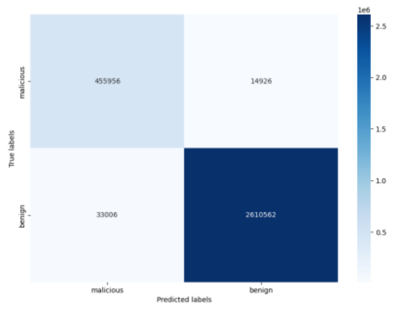


Figure 12 Confusion matrix

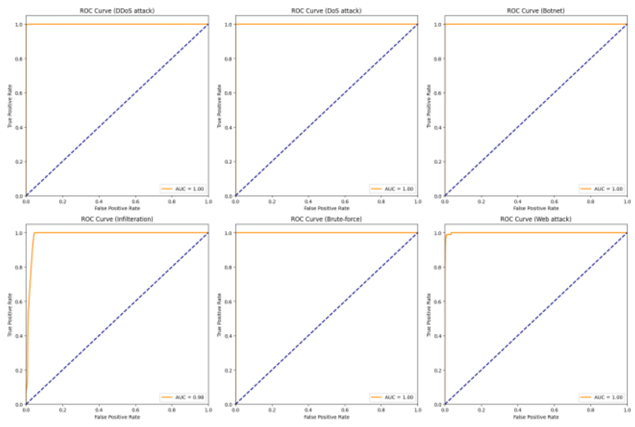


Figure 13 ROC curves for different attack types

Performance comparison with existing IDS models

In this section, we compare the proposed TCNSE model with traditional machine learning methods and several advanced deep learning intrusion detection models. Table 10 presents the performance of different research models on key metrics such as Accuracy, Precision, Recall, AUC, and F1 Score.

From the table, it is evident that the TCNSE model performs excellently on all evaluation metrics. Specifically, the TCNSE model achieves an accuracy of 98.4%, a recall of 98.9%, an AUC of 97.8%, and an F1 score of 99.1%. These results are significantly better than traditional machine learning models, such as the Random Forest model proposed by Leevy et al [37], which has an AUC of 95.5% and an F1 score of 93.2%, and the Naive Bayes model with an AUC of only 55.4% and an F1 score of 31.4%.

Compared with other deep learning models, the TCNSE model shows remarkable advantages. For example, Ferrag et al. [32] used RNN and deep autoencoder models, achieving an accuracy of 97.4% and an AUC of 98.2%; Gamage et al. [33] implemented a deep feed-forward neural network that achieved an accuracy of 98.4%, a recall of 98.0%, and an AUC of 98.3%; Lin et al. [34] obtained an accuracy of 96.2% and a recall of 96.0% with their LSTM model. In contrast, the TCNSE model improves on all key metrics, particularly excelling in recall and F1 score, reaching 99.4% and 99.1% respectively, demonstrating significant advantages in identifying actual intrusion events and maintaining a balance between precision and recall.

Table 10 Comparison of proposed model with existing models

Research	Model	Metrics
[32]	RNN, Deep Autoencoder	Accuracy=97.4%, AUC=98.2%
[33]	Deep Feed-forward Neural Network	Accuracy=98.4%, Recall=98.0%, AUC=98.3%
[34]	LSTM	Accuracy=96.2%, Recall=96.0%
[35]	Deep Autoencoder	Accuracy=97.9%, Recall=98.0%
[8]	CNN	Accuracy=91.5%, AUC=97.1%
[36]	CNN + RNN	Accuracy=97.7%, Recall=96.3%
[37]	CatBoost	AUC=95.6%, F1=93.6%
[37]	Decision Tree	AUC=91.2%, F1=88.7%
[37]	LightGBM	AUC=95.2%, F1=93.0%
[37]	Naive Bayes	AUC=55.4%, F1=31.4%
[37]	Random Forest	AUC=95.5%, F1=93.2%
Proposed Model	**TCNSE**	**Accuracy=98.4%, Precision=99.2%, Recall=98.9%,**

		AUC=97.8%, F1 Score=99.1%

Edge deployment evaluation: resource consumption and Real-Time performance analysis

To evaluate the resource consumption of our model on edge devices, we deployed it on two representative platforms—Raspberry Pi 5 and Jetson Xavier NX and provided detailed hardware specifications for both. Table 11 presents the hardware specifications of the two devices. The reason for selecting these two devices is that the Jetson Xavier NX supports GPU-accelerated inference, whereas the Raspberry Pi 5 does not.In our experiments, these devices served as target machines while an independent host simulated network attacks to recreate realistic security threat scenarios.

