

Article

CTSF: An Intrusion Detection Framework for Industrial Internet Based on Enhanced Feature Extraction and Decision Optimization Approach

Guangzhao Chai, Shiming Li *, Yu Yang, Guohui Zhou  and Yuhe Wang *

College of Computer Science and Information Engineering, Harbin Normal University, Harbin 150025, China; cgz010321@163.com (G.C.); yang11zc@163.com (Y.Y.); zhouguohui@hrbnu.edu.cn (G.Z.)

* Correspondence: hsdslm@163.com (S.L.); cs2008wyh@163.com (Y.W.)

Abstract: The traditional Transformer model primarily employs a self-attention mechanism to capture global feature relationships, potentially overlooking local relationships within sequences and thus affecting the modeling capability of local features. For Support Vector Machine (SVM), it often requires the joint use of feature selection algorithms or model optimization methods to achieve maximum classification accuracy. Addressing the issues in both models, this paper introduces a novel network framework, CTSF, specifically designed for Industrial Internet intrusion detection. CTSF effectively addresses the limitations of traditional Transformers in extracting local features while compensating for the weaknesses of SVM. The framework comprises a pre-training component and a decision-making component. The pre-training section consists of both CNN and an enhanced Transformer, designed to capture both local and global features from input data while reducing data feature dimensions. The improved Transformer simultaneously decreases certain training parameters within CTSF, making it more suitable for the Industrial Internet environment. The classification section is composed of SVM, which receives initial classification data from the pre-training phase and determines the optimal decision boundary. The proposed framework is evaluated on an imbalanced subset of the X-IIOTID dataset, which represent Industrial Internet data. Experimental results demonstrate that with SVM using both “linear” and “rbf” kernel functions, CTSF achieves an overall accuracy of 0.98875 and effectively discriminates minor classes, showcasing the superiority of this framework.

Keywords: Industrial Internet; intrusion detection; convolutional neural network; Transformer; Support Vector Machines



Citation: Chai, G.; Li, S.; Yang, Y.; Zhou, G.; Wang, Y. CTSF: An Intrusion Detection Framework for Industrial Internet Based on Enhanced Feature Extraction and Decision Optimization Approach. *Sensors* **2023**, *23*, 8793. <https://doi.org/10.3390/s23218793>

Academic Editors: Jun Wu and Qianqian Pan

Received: 27 September 2023

Revised: 20 October 2023

Accepted: 27 October 2023

Published: 28 October 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Industrial Internet [1] is an emerging domain that amalgamates conventional industrial systems with internet technologies. Leveraging advanced techniques like the Internet of Things (IoT), cloud computing, and big data analytics, it facilitates the interconnectivity and intelligent management of large-scale devices within industrial production environments. The proliferation of Industrial Internet and its digital transformation has brought forth numerous challenges [2]. Firstly, as the networking degree of industrial control systems escalates, industrial networks become exposed to a plethora of threats. Malicious actors can disrupt, interfere, or even gain control over industrial systems through network intrusion and malevolent activities, severely impacting industrial production and infrastructure. Secondly, security vulnerabilities and weaknesses in the Industrial Internet have gradually become targets for attackers. Industrial control systems often employ antiquated devices and software, lacking timely security patches and updates. Concurrently, attackers exploit advanced network attack techniques such as zero-day vulnerabilities and targeted assaults on specific industrial protocols, infiltrating and penetrating the Industrial

Internet. Lastly, the Industrial Internet harbors a substantial amount of sensitive production data and operational information. Should these data fall into the hands of malicious entities, it can lead to substantial losses for enterprises. Furthermore, malicious intrusions can result in severe consequences including production interruptions, equipment damage, and industrial accidents, thereby posing significant threats to the production and economic interests of enterprises.

To bolster the security of Industrial Internet systems, traditional network security methods are adapted for use in Industrial Internet contexts. Figure 1 depicts a generalized security architecture for safeguarding Industrial Internet platforms. The left portion of the diagram represents the entire lifecycle management process of data in the Industrial Internet, with a higher susceptibility to malicious attacks during the data collection and transmission phases. In the middle section of the diagram, passive defense measures against malicious attacks are presented, including the common service models in cloud computing: SaaS, PaaS, and IaaS. These models play a pivotal role in establishing and providing secure computing and storage environments. On the right side of the diagram, active defense measures against malicious attacks are depicted; the Intrusion Detection System (IDS) represents an active defense component within the depicted framework, playing a significant role in the Industrial Internet context. IDS, a form of network security tool [3,4], is designed to monitor and identify potential intrusion activities, thereby safeguarding the integrity of Industrial Internet systems. By implementing effective IDS solutions, the Industrial Internet can ensure the reliability, integrity, and confidentiality of its systems, thereby establishing a stable and secure environment for industrial production. IDS can be categorized into signature-based IDS (SIDS) and anomaly-based IDS (AIDS) [5]. SIDS relies on predefined attack patterns or feature databases for intrusion detection, performing matching against network traffic, system logs, or other data using known attack pattern signatures. However, it can only identify known intrusions. On the other hand, AIDS is a method that detects abnormal activities by analyzing the normal behavioral patterns of a system. AIDS constructs baseline models of normal network traffic, system behavior, and user activities to detect anomalous behavior that deviates from the established normal model. This approach can identify unknown attacks and emerging threats.

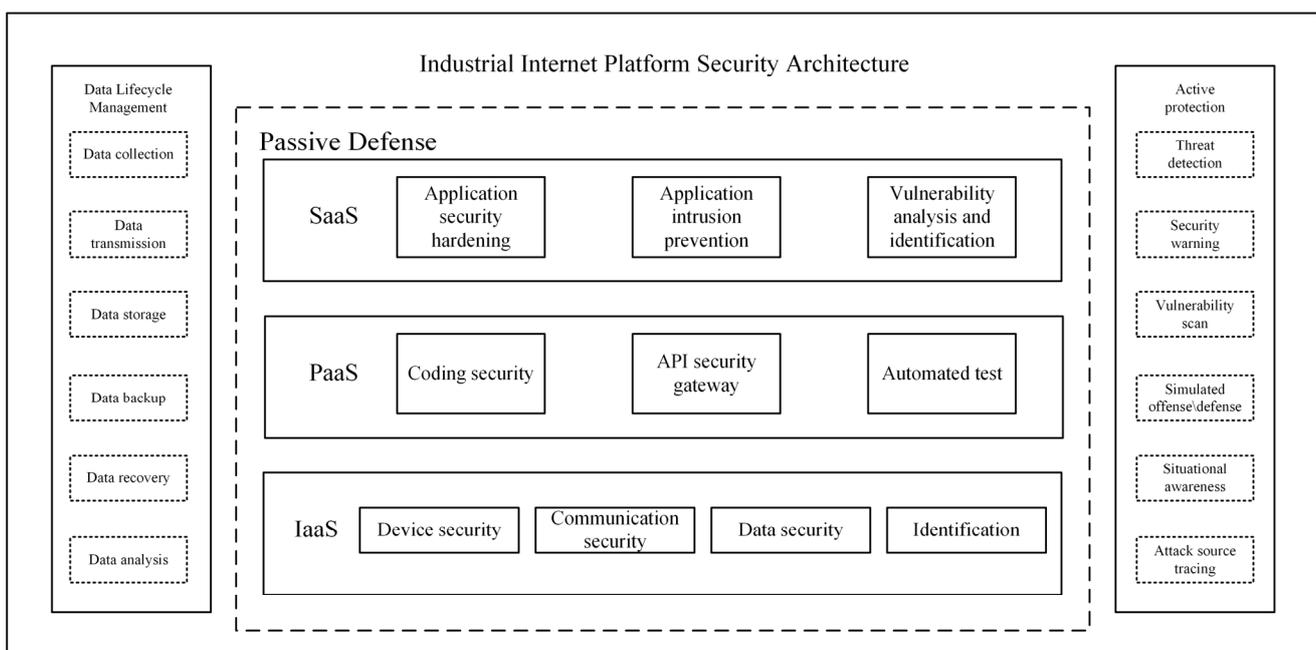


Figure 1. Industrial Internet platform security architecture.

In the context of AIDS, machine learning techniques (ML) are extensively employed to enhance the automation and intelligence of IDS, thereby reducing false positive rates and better addressing evolving network threats [6–8]. Support Vector Machine (SVM), a classical ML algorithm, finds widespread application in intrusion detection [9,10]. SVM boasts significant advantages in handling high-dimensional data, generalization, small-sample data, and linear and non-linear problems. However, in IDS, data features can sometimes be redundant, or their impact on accurately classifying data points might be marginal. Hence, SVM models are often used in conjunction with feature selection algorithms and model optimization methods to maximize SVM's classification accuracy [11]. An effective feature selection approach involves the use of deep learning networks (DL) [12–14]. DL, achieved through the combination and training of multi-layered neural networks, can automatically acquire sophisticated abstract feature representations from raw data, greatly streamlining the feature extraction process. At its core, DL utilizes neural network models [15], comprising multiple layers of neurons, with each layer performing a series of non-linear transformations and feature extractions on input data. As the network depth increases, DL gradually learns more abstract and complex feature representations, thereby achieving a deep understanding and analysis of the data. Transformer, a DL network architecture based on self-attention mechanisms [16], dynamically computes attention weights for each position based on the relationships between different feature positions in the input data. By summing the weighted attention weights for all positions, the Transformer model can simultaneously consider the feature information of the entire input data, thus capturing global feature relationships more effectively. However, when it comes to local relationships within input data, its modeling capacity is limited.

The self-attention mechanism calculates attention weights by considering all positions in the sequence, making it challenging to focus attention effectively on locally related positions. Capturing local features is crucial for Industrial Internet intrusion detection. The network traffic and data in Industrial Internet environments typically contain a wealth of subtle local information; this information may appear in specific segments or time intervals within the entire data stream, such as specific behaviors, patterns, or features associated with attacks. Capturing these local features enables more accurate identification of anomalies or potential intrusion behaviors. Furthermore, focusing solely on global features can make the model more susceptible to interference from normal variations or noise, leading to false alarms. By capturing local features, the model can better distinguish between normal and abnormal changes, thus reducing the false positive rate. Shallow convolutional kernels of convolutional neural networks (CNN) [17–19] precisely extract local features from data and leverage parameter sharing mechanisms to significantly reduce the network's parameter count.

Therefore, the CNN-Transformer network structure not only automatically extracts features but also compensates for the Transformer model's deficiencies in local feature extraction by harnessing CNN's precise extraction of local features [20].

In consideration of various factors, this paper presents a network framework named CTSF for intrusion detection in the Industrial Internet. CTSF's pre-training phase initially submits input data to a CNN for extracting sufficient local features. Subsequently, these features undergo positional encoding and are passed to an enhanced Transformer's encoder structure. The self-attention mechanism is employed for holistic feature extraction. Finally, the decoder, composed of fully connected layers, further abstracts the encoded features. After ample training in the pre-training phase, CTSF extracts the final layer's output from this phase, which is then passed to the SVM in the decision-making segment for further training, ultimately yielding classification outcomes. In theory, due to CTSF addressing the limitations of traditional Transformer and SVM models, its classification accuracy is expected to be higher than the unimproved models it comprises. This proposition will also be substantiated in the subsequent experimental section. The decoder of CTSF's pre-training phase consists of fully connected layers and does not receive label data or employ self-attention mechanisms for label-specific feature extraction. This is due to the fact that

in the context of abnormal traffic in the Industrial Internet, label data often represent a category of attack and possess only one-dimensional features. While utilizing self-attention mechanisms to extract features from label data might yield marginal effects, it significantly increases the overall network parameters. Figure 2 illustrates the fundamental process of the traditional Transformer in handling data with high-dimensional label classes, as well as the modified version proposed in this paper for processing Industrial Internet network traffic data.

