

An Efficient Detection Mechanism of Network Intrusions in IoT Environments Using Autoencoder and Data Partitioning

Xiao, Yiran

Graduate School of Information Science and Electrical Engineering, Kyushu University

Feng, Yaokai

Faculty of Information Science and Electrical Engineering, Kyushu University

Sakurai, Kouichi

Faculty of Information Science and Electrical Engineering, Kyushu University

<https://hdl.handle.net/2324/7337635>

出版情報 : Computers. 13 (10), pp.269–, 2024-10-14. Multidisciplinary Digital Publishing

Institute : MDPI

バージョン :

権利関係 : © 2024 by the authors.



Article

An Efficient Detection Mechanism of Network Intrusions in IoT Environments Using Autoencoder and Data Partitioning

Yiran Xiao ^{1,*}, Yaokai Feng ^{2,*}  and Kouichi Sakurai ^{2,*}

¹ Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 8190395, Japan

² Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 8190000, Japan

* Correspondence: yiranxiao320@outlook.com (Y.X.); fengyk@ait.kyushu-u.ac.jp (Y.F.); sakurai@inf.kyushu-u.ac.jp (K.S.)

Abstract: In recent years, with the development of the Internet of Things and distributed computing, the “server-edge device” architecture has been widely deployed. This study focuses on leveraging autoencoder technology to address the binary classification problem in network intrusion detection, aiming to develop a lightweight model suitable for edge devices. Traditional intrusion detection models face two main challenges when directly ported to edge devices: inadequate computational resources to support large-scale models and the need to improve the accuracy of simpler models. To tackle these issues, this research utilizes the Extreme Learning Machine for its efficient training speed and compact model size to implement autoencoders. Two improvements over the latest related work are proposed: First, to improve data purity and ultimately enhance detection performance, the data are partitioned into multiple regions based on the prediction results of these autoencoders. Second, autoencoder characteristics are leveraged to further investigate the data within each region. We used the public dataset NSL-KDD to test the behavior of the proposed mechanism. The experimental results show that when dealing with multi-class attacks, the model’s performance was significantly improved, and the accuracy and F1-Score were improved by 3.5% and 2.9%, respectively, maintaining its lightweight nature.

Keywords: autoencoder; network intrusion detection; model accuracy improvement; Extreme Learning Machine



Citation: Xiao, Y.; Feng, Y.; Sakurai, K. An Efficient Detection Mechanism of Network Intrusions in IoT Environments Using Autoencoder and Data Partitioning. *Computers* **2024**, *13*, 269. <https://doi.org/10.3390/computers13100269>

Academic Editor: Lilatul Ferdouse

Received: 11 August 2024

Revised: 6 October 2024

Accepted: 9 October 2024

Published: 14 October 2024



Copyright: © 2024 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

1.1. Motivation

IoT (Internet of Things) refers to a network of interconnected devices facilitating communication between devices themselves and with the cloud. With advancements in computer hardware and increasing communication bandwidth, IoT has permeated daily life. IoT technology enables smart applications across various domains, significantly enhancing productivity. Examples include integrated digital infrastructure in smart cities, vehicle traffic monitoring in intelligent transportation systems, and automated control of household devices in smart homes, among others [1]. Although often unnoticed, IoT has indeed infiltrated various fields.

Despite its ubiquitous presence, IoT development has not always been benign. According to Gartner, approximately 20% of enterprises or related entities experienced at least one IoT-based attack between 2015 and 2017 [2]. Some of these attacks had significant impact, such as the Mirai botnet, predominantly comprising embedded and IoT devices, which launched large-scale attacks in September 2016, crippling several prominent websites. Within the first 20 h, the above-mentioned Mirai botnet infected nearly 65,000 IoT devices, stabilizing at 200,000 to 300,000 infected devices, marking one of the largest recorded attacks [3]. Additionally, incidents like the BlackEnergy attack on Ukraine’s power grid [4],

the Stuxnet attack on centrifuges in Iran's nuclear facilities [5], and the exploitation of baby monitors for household surveillance [6] underscore ongoing security challenges.

To bolster IoT device security, reliable intrusion detection systems (IDS) are essential. IDS generally fall into two categories: (1) signature-based IDS and (2) anomaly-based IDS. Signature-based IDS use predefined signatures or patterns to detect known attack types, triggering alerts when network traffic or behavior matches these signatures. Anomaly-based IDS, on the other hand, establish baselines of normal behavior and detect deviations indicative of potential attacks, often leveraging machine learning to identify anomalous behavior.

Compared to anomaly-based IDS, signature-based IDS typically offer higher detection accuracy but are limited to known attacks, rendering them ineffective against unknown threats. Moreover, signature-based IDS require continual updates to their signature databases to combat new threats, which can lead to system bloat and increased response times. Lastly, the manual intervention required for signature-based IDS is impractical in IoT contexts [7]. Therefore, this study opts for an anomaly-based IDS approach using machine learning to detect attacks.

The next challenge is selecting appropriate machine learning models for intrusion detection, considering the constraints of IoT edge devices [8]: limited memory and computational resources make complex model execution difficult. Additionally, IoT devices generate sparse network traffic compared to common devices, often triggered by infrequent user interactions, and bandwidth constraints are common. Furthermore, some IoT applications require rapid response times. Hence, lightweight yet accurate models are crucial for deployment on IoT devices.