Table 11 Platform specifications

Attribute	Raspberry Pi 5	Jetson Xavier NX
CPU	Quad-core Cortex-A76 (ARM v8) 64-bit SoC @ 2.4GHz	Six-core (2x Carmel ARM v8 64-bit CPU)
GPU	–	384 NVIDIA CUDA Cores + 48 Tensor Cores
Memory	4GB LPDDR4X @ 4267MHz	6GB LPDDR4 @ 1600MHz
Operating System	Debian	Ubuntu

Unlike the training and testing phases where preprocessed data is directly used, the deployment environment requires additional steps, including realtime network traffic capture, feature extraction, and data normalization, to convert raw network data into the format required by the model. The overall workflow is illustrated in Figure 14. First, network packets are captured using Scapy, a Python-based tool capable of sending, sniffing, analyzing, and forging packets. Subsequently, traffic features are extracted via the Python version of CICFlowMeter—the same tool originally employed in generating the CIC-IDS 2018 dataset. After feature extraction, the data is further processed to remove invalid values and normalized before being fed into the model for real-time inference. Upon detecting malicious traffic, the system employs iptables to both block the attack and issue alerts to the user.

In our experiment, a host machine simulated DDoS attack against edge devices running our detection program. Figure 15 and Figure 16 illustrate the resource consumption on the Raspberry Pi 5 and Jetson Xavier NX, respectively, while Table 12 compares our model with other lightweight IDS models. The reported CPU, memory, and CUDA metrics represent average values sampled once per second, and the detection latency denotes the total delay from raw data capture through the entire processing pipeline (as shown on the right side of the Figure 14), rather than just the inference delay.

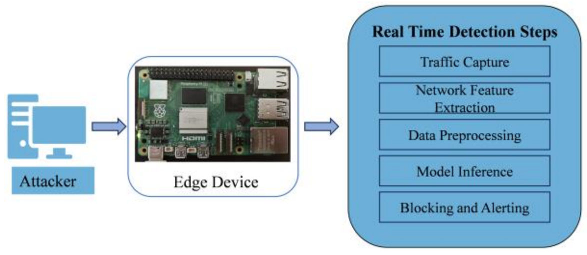


Figure 14 Workflow for Real-Time intrusion detection on edge devices

Figure 15 Raspberry Pi 5 Resource Usage

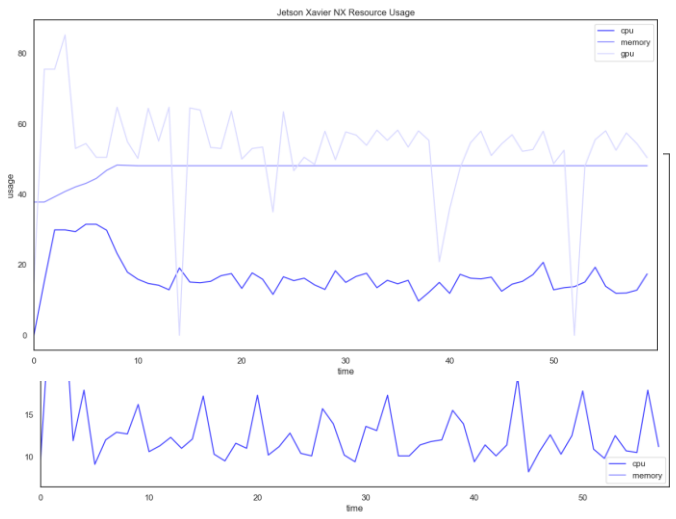


Figure 16 Jetson Xavier NX Resource Usage

Regarding CPU and memory usage, the TCNSE model achieves an average CPU utilization of only 14% and a memory usage of about 44% on Raspberry Pi 5—substantially lower than some models that report up to 73.3% CPU usage on Raspberry Pi 4B, demonstrating its lightweight advantage in resource constrained environments. On the Jetson Xavier NX, GPU acceleration causes the CPU usage to rise slightly to 18%, with memory usage reaching approximately 48%.