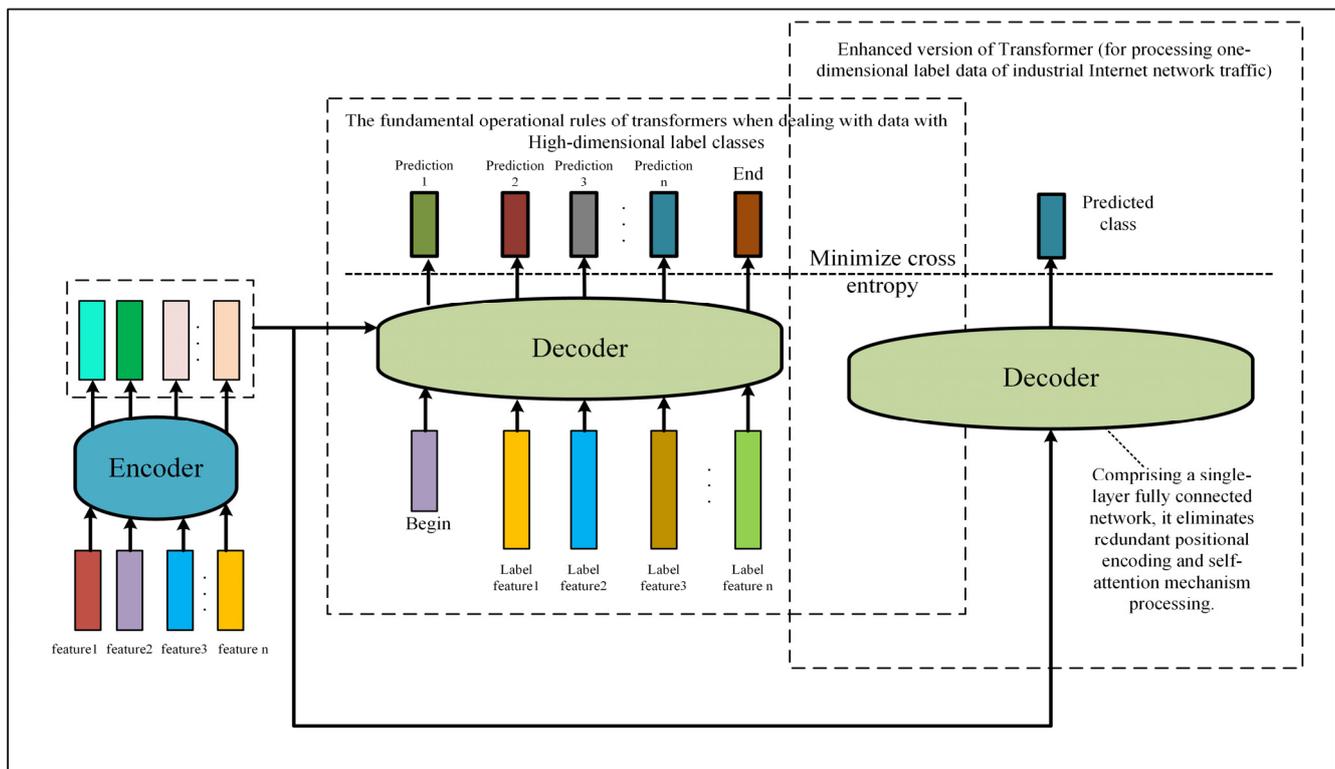


Figure 2. Comparison between the fundamental process of the Transformer in handling data with high-dimensional label classes and the basic process of the modified Transformer proposed in this paper for processing Industrial Internet network traffic data.

1. Introduces a novel network framework, CTSE, tailored for intrusion detection in the Industrial Internet. This framework is rooted in the actual conditions of the Industrial Internet environment and modifies the traditional Transformer model's decoder structure to better handle abnormal traffic.
2. The CTSE framework leverages the advantage of CNN in extracting local features, compensating for the traditional Transformer's shortcomings in this aspect, and effectively applying to intrusion detection in the Industrial Internet.
3. This paper conducts simulation experiments on the Industrial Internet dataset X-IITID with CTSE, and the results demonstrate CTSE's ability to accurately recognize small-sample categories, achieving an overall accuracy of 0.98875.

The remaining sections of this paper are organized as follows: Section 2 discusses the relevant background; Section 3 provides a comprehensive description of CTSE's structure, forward propagation, and training process; Section 4 introduces the selected dataset and its processing methodology, model selection for comparative experiments, specifications of each model's parameters, model evaluation metrics, and final experimental outcomes; Section 5 concludes the paper and proposes future avenues for improvement.

2. Related Work

2.1. SVM

SVM, a classical method in the field of machine learning, has been widely applied and has shown remarkable performance in intrusion detection [21]. With its unique characteristics and advantages, SVM has become a powerful tool for addressing classification and regression problems. SVM typically requires the use of feature selection algorithms to achieve its maximum effectiveness [11]. Almaiah et al. introduced an intrusion detection model employing SVM with feature selection using principal component analysis and various kernel functions [22]. They examined the influence of different kernel functions on SVM and found that SVM combined with the 'rbf' kernel function and principal component analysis feature selection technique achieved the highest accuracy. Saheed et al. attempted to address the dynamics of network attack patterns using the k-means clustering algorithm, followed by feature selection using genetic algorithms, and finally employed SVM for classification [23]. This approach achieved an impressive accuracy of 99% at low FAR, showcasing remarkable success. Not only in the realm of feature selection, but in other domains as well, researchers have utilized CNN for rapid feature extraction and employed the extracted features as input data for SVM classification. For instance, in the field of medical tumor detection, Khairandish et al. combined CNN with SVM [24], harnessing the automatic feature extraction capabilities of CNN and the high accuracy and speed of SVM to successfully distinguish between benign and malignant brain tumors. In the domain of weed recognition, Tao et al. used CNN to extract features from crop images [25], eliminating the need for manual feature extraction and demonstrating that CNN-SVM achieves higher classification accuracy and robustness.

2.2. CNN

CNN finds extensive application in intrusion detection within the Industrial Internet, automatically learning and extracting features from input data [26,27]. Utilizing convolutional and pooling layers, CNN can effectively capture both local and global patterns in sequential data, encompassing crucial feature correlations and dependencies. These learned features can be employed to identify potential intrusion behaviors and anomalous activities. Pingale S V and his team have proposed a robust Remora Whale Optimization (RWO)-based hybrid deep model [28]. This model preprocesses data, extracts features using CNN, and then employs a hybrid algorithm for training, achieving outstanding classification performance. El-Ghamry et al. employed various CNN architectures [29], such as VGG16, Inception, and Xception, to further abstract and process network traffic converted into color images. They also compared the performance of these CNN architectures with other machine learning algorithms, ultimately demonstrating the performance advantage and high classification accuracy of CNN architectures. Qazi et al. proposed a deep learning architecture based on one-dimensional convolutional neural networks [30], which maintained structural simplicity and low computational cost while exhibiting remarkably high classification accuracy. Halbouni et al. developed an intrusion detection system using a CNN-based learning algorithm [31], empirically confirming that increasing the number of layers and neurons per layer can enhance model performance. This intrusion detection system achieved a 99.55% accuracy and 99.63% detection rate in multi-class classification. Despite CNN's remarkable performance in intrusion detection, its effectiveness in global feature extraction is limited by the receptive field range [32]. To overcome this limitation, some researchers have begun combining CNN with models that excel in extracting global features, and one such model to be mentioned next is the Transformer.

2.3. Transformer

Transformer, a neural network model based on self-attention mechanisms, has been widely employed in the field of natural language processing. However, its advantages in sequence modeling and global feature extraction can also be applied to the domain of intrusion detection. Wu et al. introduced a Robust Transformer-based Intrusion Detec-

tion System (RTIDS) [33], which employs positional embedding techniques to correlate sequential information among features. Through a variant of stacked encoder–decoder architecture, it learns low-dimensional feature representations from high-dimensional raw data. Experimental comparisons with neural networks such as RNNs and LSTMs that extract sequence information validate the effectiveness of this Transformer-based intrusion detection system.

Wang et al. proposed a Transformer-based NIDS method for IoT [34], utilizing the self-attention mechanism in encoders and decoders to learn contextual relationships among input network features, resulting in outstanding classification performance on the ToN-LoT dataset. Liu et al. introduced an enhanced intrusion detection model based on Transformer [35], primarily altering the positional encoding method of the traditional Transformer model. By embedding the positional information of features, they effectively captured feature dependencies, consequently improving classification accuracy. Tan et al. presented a Transformer-based intrusion detection system [36] which, due to the absence of recurrence and the use of slot-based functions, can achieve real-time detection efficiency exceeding that of bidirectional LSTMs.

While Transformer exhibits strong capabilities in global feature extraction in the aforementioned studies, it still has limitations when applied to Industrial Internet of Things intrusion detection. Firstly, it lacks an explicit mechanism for local modeling. The Transformer model primarily models relationships between elements in a sequence using self-attention mechanisms, without a dedicated mechanism for explicit handling of local features. This implies that the Transformer model might rely on global context when processing local information, potentially leading to imprecise capture of local features. Secondly, in industrial IoT environments, network traffic labels often consist of only one dimension. In the traditional Transformer architecture, the decoder employs self-attention mechanisms to process label data. When network traffic is passed into the Transformer architecture, the decoder expends significant time and parameters to extract features from one-dimensional label data, which is undesirable.

Addressing the aforementioned issues, one solution for the first problem is to combine the advantages of CNN-based local feature extraction and Transformer-based global feature extraction, thereby compensating for the deficiencies of both architectures. A solution to the second problem is proposed in the third section of this paper.

2.4. CNN-Transformer

The CNN-Transformer fusion combines the advantages of both the CNN and Transformer model, simultaneously possessing the capabilities of local feature extraction and global context modeling [37]. Yao R et al. introduced a hybrid intrusion detection system architecture based on CNN-Transformer [38]. They first employed an XGBoost-based feature selection strategy to eliminate irrelevant features from network traffic. Subsequently, they constructed a CNN to extract local features from the traffic and then utilized a Transformer to establish feature correlations and extract global features. The effectiveness and superiority of this hybrid structure were demonstrated through testing on the KDDCup99, NSL-KDD, and CICIDS-2017 datasets. Luo S et al. proposed a layered intrusion detection model with spatio-temporal feature fusion based on CNN-Transformer [39]. They leveraged CNN to extract spatial features and employed the Transformer's positional encoder to embed location information into input data, cleverly capturing time features while globally modeling data. This model highlights the significance of the Transformer's attention mechanism in soft feature selection. Their experiments on the UNSW-NB15 dataset underscore the advancement of the proposed model. Indeed, what sets CTSF apart from the models proposed by the aforementioned researchers is the incorporation of SVM as a new final decision-making component. With the inclusion of SVM, CTSF combines the strengths of both the softmax classifier and the SVM classifier, thus further enhancing classification accuracy.