The detection program accumulates a certain amount of network flow data to compute statistical features—such as the maximum inter-arrival time of forward packets, the mean inter-arrival time of the flow, and the packet rate—which are essential for accurately reflecting traffic dynamics and improving intrusion detection performance. Although these data are temporarily stored in memory (resulting in a somewhat higher memory footprint compared to, for example, [41]), the program incorporates timeout and data-volume thresholds to process and clear the data once they reach a predetermined limit, ensuring that memory does not accumulate indefinitely.

Table 12 Comparison of proposed model with existing models on edge devices

Research	Dataset	Metrics on Model Training	Edge Device	Energy Consumption on Edge Device
[38]	CICIoT2023	-	Raspberry Pi 3	Energy consumption: 6.04W, Latency: 6.76s
[39]	UNSW-NB15	Training time: 720s, GPU Memory Usage: 9.8GB, CPU Usage: 80%	-	-
[40]	CICIDS2017	-	Raspberry Pi 4B	CPU Usage: 73.3%, Memory Usage: 51.9%
[41]	CICIDS2017	Model size: 14.14MB	Raspberry Pi 2B	CPU Usage: 16%, Memory Usage: 14%
[42]	CICIDS2017	-	Raspberry Pi 4B	CPU Usage: 36%, Memory Usage: 10%
Proposed Model	CICIDS2018	Training time: 46s, GPU Memory Usage: 1041MB, CPU Usage: 45%, Model size: 79KB	Raspberry Pi 5	CPU Usage: 14%, Memory: 44%, Latency: 45ms
			Jetson Xavier NX	CPU Usage: 18%, Memory: 64%, Latency: 21ms, Energy Consumption: 4.4W

In terms of real-time detection, inference latency is a critical performance metric. The TCNSE model achieves an average detection latency of 45 milliseconds on Raspberry Pi 5, compared to the CNN-LSTM model in [38], which exhibits a latency of up to 6.76 seconds on Raspberry Pi 3. On the Jetson Xavier NX, GPU acceleration further reduces the TCNSE inference latency to 21 milliseconds.

Finally, energy efficiency is particularly important for IoT and edge devices. On the Jetson Xavier NX, the TCNSE model consumes approximately 4.4W, whereas some competing models on Raspberry Pi 3 consume up to 6.04W. The compact model size of only 79KB and the carefully designed feature selection strategy contribute to reduced computational and memory overhead, leading to significant overall energy savings.

CONCLUSION AND FUTURE WORK

We propose TCNSE, a lightweight intrusion detection model that integrates Temporal Convolutional Networks (TCN) with Squeeze-and-Excitation (SE) modules, utilizing only 20 key features from the CIC-IDS 2018 dataset. Our approach—featuring comprehensive data cleaning, an integrated weighted ensemble feature selection method, and effective imbalance handling—reduces the feature space and computational complexity, thereby shortening both training and inference times without compromising performance. By fully leveraging temporal dependencies and inter-channel relationships, TCNSE robustly detects complex intrusion behaviors. Experimental results show that the model achieves 98.4% accuracy, 99.40% precision, 98.70% recall, and a 99.10% F1 score, underscoring the critical balance between performance and efficiency.

Looking ahead, we plan to enhance TCNSE's resilience against adversarial attacks by incorporating techniques such as FGSM, PGD, and robust loss functions, as well as exploring GAN-generated adversarial examples to counter sophisticated evasion strategies. Future work will also involve applying advanced optimization algorithms (e.g., Particle Swarm Optimization) to further refine feature selection, validating the model on additional intrusion detection datasets, and exploring transfer and ensemble learning methods to address data imbalance and improve detection of minority attacks. Finally, we aim to optimize real-world deployment using containerization technologies like Docker to develop an efficient, stable, and real-time intrusion detection system.

DATA AVAILABILITY

The data underlying this article are available at https: //www.unb.ca/cic/datasets/ids-2018.html. The source code for this research is available for download at: https://github.com/Sake169/TCNSE-IDS.

DECLARATION OF CONFLICTING INTERESTS

The authors declared no potential conflicts of interest with respect to the research, author-ship, and publication of this article.

FUNDING

The authors received no financial support for the research.