3. Framework Analysis

In the field of industrial production and operations, specific types of equipment, sensors, and control systems generate network traffic data within the Industrial Internet environment. This data serves to record real-time status, performance parameters, and operational conditions of industrial processes, facilitating remote monitoring, data analysis, and automated control. These devices encompass a range of sensors (such as temperature, humidity, pressure, and flow sensors), PLCs (Programmable Logic Controllers), SCADA systems (Supervisory Control and Data Acquisition systems), industrial robots, IoT devices, data acquisition cards, industrial switches, routers, and more. Through these devices, the Industrial Internet environment can produce diverse network traffic data to support the efficient management and optimization of industrial processes. This data contains various information crucial for achieving intelligent production, monitoring, and management. To extract features more accurately from network traffic for enhanced classification precision, a novel intrusion detection framework named CTSF is proposed in this paper. It comprises two components: a pre-training section and a decision-making section. The pre-training section of CTSF is composed of a CNN and an improved Transformer structure tailored to the domain of Industrial Internet intrusion detection, while the classification section involves an SVM classifier. The overall structure is illustrated in Figure 3. In this section, we will derive the forward propagation process of CTSF, present the parameters utilized during this process, and elucidate their respective significance as shown in Table 1. The parameters in Table 1 represent abstract symbols, and these symbols collaborate during the inference process of forward propagation, mathematically illustrating the feature extraction and model prediction processes of CTSF.

Table 1. The parameters employed in the inference process of CTSF forward propagation and the corresponding meaning of each parameter.

No.	Parameters	Meaning
1	n	The number of features a piece of network traffic data has
2	X	A feature vector corresponding to a network traffic event
3	N_c	The “length” of the convolution kernel
4	$step_c$	The moving step of the convolution kernel
5	X_{c_i}	The initial eigenvector of each step is convolved to obtain the process value
6	w_c	The weight parameters of the convolution kernel
7	b_c	The bias vector of the convolution kernel
8	X_c	The feature vector obtained after each convolution kernel of CTSF executes all steps
9	N_p	The “length” of X_c , it is numerically equal to n
10	N_m	The “length” of the max pooling kernel
11	$step_m$	The moving step size of the pooling kernel
12	X_m	The feature vector is obtained by X_c through the maximum pooling layer
13	X_M	The feature vector obtained after X undergoes all convolution operations and pooling operations of CTSF
14	X_t	The feature vector obtained after X passes through the position encoder part of the CTSF
15	\vec{p}_i	Represents the vector corresponding to position i , containing pairs of sine and cosine for each frequency.
16	a	Represents an integer value that controls the odd and even conditions in \vec{p}_i .
17	ω_k	frequency
18	$f(i)^{(a)}$	The function that generates the position vector $\vec{p}_i^{(a)}$
19	d_{model}	Represents the dimension of each feature deepening
20	num	Numerically equal to the feature dimension of X_M
21	W_Q, W_K, W_V	Corresponds to the weight matrix of the query matrix, the weight matrix of the key matrix, and the weight matrix of the value matrix
22	Q_i	The query vector for the i -th header in the CTSF pre-training part
23	num_h	The number of heads in the multi-head attention mechanism
24	K_i	The key vector for the i -th header in the CTSF pre-training section
25	V_i	The value vector for the i -th header in the CTSF pre-training section

Table 1. Cont.

No.	Parameters	Meaning
26	A_i	The output of the self-attention mechanism of the i -th head
27	X_A	The results obtained by the self-attention mechanism of all heads are combined to obtain
28	μ, σ	represent the mean and standard deviation of X_A , respectively
29	X_{NA}	The vector obtained after X_A passes through the standard layer
30	F_{R1}	Represents the process of mapping X_i to X_{NA}
31	X_{R1}	CTSF establishes a residual connection at the multi-head attention mechanism layer, and this value represents the result of the residual connection
32	F_{R2}	Represents X_{R1} through the mapping process of the fully connected layer and the normalized layer
33	X_{R2}	CTSF establishes a residual connection in the feedforward fully connected layer, and this value represents the result of the residual connection
34	F_l	Represents X_{R2} through the mapping process of the fully connected layer
35	P	Represents the result of the original feature vector X after sufficient pre-training
36	w_f	The weight parameter matrix of the feed-forward fully connected layer of the CTSF pre-training part
37	b_f	The bias value vector of the feed-forward fully connected layer of the CTSF pre-training part
38	w_d	The weight parameter matrix of the decoder layer of the CTSF pre-training part
39	b_d	The bias value vector of the decoder layer of the CTSF pre-training part
40	w_l	Represents the weight parameters of the SVM
41	b_l	Represents the bias value of the SVM

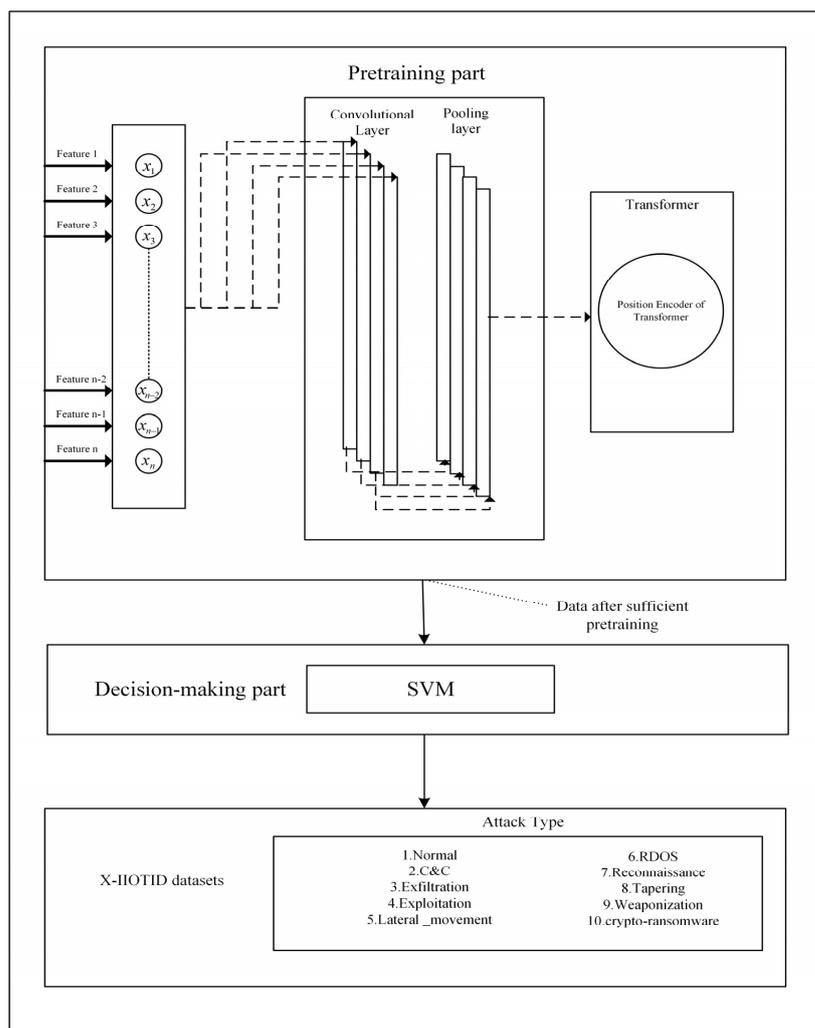


Figure 3. Basic structure of CTSF(The dotted arrows in the pre-training section represent the direction of data transfer from one module to another.).

3.1. Pre-Training Part

3.1.1. CNN in the Pre-Training Part

In the pre-training phase, CTSF primarily employs CNN for extracting local features from the input data. Assume a network traffic event X with a shape of $(1, n)$, where n represents the number of features in X . A convolutional kernel is used with a shape of $(1, N_c)$ and a movement stride of $step_c$. Considering that the convolution operation reduces the dimension of the original feature vector, CTSF applies zero-padding to both ends of the data before convolution.

This step aims to minimize the potential loss of edge features. The padding process is demonstrated by Formula (1):

$$X = [0, x_1, x_2, x_3, \dots, x_n, 0] \quad (1)$$

After the padding operation, the shape of X becomes $(1, n + 2)$. Following the padding, CTSF proceeds with the convolution operation. Each initial feature vector at every stride undergoes convolution to yield the process value X_{c_i} . This process is illustrated by Formula (2):

$$X_{c_i} = ReLU(w_c X_{i+i+N_c}^T + b_c) \quad (2)$$

Here, w_c and b_c represent the weight parameters and bias vector of the convolutional kernel, respectively. After completing all strides for each convolutional kernel, CTSF obtains X_c , with a shape of $(1, N_p)$, where N_p is computed as demonstrated in Formula (3). In fact, N_p is numerically equal to n , as is due to the padding operation performed by CTSF before the convolution.

$$N_p = \left\lceil \frac{n - N_c + 3}{step_c} \right\rceil \quad (3)$$

Subsequently, X_c will undergo a max-pooling layer with a shape of $(1, N_m)$ and a stride of $step_m$. The process of obtaining X_m is illustrated in Formulas (4)–(6):

$$X_{m_i} = Max(X_{c_i}, X_{c_{i+1}}, \dots, X_{c_{i+N_m-1}}, \dots, X_{c_{i+N_m}}) \quad (4)$$

$$Num_m = \left\lceil \frac{n - N_m + 1}{step_m} \right\rceil \quad (5)$$

$$X_m = [X_{m_1}, X_{m_2}, \dots, X_{m_{Num_m-1}}, X_{m_{Num_m}}] \quad (6)$$

After undergoing multiple layers of identical convolutional operations, X results in X_M , and the mapping relationship is depicted in Formula (7):

$$X_M = f_{cnn}(X) \quad (7)$$

3.1.2. Transformer in the Pre-Training Part

The encoder section of the pre-training phase's Transformer structure consists of two encoder layers, each containing multi-head self-attention mechanisms, normalization layers, and feed-forward fully connected layers with residual connections. The primary purpose of the encoder is to capture the global features within the input data X_M . With the application of the self-attention mechanism, the encoder can effectively model X_M , capturing both global and selective local feature information. The decoder section comprises a fully connected layer with an output dimension matching the number of output channels from the CNN in the pre-training phase. The improved decoder part eliminates the self-attention mechanism and avoids extracting information from the labels. Instead, it transforms the features extracted from the encoder into a higher-level feature representation, reducing the training parameters in CTSF while better understanding abstract features within the input data. The structure of the pre-training phase's Transformer is illustrated in Figure 4.

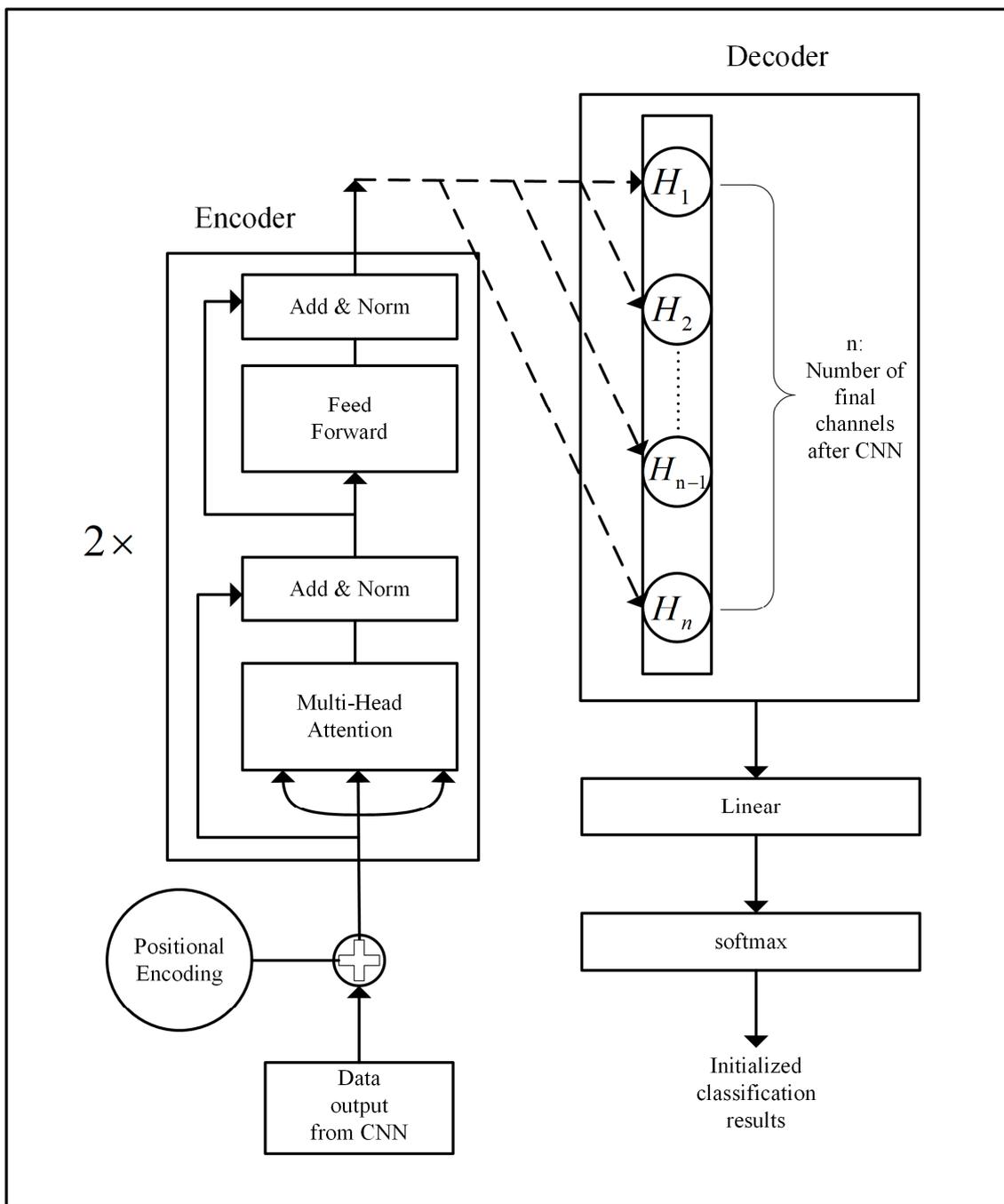


Figure 4. The improved Transformer architecture of the pre-training part. (The dotted arrows represent that not all data from the encoder part is input to each individual neuron in the enhanced decoder structure, but that the input of each feature corresponds one-to-one with the subscripts of H_i .)

This section receives the feature information X_M passed from the CNN section and first processes it with positional encoding. Formulas (8)–(11) illustrate the process through which CTSF obtains X_t .

$$\vec{p}_i^{(a)} = f(i)^{(a)} := \begin{cases} \sin(\omega_k \cdot i), & \text{if } a = 2k \\ \cos(\omega_k \cdot i), & \text{if } a = 2k + 1 \end{cases} \tag{8}$$

$$\omega_k = \frac{1}{10000^{2k/d_{\text{model}}}} \tag{9}$$

$$X_{t_i} = X_{M_i} + \vec{p}_i \quad (10)$$

$$X_t = [X_{t_1}, X_{t_2}, \dots, X_{t_{num}}] \quad (11)$$

Here, a represents an integer value that controls the odd and even conditions in \vec{p}_i , num which is equal to the feature dimension of X_M , and d_{model} represents the dimension by which each feature is deepened.

In the multi-head attention layer of the encoder part, each feature in X_t has three attributes: $Q = X_t \cdot W_Q$, $K = X_t \cdot W_K$, $V = X_t \cdot W_V$, where W_Q , W_K , and W_V are learnable weight matrices. Features and attributes in X_t are evenly mapped into multiple subspaces where the following is true: the query matrix Q maps the input sequence to the query subspace, resulting in query vectors $Q_1, Q_2, Q_3 \dots Q_{numh-1}, Q_{numh}$; the key matrix K maps the input sequence to the key subspace, resulting in key vectors $K_1, K_2, K_3 \dots K_{numh-1}, K_{numh}$; and the value matrix V maps the input sequence to the value subspace, resulting in value vectors $V_1, V_2, V_3 \dots V_{numh-1}, V_{numh}$. Here, $numh$ represents the number of heads in the multi-head attention mechanism. Next, CTSF calculates the attention weights for each head. For the i -th head, CTSF computes the similarity between the query vector Q_i and the key vector K_i , scales the similarity by the dimension d_K of K_i , normalizes the similarity using the softmax function, and finally computes the weighted average of values using the attention weights to obtain the output A_i of the self-attention mechanism (Formula (12)).

$$A_i = Attention(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_K}}\right) V_i \quad (12)$$

CTSF combines the outputs of the self-attention mechanism from all heads to obtain X_A (Formula (13)). At this point, the encoder part has adequately captured both the global and partial local features of X_M .

$$X_A = [A_1, A_2, A_3, \dots, A_{numh-1}, A_{numh}] \quad (13)$$

Formula (14) shows the process of obtaining X_{NA} from X_A through the normalization layer:

$$X_{NA} = \frac{X_A - \mu}{\sigma} \quad (14)$$

Here, μ represents the mean of X_A , and σ represents the standard deviation of X_A . CTSF establishes residual connections in the multi-head attention layer, as demonstrated in Formula (15).

$$X_{R1} = X_t + F_{R1}(X_t) \quad (15)$$

Here, F_{R1} represents the mapping process from X_t to X_{NA} . Subsequently, X_{R1} undergoes a feed-forward fully connected layer and a normalization layer. Moreover, CTSF establishes residual connections in the feed-forward fully connected layer, as manifested in Formula (16):

$$X_{R2} = X_{R1} + F_{R2}(X_{R1}) \quad (16)$$

In this context, F_{R2} denotes the mapping process of X_{R1} accomplished through the fully connected layer and the normalization layer. Subsequently, CTSF forwards X to the decoder, where it undergoes further abstraction through the decoder's fully connected layers and partial feature dimension reduction (with CTSF's decoder output channels being fewer than the decoder input channels), thereby reducing model training parameters. The results are then passed to the fully connected layer outside the decoder, followed by classification using the softmax function. (Formula (17))

$$cls = softmax(F_l(X_{R2})) \quad (17)$$

Here, F_l represents the mapping process of X_{R2} through the fully connected layer.

3.2. Decision-Making Part

The CTSF framework reassigns well-pretrained initialization classification data to SVM for retraining. This design first combines the classification results of softmax and SVM to further enhance accuracy. Secondly, SVM maps this data to a high-dimensional space using a kernel function and optimizes model parameters with the hinge loss function. It searches for an optimal hyperplane in the feature space that effectively separates different class samples while maximizing the margin between the hyperplane and the nearest sample points to improve the framework's generalization capability. After training, the SVM produces a score vector for each class using the decision function $F(P)$, where P represents the output from the trained pre-training section: $F(P) = \text{sign}(w_l P + b_l)$. Here, w_l and b_l represent weight parameters and bias values, respectively. Introducing the argmax function. This will yield the predicted result, denoted as pred . Formula (18) shows this process:

$$\text{pred} = \text{argmax}(F(P)) \quad (18)$$

3.3. The Training Process of the CTSF

The training of CTSF consists of two parts: the initial training of the pre-training section and the final training of the decision section. Table 2 presents the overall training process of CTSF in the form of pseudocode:

Table 2. Pseudocode of CTSF overall training process. (The content after the '#' symbol is a comment.)

Input:

train_data_iterator: It stores all the X and the attack types corresponding to X that are needed to train the model.
 $w_c, b_c, W_Q, W_K, W_V, w_f, b_f, w_d, b_d, w_l, b_l$: Matrix of weight parameters and bias values.
 num_epochs: Total training epochs.
 num_batches: The number of batches included in an epoch.
 train_features: SVM training data (excluding categories), the type is an array.
 train_labels: Corresponding to train_features, representing the category of training data, the type is also an array.

Process:

```

preprocess_data(X) # Preprocess the dataset.
initialize_parameters( $w_c, b_c, W_Q, W_K, W_V, w_f, b_f, w_d, b_d, w_l, b_l$ ) # Initialize weight parameters.
for epoch in range(num_epochs):
    total_loss = 0
    for batch_data, batch_labels in train_data_iterator:
        predictions = pretraining_CTSF(batch_data)
        loss = compute_loss(predictions, batch_labels)
        total_loss += loss
        backpropagation(loss)
        update_parameters( $w_c, b_c, W_Q, W_K, W_V, w_f, b_f, w_d, b_d$ )
    end for
    average_loss = total_loss/num_batches
end for
for batch_data, batch_labels in train_data_iterator:
    batch_features = pretraining_CTSF.extract_features(batch_data) # Obtain data from the pre-training part.
    train_features.append(batch_features)
    train_labels.append(batch_labels)
end for
svm_CTSF(train_features, train_labels) # update  $w_l$  and  $b_l$ 

```

Output:

```

predictions = svm_CTSF.predict(test_features)

```

4. Experimental Setup and Discussion

In this study, the X-IIoTID dataset from the Industrial Internet domain was employed to evaluate various performance metrics of the CTSF framework. CNN, RNN, CNN-RNN, and CNN-Transformer (improved) models were selected as control groups. These currently advanced models were also tested using the same dataset with consistent baseline

parameters to derive performance metrics. Finally, a comparative analysis was conducted between the mentioned models and the CTSF framework to demonstrate the superiority of our proposed approach. Section 4.1 presents the basic information about the dataset and its preprocessing procedures. Section 4.2 outlines the basic parameter settings for the CTSF model and the control group models. Section 4.3 introduces the evaluation metrics for the models. Section 4.4 presents the analysis and discussion of the experimental results.