REFERENCES

[1] Hesamodin Mohammadian, Ali A Ghorbani, and Arash Habibi Lashkari. A gradient-based approach for adversarial attack on deep learning-based network intrusion detection systems. Applied Soft Computing, 137:110173, 2023.

[2] Internet Crime Complaint Center (IC3). 2023 Internet Crime Report, 2024. Accessed: 11-Oct-2024.

[3] Telenor IoT. AIoT: How Artificial Intelligence is Transforming the Internet of Things, 2024. Accessed: 11-Oct-2024.

[4] Joel Margolis, Tae Tom Oh, Suyash Jadhav, Young Ho Kim, and Jeong Neyo Kim. An in-depth analysis of the mirai botnet. In 2017 International Conference on Software Security and Assurance (ICSSA), pages 6–12. IEEE, 2017.

[5] Dingyu Shou, Chao Li, Zhen Wang, Song Cheng, Xiaobo Hu, Kai Zhang, Mi Wen, and Yong Wang. An intrusion detection method based on attention mechanism to improve cnn-bilstm model. The Computer Journal, 67(5):1851–1865, 2024.

[6] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271, 2018.

[7] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7132– 7141, 2018. Jiyeon Kim, Jiwon Kim, Hyunjung

Kim, Minsun Shim, and Eunjung Choi. Cnn-based network intrusion detection against denial-of-service attacks. Electronics, 9(6):916, 2020.

[8] Jiyeon Kim, Jiwon Kim, Hyunjung Kim, Minsun Shim, and Eunjung Choi. Cnn-based network intrusion detection against denial-of-service attacks. Electronics, 9(6):916, 2020

[9] Jiawei Du, Kai Yang, Yanjing Hu, and Lingjie Jiang. Nids-cnnlstm: Network intrusion detection classification model based on deep learning. IEEE Access, 11:24808–24821, 2023.

[10] Hakan Can Altunay and Zafer Albayrak. A hybrid cnn+ lstm-based intrusion detection system for industrial iot networks. Engineering Science and Technology, an International Journal, 38:101322, 2023.

[11] Ruizhe Yao, Ning Wang, Zhihui Liu, Peng Chen, and Xianjun Sheng. Intrusion detection system in the advanced metering infrastructure: a cross-layer featurefusion cnn-lstm-based approach. Sensors, 21(2):626, 2021.

[12] Emad Ul Haq Qazi, Muhammad Hamza Faheem, and Tanveer Zia. Hdlnids: hybrid deep-learning-based network intrusion detection system. Applied Sciences, 13(8):4921, 2023.

[13] Mingshu He, Xiaojuan Wang, Junhua Zhou, Yuanyuan Xi, Lei Jin, and Xinlei Wang. Deep-feature-based autoencoder network for few-shot malicious traffic detection. Security and Communication Networks, 2021(1):6659022, 2021.

[14] N Kabilan, Vinayakumar Ravi, and V Sowmya. Unsupervised intrusion detection system for in-vehicle communication networks. Journal of Safety Science and Resilience, 5(2):119–129, 2024.

[15] Khaled Alrawashdeh and Carla Purdy. Toward an online anomaly intrusion detection system based on deep learning. In 2016 15th IEEE international conference on machine learning and applications (ICMLA), pages 195–200. IEEE, 2016.

[16] Francisco Sales de Lima Filho, Frederico AF Silveira, Agostinho de Medeiros Brito Junior, Genoveva Vargas- Solar, and Luiz F Silveira. Smart detection: an online approach for dos/ddos attack detection using machine learning. Security and Communication Networks, 2019(1):1574749, 2019.

[17] Wei Wang, Songlei Jian, Yusong Tan, Qingbo Wu, and Chenlin Huang. Robust unsupervised network intrusion detection with self-supervised masked context reconstruction. Computers & Security, 128:103131, 2023.

[18] Farhan Ullah, Shamsher Ullah, Gautam Srivastava, and Jerry Chun-Wei Lin. Ids-int: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. Digital Communications and Networks, 10(1):190–204, 2024.

[19] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In 2009 IEEE symposium on computational intelligence for security and defense applications, pages 1–6. Ieee, 2009.

[20] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of networkbased intrusion detection data sets. Computers & security, 86:147–167, 2019.

[21] Nour Moustafa and Jill Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. Information Security Journal: A Global Perspective, 25(1-3):18–31, 2016.

[22] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. International Journal of Engineering & Technology, 7(3.24):479–482, 2018.