4.1. Dataset Description and Preprocessing

The X-IIoTID datasets, specifically intrusion datasets agnostic to connections and devices [40–43], were used to capture the heterogeneity and interoperability of IoT systems. These datasets encompass behaviors of emerging IoT connection protocols, recent device activities, diverse attack types and scenarios, and various attack protocols. They define attack classes and incorporate multi-view attributes including network traffic, host resources, logs, and alerts. In this study, we selected a subset of the X-IIoTID dataset, containing 40,000 instances, each with 62 features, and the distribution of different classes is illustrated in Figure 5. During the dataset preprocessing phase, we removed columns named ‘is_SYN_with_RST’ and ‘Bad_checksum’ where all values were ‘False’. From columns labeled ‘class1’, ‘class2’, and ‘class3’ (‘class1’, ‘class2’, and ‘class3’, respectively, represent ‘normal and attack’, ‘normal and sub-category attack’, and ‘normal and sub-sub-category attack’.), we chose ‘class2’ as the label data. Subsequently, the data features were standardized. The processed dataset, after these procedures, took the shape of (40,000, 57). Finally, we employed the shuffle method to randomize the entire dataset and split it into training, validation, and testing sets in proportions of 80%, 10%, and 10%, respectively.

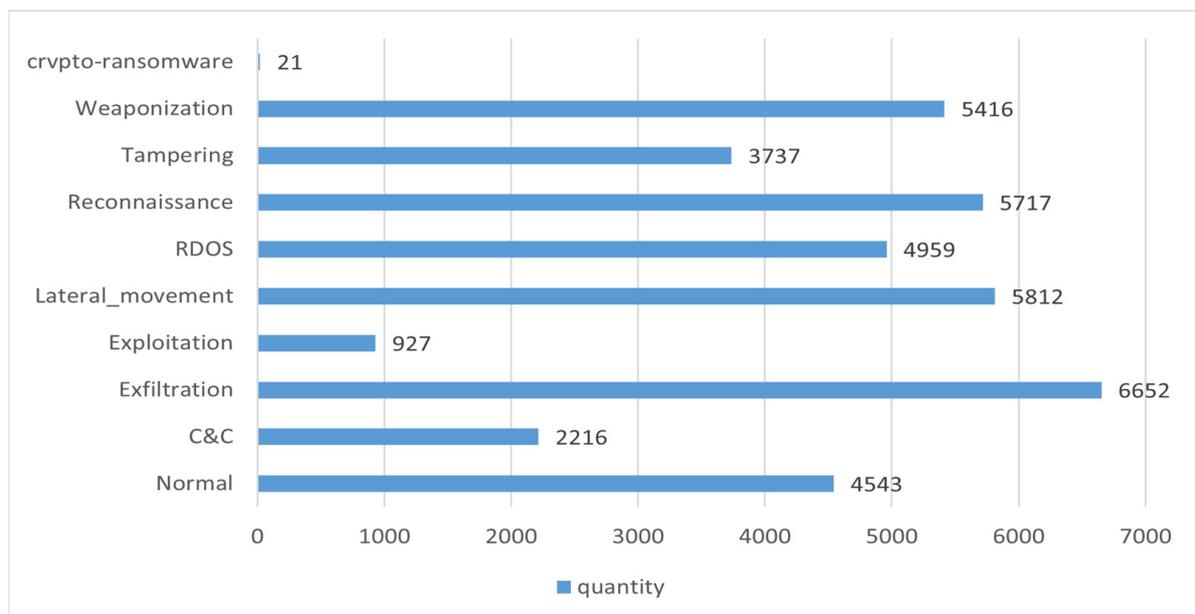


Figure 5. The number of various types in the data set selected in this paper.

4.2. Parameter Settings of CTSF and Control Models

4.2.1. Parameter Settings of CTSF

Prior to training CTSF, we reshaped the data to (40,000, 1, 57) to facilitate its passage through Conv1D for feature extraction.

The size and shape of the convolution kernel played a critical role in the CTSF, as the convolution operation involved sliding the kernel over the input data to extract features. This was essential because it directly affected how the model perceived and captured features from the input data. The CNN in the pre-training part of CTSF consisted of seven convolutional blocks, each comprising their own convolutional layer and max-pooling

layer. The max-pooling layers in all seven blocks shared the same parameters (stride = 2, padding = 0). The size of the convolutional kernels within each convolutional block was set to 2, and the padding parameter was uniformly configured as 1. The parameter configurations for each convolutional block were as follows. In the first convolutional block, the convolutional layer had an input channel of 1 and an output channel of 2. In the second convolutional block, the convolutional layer featured 2 input channels and 4 output channels. Subsequently, the convolutional layers in the following blocks had an input channel count that was double that of the previous layer, with the output channel count also being double that of the previous layer.

The sizes of the convolutional layers mentioned above were all set to 2, which was highly advantageous for extracting local features from the input data, thereby maximizing and enhancing CTSF's capability to extract local features effectively. If the size of the convolutional kernels is set larger than 2, theoretically, the local feature extraction capability of CTSF may not meet expectations. In the pre-training phase, to ensure that the input data could be passed through the CNN to the Transformer, the shape of the data coming out of the CNN had to have a third dimension of 1. Therefore, we configured the CNN part with 7 convolutional layers and 7 pooling layers. Each convolutional layer was accompanied by a max-pooling layer for feature reduction. After passing through the CNN section, the shape of the input data became (batch_size, 128, 1).

The pre-training section of CTSF's Transformer consisted of a positional encoder, an encoder, an improved decoder, and a fully connected layer. The following are the parameter settings for each component:

1. The parameter `d_model` in the positional encoder was set to 200, representing the extension of each feature of the input data to 200 dimensions. The dropout parameter was set to the default value of 0.1, and the `max_len` parameter limited the maximum number of features in the input data, which was set to the default value of 5000 in this study.
2. The parameter `d_model` in the encoder was set to 200, consistent with the `d_model` parameter in the positional encoder; the `nhead` parameter was set to 2, indicating that there were 2 parallel heads in the multi-head attention layer to concurrently extract features from the input data; the `nhid` parameter was set to 2, indicating that the entire encoder consists of two encoder layers; the dropout parameter was set to 0.2 to ensure that the overall framework did not overfit as much as possible. The shape of the input data changed to (batch_size, 128, 200) after passing through the encoder section.
3. The improved decoder consisted of a meticulously designed fully connected layer with an input feature dimension of 200 and an output feature dimension of 128, denoted as `in_features` and `out_features`, respectively. After passing through the decoder section, the shape of the input data changed to (batch_size, 128, 128).
4. The section of fully connected layers consisted of a total of five layers. The parameters for the first fully connected layer: `in_features` with a value of 128×128 and `out_features` with a value of 1000. The second layer had 1000 as `in_features` and 500 as `out_features`. In the third layer, the `in_features` parameter was set to 500, and the `out_features` parameter was set to 200. For the fourth layer, `in_features` was 200 and `out_features` was 50. Finally, in the fifth layer, `in_features` was 50 and `out_features` was 10; here, 10 represents the number of classes for initialization classification. The activation functions employed were ReLU for the first four layers and softmax for the final layer.

The classification component of CTSF employed an SVM classifier for final categorization. Throughout the experimentation process, the kernel parameter was set to 'rbf', 'poly', 'linear', and 'sigmoid' for training. Section 4.4 demonstrates the classification performance of CTSF using various kernel functions.

4.2.2. Parameter Settings of CNN-Transformer (Improved)

The parameters of CNN-Transformer (improved) were identical to those of CTSF's pre-training phase. The reason for setting up this model as a control group was to demonstrate the effectiveness of the operation in CTSF where ample pre-training using the softmax function was performed on the data before passing the initialized classification data to the SVM.

4.2.3. Parameter Settings of CNN-RNN

CNN-RNN comprises a CNN section, an RNN section, and a fully connected layer section. The parameters of the CNN section were the same as those of the CNN part in CTSF's pre-training phase. After passing through the CNN section, the input data's shape became (batch_size, 128, 1). The RNN section's input_size parameter was set to 1, indicating that the input data had 128 features, each with a one-dimensional dimension. The hidden_size parameter was set to 150, representing 150 hidden layers in the RNN. The num_layers parameter was set to 2. The RNN section used the result of the last hidden layer and squeezed the output data using the squeeze function to make it two-dimensional. Finally, the input data's shape became (batch_size, 150). The fully connected layer section comprised three fully connected layers. The in_features and out_features parameters of the first fully connected layer were set to 150 and 100, respectively. The parameters of the second fully connected layer were 100 and 50. The parameters of the third fully connected layer were 50 and 10.

4.2.4. Parameter Settings of CNN

CNN comprises a CNN section and a fully connected layer section. The parameters of the CNN section were the same as those of the CNN part in CTSF's pre-training phase. The fully connected layer section included three fully connected layers. The in_features and out_features parameters of the first fully connected layer were set to 128 and 100, respectively. The parameters of the second fully connected layer were 100 and 50. The parameters of the third fully connected layer were 50 and 10.

4.2.5. Parameter Settings of RNN

RNN comprises a RNN section and a fully connected layer section. Before training the RNN model, we preprocessed the dataset into a shape of (40,000, 57, 1) to ensure smooth forward propagation in the RNN. The parameters of both components of this model remained consistent with those in the corresponding segments of the CNN-RNN model.

The training parameters of CTSF and the control group models are presented in Table 3.

Table 3. Training parameters for CTSF and control models.

Model	Batch Size	Optimizer	Epochs	Learning Rate	Gamma	Step_Size	SVM C	SVM Degree
CTSF	32	Adadelta	500	0.01	0.65	50	1	3
CNN-Transformer (improved)	32	Adadelta	500	0.01	0.65	50	NULL	NULL
CNN-RNN	32	Adadelta	500	0.01	0.65	50	NULL	NULL
CNN	32	Adadelta	500	0.01	0.65	50	NULL	NULL
RNN	32	Adadelta	500	0.01	0.65	50	NULL	NULL

4.3. Model Evaluation Metrics

The confusion matrix is a table commonly used in data science and machine learning to summarize the prediction results of classification models. It is represented by an n-by-n matrix, where records from the dataset are aggregated based on two criteria: 'True Class'

and ‘Predicted Class’, with n representing the number of classes in the dataset. Taking binary classification as an example, the structure of the confusion matrix is illustrated in Table 4.

Table 4. Confusion matrix.

	Total (P + N)	Predicted Condition	
		Positive (PP)	Negative (PN)
Actual condition	Positive (P) Negative (N)	True positive (TP) False positive (FP)	False negative (FN) True negative (TN)

- Accuracy is used to measure the prediction accuracy of a classification model, which is the ratio of correctly predicted samples to the total number of samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (19)$$

- Precision measures the model’s ability to correctly predict positive instances among the predicted positives, i.e., the ratio of true positive instances to the total number of predicted positive samples.

$$Precision = \frac{TP}{TP + FP} \quad (20)$$

- Recall measures the model’s ability to identify positive samples, i.e., the ratio of true positive instances to the total number of actual positive samples.

$$Recall = \frac{TP}{TP + FN} \quad (21)$$

- F1 Score: an evaluation metric that comprehensively considers both precision and recall, calculated as the harmonic mean of precision and recall.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (22)$$

4.4. Analysis and Discussion of Experimental Results

CTSF’s performance using different kernel functions is presented in Table 5, while the performance of the control group models is shown in Table 6.

Table 5. Performance of CTSF using different kernel functions.