[23] Ankit Thakkar and Ritika Lohiya. A review of the advancement in intrusion detection datasets. Procedia Computer Science, 167:636–645, 2020.

[24] Joffrey L Leevy, Taghi M Khoshgoftaar, Richard A Bauder, and Naeem Seliya. A survey on addressing high-class imbalance in big data. Journal of Big Data, 5(1):1–30, 2018.

[25] Mairead L Bermingham, Ricardo Pong-Wong, Athina Spiliopoulou, Caroline Hayward, Igor Rudan, Harry Campbell, Alan F Wright, James F Wilson, Felix Agakov, Pau Navarro, et al. Application of highdimensional feature selection: evaluation for genomic prediction in man. Scientific reports, 5(1):10312, 2015.

[26] Ammar Alazab, Michael Hobbs, Jemal Abawajy, and Moutaz Alazab. Using feature selection for intrusion detection system. In 2012 international symposium on communications and information technologies (ISCIT), pages 296–301. IEEE, 2012.

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. IEEE Transactions on Pattern Analysis & Machine Intelligence, 42(02):318–327, 2020.

[28] Vitor Gabriel da Silva Ruffo, Daniel Matheus Brand˜ao Lent, Mateus Komarchesqui, Vin´ıcius Ferreira Schiavon, Marcos Vinicius Oliveira de Assis, Luiz Fernando Carvalho, and Mario Lemes Proen¸ca Jr. Anomaly and intrusion detection using deep learning for softwaredefined networks: A survey. Expert Systems with Applications, page 124982, 2024.

[29] Laurens Van Der Maaten, Eric O Postma, H Jaap Van Den Herik, et al. Dimensionality reduction: A comparative review. Journal of machine learning research, 10(66-71):13, 2009.

[30] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In International conference on machine learning, pages 1139–1147. PMLR, 2013.

[31] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3):107–115, 2021.

[32] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50:102419, 2020.

[33] Sunanda Gamage and Jagath Samarabandu. Deep learning methods in network intrusion detection: A survey and an objective comparison. Journal of Network and Computer Applications, 169:102767, 2020.

[34] Peng Lin, Kejiang Ye, and Cheng-Zhong Xu. Dynamic network anomaly detection system by using deep learning techniques. In Cloud Computing–CLOUD 2019: 12th International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 12, pages 161–176. Springer, 2019.

[35] Feng Zhao, Hao Zhang, Jia Peng, Xiaohong Zhuang, and Sang-Gyun Na. A semi-self-taught network intrusion detection system. Neural Computing and Applications, 32(23):17169–17179, 2020.

[36] Muhammad Ashfaq Khan. Hcrnnids: Hybrid convolutional recurrent neural network-based network intrusion detection system. Processes, 9(5):834, 2021.

[37] Joffrey L Leevy, John Hancock, Richard Zuech, and Taghi M Khoshgoftaar. Detecting cybersecurity attacks across different network features and learners. Journal of Big Data, 8(1):1–29, 2021.

[38] Hawazen Alzahrani, Tarek Sheltami, Abdulaziz Barnawi, Muhammad Imam, and Ansar Yaser. A lightweight intrusion detection system using convolutional neural network and long short-term memory in fog computing. Computers, Materials and Continua, 80(3):4703–4728, 2024.

[39] Amogh Deshmukh and Kiran Ravulakollu. An efficient cnn-based intrusion detection system for iot: Use case towards cybersecurity. Technologies, 12(10):203, 2024.

[40] Charles Stolz, Fuhao Li, and Jielun Zhang. Implementing lightweight intrusion detection system on resource constrained devices. In 2024 Cyber Awareness and Research Symposium (CARS), pages 1–6, 2024.

[41] Promise R Agbedanu, Richard Musabe, James Rwigema, Ignace Gatare, and Yanis Pavlidis. Ipcasamknn: A novel network ids for resource constrained devices. In 2022 2nd International Seminar on Machine Learning, Optimization, and Data Science (ISMODE), pages 540–545. IEEE, 2022.

[42] Xuan-Ha Nguyen, Xuan-Duong Nguyen, Hoang-Hai Huynh, and Kim-Hung Le. Realguard: A lightweight network intrusion detection system for iot gateways. Sensors, 22(2):432, 2022.