	Linear			rbf			Sigmoid			Poly			
	PRE	RC	F1	PRE	RC	F1	PRE	RC	F1	PRE	RC	F1	
Normal	0.951	0.968	0.959	0.950	0.970	0.960	0.848	0.822	0.835	0.913	0.942	0.927	
C and C	1	0.995	0.997	0.981	1	0.990	0.985	0.958	0.972	0.995	1	0.997	
Exfiltration	1	0.998	0.999	1	0.997	0.998	0.929	0.988	0.958	0.997	0.997	0.997	
Exploitation	0.980	0.980	0.980	1	0.943	0.970	0.962	0.777	0.860	0.989	0.913	0.95	
Lateral_movement	0.996	0.991	0.993	0.994	0.994	0.994	0.882	0.891	0.887	0.989	0.994	0.991	
RDOS	0.994	0.998	0.996	1	0.996	0.997	0.998	1	0.999	1	0.983	0.991	
Reconnaissance	0.977	0.966	0.972	0.976	0.973	0.974	0.935	0.933	0.934	0.958	0.942	0.950	
Tampering	0.992	0.994	0.993	0.994	0.994	0.994	0.952	0.948	0.950	0.994	0.997	0.995	
Weaponization	0.996	0.994	0.995	0.998	1	0.999	0.986	0.972	0.979	0.989	0.996	0.992	
crypto-ransomware	0.667	1	0.800	1	1	1	0	0	0	0.5	1	0.667	
macro avg	0.960	0.990	0.970	0.990	0.990	0.99	0.85	0.83	0.84	0.93	0.98	0.95	
weighted avg	0.990	0.990	0.990	0.990	0.990	0.99	0.93	0.93	0.93	0.98	0.98	0.98	
Accuracy		0.98825			0.9885			0.9365			0.97925		
Training Time(s)		55,755.99			55,753.90			55,752.77			55,751.52		

Table 6. Performance of the control model.

	CNN-Transformer (Improved)			CNN-RNN			CNN			RNN		
	PRE	RC	F1	PRE	RC	F1	PRE	RC	F1	PRE	RC	F1
Normal	0.947	0.938	0.943	0.877	0.941	0.908	0.816	0.901	0.856	0.930	0.962	0.946
C and C	0.990	1	0.995	0.995	0.991	0.993	0.972	1	0.986	0.987	1	0.993
Exfiltration	1	1	1	0.998	0.996	0.997	0.994	0.998	0.996	1	1	1
Exploitation	0.964	0.987	0.976	0.962	0.793	0.870	0.915	0.606	0.729	0.962	0.906	0.934
Lateral_movement	0.994	0.991	0.993	0.994	0.989	0.992	0.976	0.976	0.976	0.988	0.996	0.992
RDOS	1	0.997	0.998	0.995	0.991	0.993	1	0.993	0.996	1	0.991	0.995
Reconnaissance	0.959	0.964	0.962	0.935	0.924	0.929	0.920	0.824	0.869	0.969	0.944	0.957
Tampering	1	0.997	0.998	0.997	0.991	0.994	0.977	0.991	0.984	1	1	1
Weaponization	0.996	0.998	0.997	0.998	1	0.999	0.934	0.998	0.965	1	1	1
crypto-ransomware	1	1	1	0	0	0	0	0	0	1	1	1
macro avg	0.99	0.99	0.99	0.88	0.86	0.87	0.85	0.83	0.84	0.98	0.98	0.98
weighted avg	0.99	0.99	0.99	0.97	0.97	0.97	0.95	0.95	0.95	0.98	0.98	0.98
Accuracy		0.9858			0.9728			0.9475			0.9847	
Training Time(s)		55,750			25,805			17,515			17,915	

The confusion matrices of CTSF using various kernel functions for classification are displayed in Figure 6. The confusion matrices of the other models in the comparative experiments are presented in Figure 7. In these matrices, ‘Exfiltration’, ‘Exploitation’, ‘Lateral_movement’, ‘Reconnaissance’, ‘Weaponization’, and ‘crypto-ransomware’ are abbreviated as ‘Exf’, ‘Exp’, ‘L_m’, ‘Rec’, ‘Wea’, and ‘c-r’, respectively, for clarity.

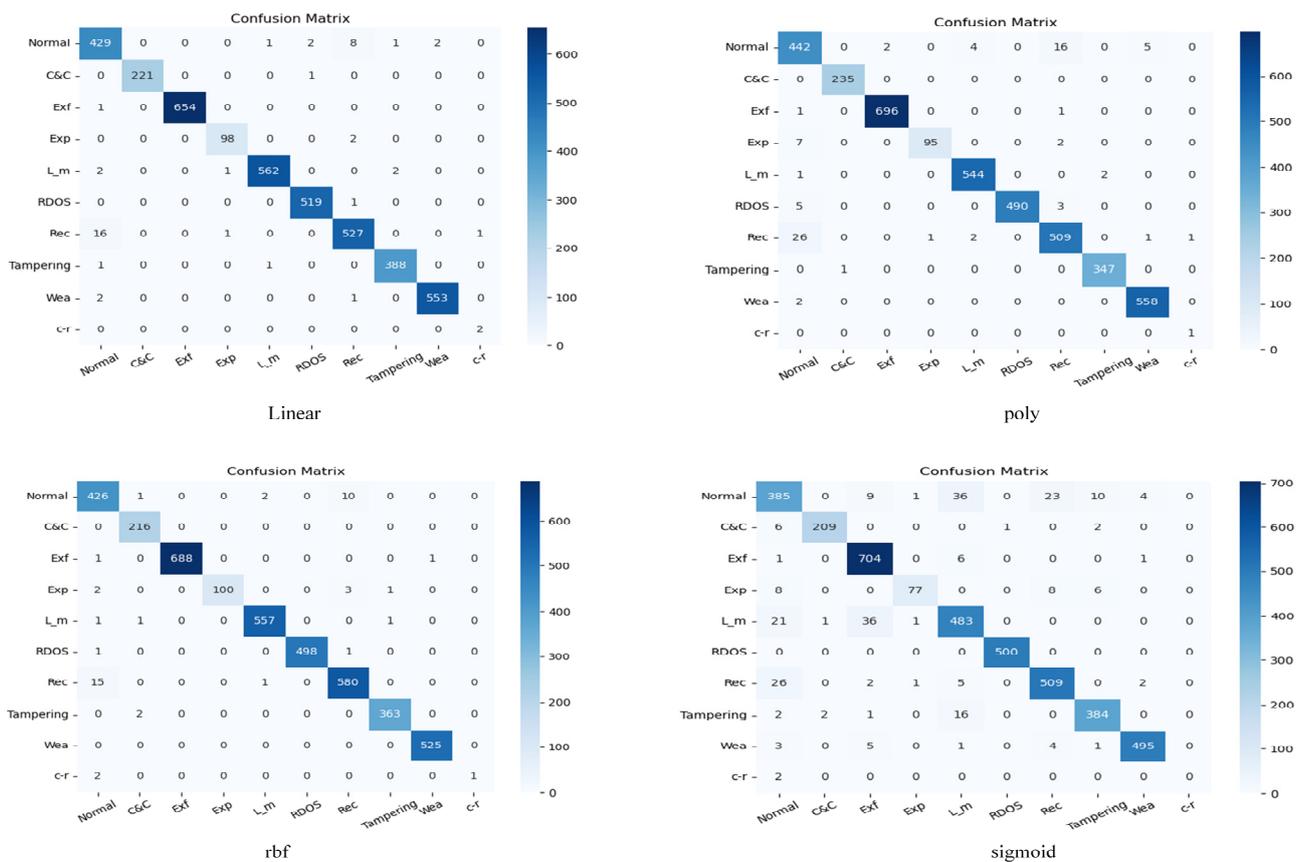


Figure 6. CTSF uses each kernel function to classify the confusion matrix.

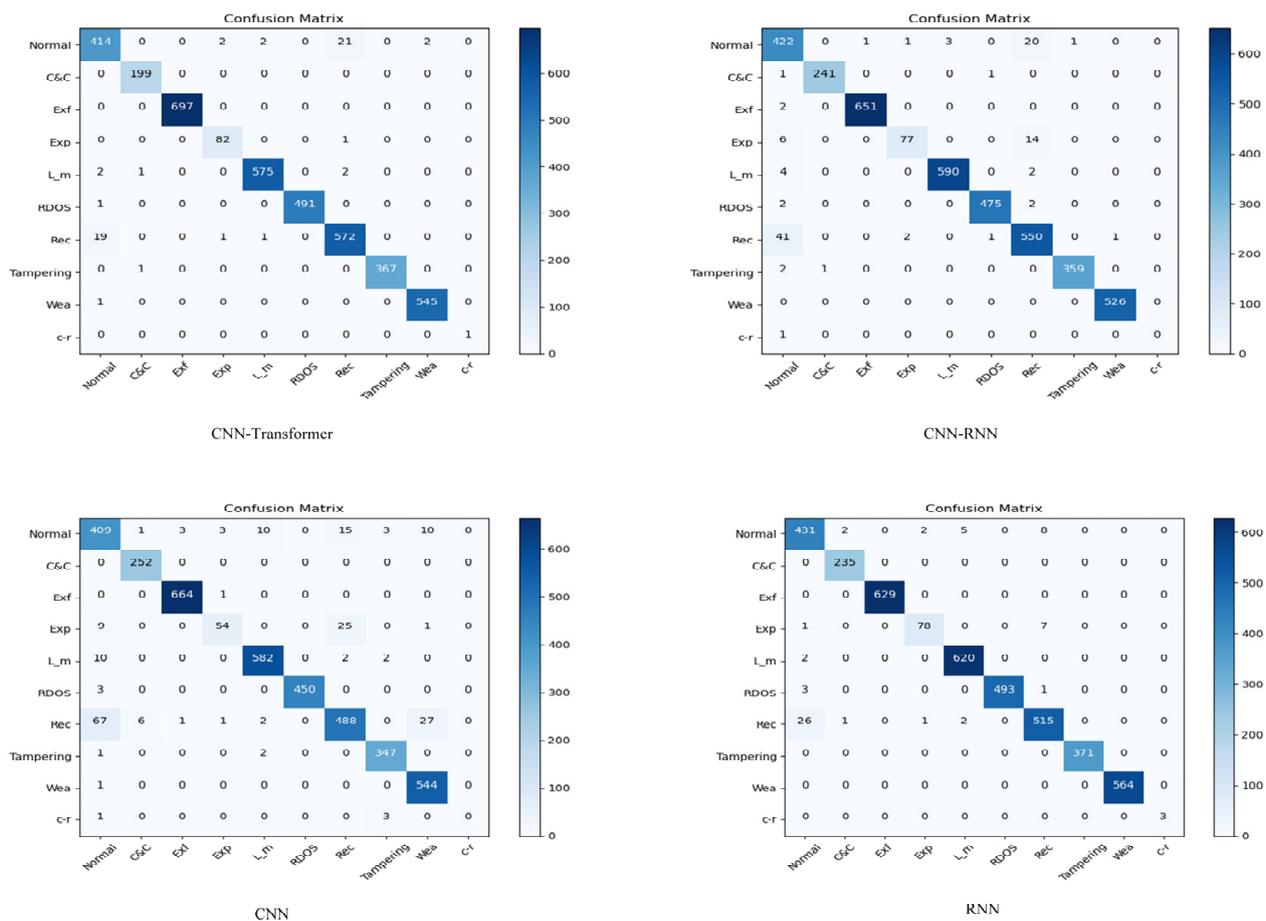


Figure 7. Confusion matrix of the model in the control experiment.

In Tables 5 and 6, the metrics Precision (PRE), Recall (RC), and F1-score (F1) are abbreviated. From Table 5, it can be concluded that the CTSF framework achieves the highest accuracy of 0.9885 when using the “rbf” kernel function and the lowest accuracy of 0.9365 when using the “sigmoid” kernel function. Figure 8 provides visualizations of CTSF’s performance using different kernel functions. The visualizations reveal that the decision boundary for the “Normal” class is less accurate when the “sigmoid” kernel function is used. In Table 4, the highest accuracy within the control groups is achieved by the CNN-Transformer (improved) model, with an accuracy of 0.9858. All three metrics related to the crypto-ransomware class reach 1, same as the results obtained by CTSF (with rbf); this indicates that the specialized design of the CTSF pretraining section equips the overall framework with a robust capability for detecting small-sample classes. Furthermore, the accuracy of the CNN-Transformer (improved) model is lower than that of CTSF using both the “rbf” and “linear” kernel functions. Since the parameters of CTSF’s pretraining section are identical to those of the control group’s CNN-Transformer model, this data corroborates the effectiveness of CTSF’s design (CTS first pretrains the initialization classification results and subsequently passes them to SVM for final classification). Furthermore, from Table 5, we can observe that CTSF, using the ‘rbf’ kernel function, took 55,753.90 s to train for 500 epochs. Table 6 displays the training time for the models in the comparative experimental group, with the fastest training time achieved by the CNN model at 17,515 s. In fact, from the above data, it can be observed that while CTSF exhibits advantages in recognizing small sample classes and overall classification accuracy, it is not lightweight. We will delve into this limitation in more detail in Section 5.

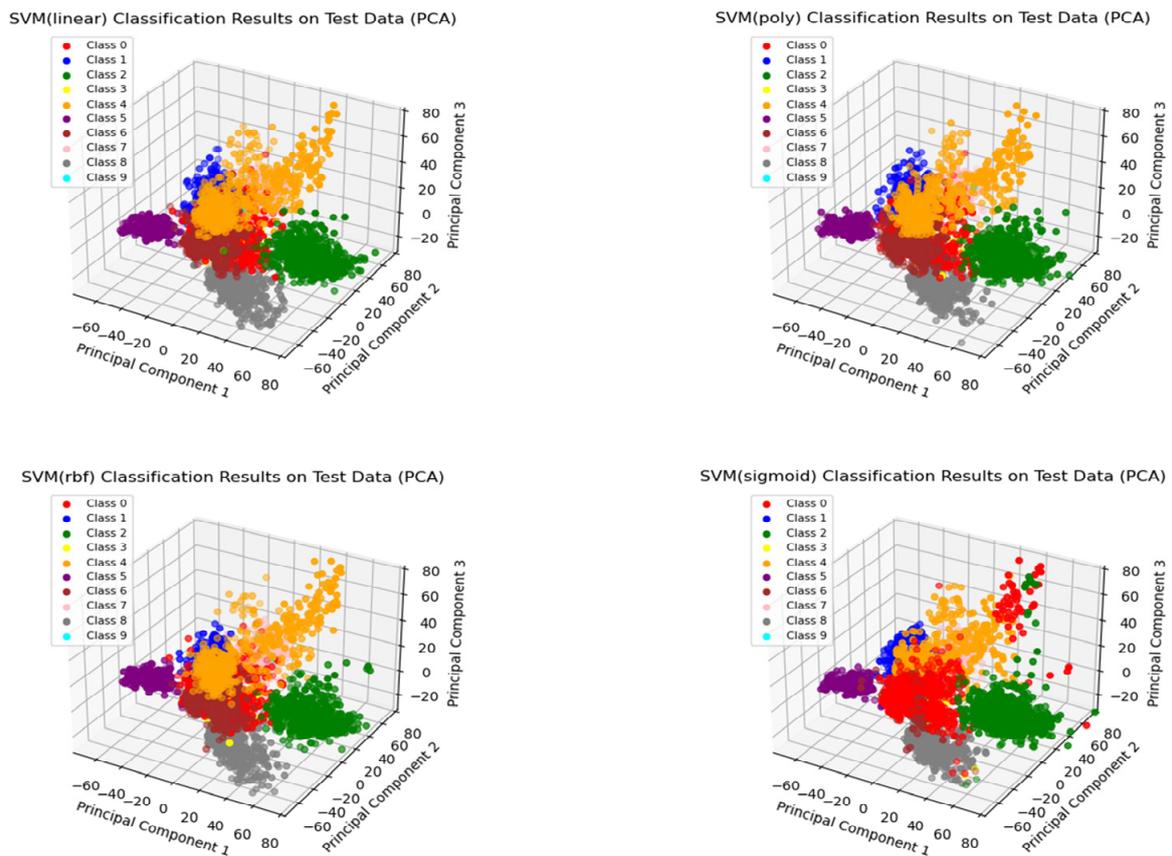


Figure 8. Visualization of the classification performance of CTSF with various kernel functions (Category numbers from 0 to 9 are: Normal, C and C, Exfiltration, Exploitation, Lateral_movement, RDOS, Reconnaissance, Tampering, Weaponization, crypto-ransomware).

Regarding the class “crypto-ransomware”, which contains only 21 instances in this dataset, the CNN-RNN and CNN models exhibit poor performance, highlighting their limitations in handling small-sample data. On the contrary, the CTSF, CNN-Transformer (improved), and RNN models, employing non-“sigmoid” kernels, perform better with this class, demonstrating their efficacy in handling small-sample data.

Furthermore, Table 5 indicates that with identical CNN parameters, CNN-Transformer (improved) outperforms CNN-RNN in terms of classification accuracy. This suggests that our proposed improved version of Transformer excels in capturing global features compared to RNN.

Finally, Table 7 presents a comparison between CTSF and various benchmark models [40] on different attack class detection rates in X-IIoTID datasets. As per Table 7, CTSF achieved a detection rate of 1 for the “C&C” class, significantly higher than other benchmark models. Similarly, the “Weaponization” and “crypto-ransomware” classes also achieved a detection rate of 1. Additionally, while CTSF’s detection rate for some classes may be slightly lower than that of some models, its detection rates exhibit smoother fluctuations for all classes, avoiding scenarios where a high detection rate for one class is accompanied by a low detection rate for another class. Lastly, for the small-sample class “crypto-ransomware”, CTSF’s detection rate surpasses that of all benchmark models, indicating its advantage in accurately detecting such classes.

Table 7. Comparison results of various benchmark models and CTSF in different attack class detection rates for X-IIoTID datasets.

Model	C and C	Exfiltration	Exploitation	Lateral_Movement	RDOS	Reconnaissance	Tampering	Weaponization	Crypto-Ransomware
DT	0.8966	0.8976	0.9852	0.9948	0.9999	0.9922	0.9947	0.9997	0.9986
NB	1	0.9823	0.5267	0.0834	0.9906	0.0150	0.9912	0.9867	0.9962
KNN	0.8096	0.7140	0.9128	0.9862	0.9999	0.9688	0.7868	0.9985	0.9960
SVM	0.8194	0.8394	0.8987	0.9983	0.9996	0.9170	0.9891	0.9996	0.9992
LR	0.5801	0.4634	0.7842	0.9809	0.9983	0.8666	0.7407	0.9907	0.9986
DNN	0.7716	0.9991	0.8129	0.9769	0.9996	0.9585	0.9815	0.9994	0.8188
GRU	0.8826	0.9993	0.8623	0.9789	0.9998	0.9886	0.9918	0.9997	0.9694
CTSF (with rbf)	1	0.997	0.943	0.994	0.996	0.973	0.994	1	1

5. Conclusions and Future Work

In this paper, we propose a novel network framework, named CTSF, for Industrial Internet intrusion detection. The CNN in the pretraining part of CTSF is employed to extract local features from input data as comprehensively as possible. The Transformer in the pretraining part, tailored for Industrial Internet intrusion detection datasets, is an improved version that abandons the decoder's self-attention mechanism due to the one-dimensional nature of label data in network traffic. This modification not only reduces the number of training parameters but also enhances the extraction of global features from input data. Ultimately, CTSF passes the well-pretrained data to SVM and selects an appropriate kernel function for classification.

To validate the efficacy of this framework in addressing intrusion-detection challenges in the context of Industrial Internet, we performed performance testing on a subset of X-IIoTID datasets, comparing it with advanced models as benchmarks. The experiments demonstrate that CTSF not only achieves a high accuracy but also accurately identifies suitable decision boundaries and correctly classifies instances, even in scenarios involving imbalanced or small sample data. Despite achieving high accuracy and excellent performance on small sample data, CTSF is not as lightweight in terms of computational resources. Heavier models typically require more computational resources to operate, which can potentially lead to higher hardware requirements, thereby increasing maintenance and operational costs. Furthermore, deploying less lightweight models into Industrial Internet systems may necessitate additional engineering resources, as they may require more extensive configuration and fine-tuning. Furthermore, the Industrial Internet environment also demands real-time capabilities, and the training and inference times of non-lightweight models can potentially become an issue. Therefore, looking ahead, we have the following considerations:

1. In CTSF, local and global features of data are extracted using CNN and an enhanced Transformer. While this paper has improved and optimized the traditional Transformer decoder for the characteristics of Industrial Internet traffic data, making it more lightweight, CTSF as a whole is still not sufficiently lightweight. Next, we plan to explore optimization algorithms to further enhance the pre-training phase of CTSF, aiming to reduce its resource consumption while maintaining its accuracy and detection rate.
2. In the pretraining phase, we can achieve parallel computing by adopting parallel training and parameter asynchronous updating methods. Distributing computational tasks to edge devices or within a distributed computing environment alleviates the burden on individual devices and reduces latency. Parallel and distributed computing can decompose the training tasks of large-scale models into multiple smaller tasks which are executed concurrently across various computing nodes, significantly reducing training time and enhancing training efficiency. It is worth noting that ensuring load balance among different nodes is a challenge in distributed computing. Some nodes may become busier than others, leading to performance imbalances. By taking into account both the benefits and challenges, and implementing appropriate opti-

mization strategies, such as employing more efficient communication protocols and using suitable load balancing algorithms, the scalability and performance of the CTSF framework can be improved.

- Utilizing pruning algorithms to reduce the CTSF's size. These algorithms can identify and remove connections or layers that contribute minimally to the CTSF's accuracy.

Author Contributions: Conceptualization, G.C., S.L. and Y.W.; methodology, G.C., S.L., Y.Y. and Y.W.; software, G.C.; validation, G.C. and S.L.; formal analysis, G.C., Y.Y. and S.L.; investigation, G.C., Y.W. and S.L.; resources, G.C., Y.W. and S.L.; data curation, G.C.; writing—original draft preparation, G.C.; writing—review and editing, G.C., Y.W., Y.Y. and S.L.; visualization, G.C.; supervision, G.C., Y.W. and S.L.; project administration, G.C. and S.L.; funding acquisition, S.L. and G.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the Provincial Universities Basic Business Expense Scientific Research Projects of Heilongjiang Province (2021-KYYWF-0179), the Science and Technology Project of Henan Province (212102310991), the Key Scientific Research Project of Henan Province (21A413001), and the Postgraduate Innovation Project of Harbin Normal University (HSDSSCX2023-10).

Data Availability Statement: The original data can be obtained by contacting the corresponding author.

Acknowledgments: Thanks to Shiming Li from Harbin Normal University for his guidance and assistance in this study. Special thanks to Yu Yang for formatting and language polishing this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Li, J.Q.; Yu, F.R.; Deng, G.; Luo, C.; Ming, Z.; Yan, Q. Industrial internet: A survey on the enabling technologies, applications, and challenges. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1504–1526. [[CrossRef](#)]
- Qin, W.; Chen, S.; Peng, M. Recent advances in Industrial Internet: Insights and challenges. *Digit. Commun. Netw.* **2020**, *6*, 1–13. [[CrossRef](#)]
- Thakkar, A.; Lohiya, R. A survey on intrusion detection system: Feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artif. Intell. Rev.* **2022**, *55*, 453–563. [[CrossRef](#)]
- Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 20. [[CrossRef](#)]
- Ahmad, Z.; Shahid Khan, A.; Wai Shiang, C.; Abdullah, J.; Ahmad, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4150. [[CrossRef](#)]
- Saranya, T.; Sridevi, S.; Deisy, C.; Chung, T.D.; Khan, M.A. Performance analysis of machine learning algorithms in intrusion detection system: A review. *Procedia Comput. Sci.* **2020**, *171*, 1251–1260. [[CrossRef](#)]
- Maseer, Z.K.; Yusof, R.; Bahaman, N.; Mostafa, S.A.; Foozy, C.F.M. Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset. *IEEE Access* **2021**, *9*, 22351–22370. [[CrossRef](#)]
- Almomani, O.; Almaiah, M.A.; Alsaaidah, A.; Smadi, S.; Mohammad, A.H.; Althunibat, A. Machine learning classifiers for network intrusion detection system: Comparative study. In Proceedings of the 2021 International Conference on Information Technology (ICIT), Amman, Jordan, 14–15 July 2021; pp. 440–445.
- Halimaa, A.; Sundarakantham, K. Machine learning based intrusion detection system. In Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 23–25 April 2019; pp. 916–920.
- Mohammadi, M.; Rashid, T.A.; Karim, S.H.T.; Aldalwie, A.H.M.; Tho, Q.T.; Bidaki, M.; Rahmani, A.M.; Hosseinzadeh, M. A comprehensive survey and taxonomy of the SVM-based intrusion detection systems. *J. Netw. Comput. Appl.* **2021**, *178*, 102983. [[CrossRef](#)]
- Ngueajio, M.K.; Washington, G.; Rawat, D.B.; Ngueabou, Y. Intrusion Detection Systems Using Support Vector Machines on the KDDCUP'99 and NSL-KDD Datasets: A Comprehensive Survey. In *Intelligent Systems and Applications, Proceedings of the 2022 Intelligent Systems Conference (IntelliSys), Amsterdam, The Netherlands, 1–2 September 2022*; Springer International Publishing: Cham, Switzerland, 2022; Volume 2, pp. 609–629.
- Kocher, G.; Kumar, G. Machine learning and deep learning methods for intrusion detection systems: Recent developments and challenges. *Soft Comput.* **2021**, *25*, 9731–9763. [[CrossRef](#)]
- LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [[CrossRef](#)]
- Shrestha, A.; Mahmood, A. Review of deep learning algorithms and architectures. *IEEE Access* **2019**, *7*, 53040–53065. [[CrossRef](#)]

15. Aleesa, A.M.; Zaidan, B.B.; Zaidan, A.A.; Sahar, N.M. Review of intrusion detection systems based on deep learning techniques: Coherent taxonomy, challenges, motivations, recommendations, substantial analysis and future directions. *Neural Comput. Appl.* **2020**, *32*, 9827–9858. [[CrossRef](#)]
16. Han, K.; Xiao, A.; Wu, E.; Guo, J.; Xu, C.; Wang, Y. Transformer in transformer. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 15908–15919.
17. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
18. Bhatt, D.; Patel, C.; Talsania, H.; Patel, J.; Vaghela, R.; Pandya, S.; Modi, K.; Ghayvat, H. CNN variants for computer vision: History, architecture, application, challenges and future scope. *Electronics* **2021**, *10*, 2470. [[CrossRef](#)]
19. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaria, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 1–74. [[CrossRef](#)] [[PubMed](#)]
20. Yuan, F.; Zhang, Z.; Fang, Z. An effective CNN and Transformer complementary network for medical image segmentation. *Pattern Recognit.* **2023**, *136*, 109228. [[CrossRef](#)]
21. Mirlekar, S.; Kanojia, K.P. A Comprehensive Study on Machine Learning Algorithms for Intrusion Detection System. In Proceedings of the 2022 10th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-22), Nagpur, India, 29–30 April 2022; pp. 1–6.
22. Almaiah, M.A.; Almomani, O.; Alsaaidah, A.; Al-Otaibi, S.; Bani-Hani, N.; Hwaitat, A.K.A.; Al-Zahrani, A.; Lutfi, A.; Awad, A.B.; Aldhyani, T.H. Performance Investigation of Principal Component Analysis for Intrusion Detection System Using Different Support Vector Machine Kernels. *Electronics* **2022**, *11*, 3571. [[CrossRef](#)]
23. Saheed, Y.K.; Arowolo, M.O.; Tosho, A.U. An Efficient Hybridization of K-Means and Genetic Algorithm Based on Support Vector Machine for Cyber Intrusion Detection System. *Int. J. Electr. Eng. Inform.* **2022**, *14*, 426–442. [[CrossRef](#)]
24. Khairandish, M.O.; Sharma, M.; Jain, V.; Chatterjee, J.M.; Jhanjhi, N.Z. A hybrid CNN-SVM threshold segmentation approach for tumor detection and classification of MRI brain images. *IRBM* **2022**, *43*, 290–299. [[CrossRef](#)]
25. Tao, T.; Wei, X. A hybrid CNN-SVM classifier for weed recognition in winter rape field. *Plant Methods* **2022**, *18*, 29. [[CrossRef](#)]
26. Krichen, M. Convolutional neural networks: A survey. *Computers* **2023**, *12*, 151. [[CrossRef](#)]
27. Alahmari, F.; Naim, A.; Alqahtani, H. E-Learning Modeling Technique and Convolution Neural Networks in Online Education. In *IoT-Enabled Convolutional Neural Networks: Techniques and Applications*; River Publishers: Aalborg, Denmark, 2023; pp. 261–295.
28. Pingale, S.V.; Sutar, S.R. Remora whale optimization-based hybrid deep learning for network intrusion detection using CNN features. *Expert Syst. Appl.* **2022**, *210*, 118476. [[CrossRef](#)]
29. El-Ghamry, A.; Darwish, A.; Hassaniien, A.E. An optimized CNN-based intrusion detection system for reducing risks in smart farming. *Internet Things* **2023**, *22*, 100709. [[CrossRef](#)]
30. Qazi, E.U.H.; Almorjan, A.; Zia, T. A One-Dimensional Convolutional Neural Network (1D-CNN) Based Deep Learning System for Network Intrusion Detection. *Appl. Sci.* **2022**, *12*, 7986. [[CrossRef](#)]
31. Halbouni, A.H.; Gunawan, T.S.; Halbouni, M.; Assaig, F.A.A.; Effendi, M.R.; Ismail, N. CNN-IDS: Convolutional Neural Network for Network Intrusion Detection System. In Proceedings of the 2022 8th International Conference on Wireless and Telematics (ICWT), Yogyakarta, Indonesia, 21–22 July 2022; pp. 1–4.
32. Xia, L.; Mi, S.; Zhang, J.; Luo, J.; Shen, Z.; Cheng, Y. Dual-Stream Feature Extraction Network Based on CNN and Transformer for Building Extraction. *Remote Sens.* **2023**, *15*, 2689. [[CrossRef](#)]
33. Wu, Z.; Zhang, H.; Wang, P.; Sun, Z. RTIDS: A robust transformer-based approach for intrusion detection system. *IEEE Access* **2022**, *10*, 64375–64387. [[CrossRef](#)]
34. Wang, M.; Yang, N.; Weng, N. Securing a Smart Home with a Transformer-Based IoT Intrusion Detection System. *Electronics* **2023**, *12*, 2100. [[CrossRef](#)]
35. Liu, Y.; Wu, L. Intrusion Detection Model Based on Improved Transformer. *Appl. Sci.* **2023**, *13*, 6251. [[CrossRef](#)]
36. Tan, M.; Iacovazzi, A.; Cheung, N.M.M.; Elovici, Y. A neural attention model for real-time network intrusion detection. In Proceedings of the 2019 IEEE 44th Conference on Local Computer Networks (LCN), Osnabrueck, Germany, 14–17 October 2019; pp. 291–299.
37. Khan, A.; Rauf, Z.; Sohail, A.; Rehman, A.; Asif, H.; Asif, A.; Farooq, U. A survey of the Vision Transformers and its CNN-Transformer based Variants. *arXiv* **2023**, arXiv:2305.09880.
38. Yao, R.; Wang, N.; Chen, P.; Ma, D.; Sheng, X. A CNN-transformer hybrid approach for an intrusion detection system in advanced metering infrastructure. *Multimed. Tools Appl.* **2023**, *82*, 19463–19486. [[CrossRef](#)]
39. Luo, S.; Zhao, Z.; Hu, Q.; Liu, Y. A hierarchical CNN-transformer model for network intrusion detection. In Proceedings of the 2nd International Conference on Applied Mathematics, Modelling, and Intelligent Computing (CAMMIC 2022), Kunming, China, 25–27 March 2022; Volume 12259, pp. 853–860.
40. Al-Hawawreh, M.; Sitnikova, E.; Aboutorab, N. X-IloTID: A connectivity-agnostic and device-agnostic intrusion data set for industrial Internet of Things. *IEEE Internet Things J.* **2021**, *9*, 3962–3977. [[CrossRef](#)]
41. Alanazi, R.; Aljuhani, A. Anomaly Detection for Industrial Internet of Things Cyberattacks. *Comput. Syst. Sci. Eng.* **2023**, *44*, 2361–2378. [[CrossRef](#)]

42. Aftab, S.; Shah, Z.S.; Memon, S.A.; Shaikh, Q. A machine-learning-based Intrusion detection for IIoT infrastructure. In Proceedings of the 2023 7th International Multi-Topic ICT Conference (IMTIC), Jamshoro, Pakistan, 10–12 May 2023; pp. 1–6.
43. Alenezi, N.; Aljuhani, A. Intelligent Intrusion Detection for Industrial Internet of Things Using Clustering Techniques. *Comput. Syst. Sci. Eng.* **2023**, *46*, 2899–2915. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.