1.2. Background

1.2.1. Autoencoder and Its Application to Intrusion Detection

After thorough research, we decided to explore the potential of using autoencoders as the core component of an intrusion detection system (IDS) in an IoT environment. The rationale is twofold: first, autoencoders excel in handling binary clustering tasks, which aligns perfectly with the need to classify data into normal and anomalous categories. Second, numerous studies have confirmed the important role of autoencoders in the field of attack detection, which will be illustrated with examples in the following sections. Next, we will introduce basic information about autoencoders and the steps for their implementation.

An autoencoder is a neural network with the learning objective of making the output identical to the input. Its structure is divided into two parts: the encoder and the decoder. As shown in Figure 1, the encoder compresses the data into a low-dimensional space and the decoder restores it to reconstruct the original data. Because the autoencoder can effectively extract features from the data during this process, early autoencoders were often used for data compression and feature learning. After the backpropagation (BP) algorithm was proposed [9], the autoencoder algorithm, as one of the implementations of BP, also gained attention. The formal introduction of autoencoders as a type of neural network structure came from Yann LeCun's research published in 1987 [10]. Since then, autoencoders have continuously evolved and have spawned many variants. Today, one of the uses of autoencoders is for unsupervised learning, handling binary clustering tasks.

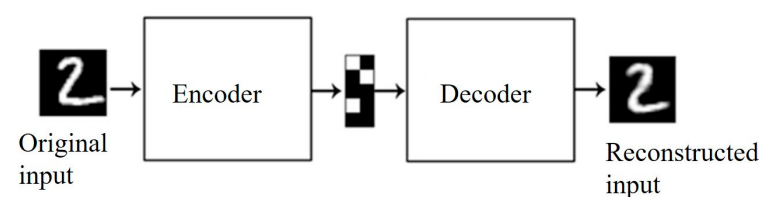


Figure 1. Schematic diagram of the autoencoder.

The steps for using an autoencoder for attack detection are as follows: First, train the autoencoder using normal data. Then, input the data to be detected and calculate the reconstruction error. Generally, for normal data the reconstruction error is small, while for attack data the reconstruction error is large. Based on this reconstruction error, if it exceeds a pre-set threshold, the data are identified as attack data. For instance, Hyunseung Choi et al. utilized four models—basic autoencoder, denoising autoencoder, stacked autoencoder, and variational autoencoder—to train the training data and classify normal and abnormal data by setting thresholds [11]. Autoencoders can also be combined with other models. For example, Cosimo Ieracitano et al. proposed a deep classifier based on autoencoders [12], where the low-dimensional feature vectors obtained from the encoding stage are fed into a dense, fully connected layer, and the Softmax activation function is used to classify the data as normal or abnormal. It is evident that autoencoders play a significant role in the field of attack detection.

1.2.2. On-Device Learning Anomaly Detector

The next problem to be solved is how to construct a lightweight autoencoder to make it suitable for the IoT environment. The paper of ONLAD [13] and its underlying component, the Extreme Learning Machine, have given us inspiration.

ONLAD (On-Device Learning Anomaly Detector) is an autoencoder model for detecting network intrusions that employs ELM (Extreme Learning Machine) as its main component.

ELM is a machine learning algorithm for single-hidden layer feedforward neural networks, proposed by Guang-Bin Huang et al. in 2004 [14]. Unlike traditional gradient descent methods that iteratively update all weight parameters, ELM randomly selects input weights and analytically determines output weights using matrix inversion. This approach avoids issues such as overfitting and local minima associated with gradient descent methods, significantly improving learning speed and generalization performance. Additionally, since ELM has only one hidden layer, the model is relatively small in scale. It can be said that ELM is a lightweight machine learning model in terms of both time and space efficiency.

Figure 2 illustrates the basic architecture of ONLAD. In the figure, α represents the input weights, β is the output weights, b is the bias vector, and G is the activation function. The objective of ONLAD is to generate the output y from the input x through the model and make y as close as possible to the target t . Since ONLAD operates as an autoencoder, t is the same as x . The process of generating y can be represented by the following formula:

$$\hat{y} = G(\alpha x + b)\beta, \quad (1)$$

After α and b are randomly initialized, they are not changed. The parameter β is continuously updated. Let H denote the output of the hidden layer, which is $G(\alpha x + b)$. The optimal output weights β are calculated as follows:

$$\hat{\beta} = H^+ t, \quad (2)$$

Here, H^+ is the pseudoinverse of H .

In the i -th round of the learning process, the calculation of β_i for updating the model parameters is as follows:

$$\begin{aligned} P_i &= P_{i-1} - P_{i-1} \cdot H_i^T (I + H_i P_{i-1} H_i^T)^{-1} H_i P_{i-1}, \\ \beta_i &= \beta_{i-1} + P_i H_i^T (t_i + H_i \beta_{i-1}), \end{aligned} \quad (3)$$

In this context, P_i is the covariance matrix of the i -th iteration. Specifically, at the very beginning, $p_0 = (H_0^T H_0)^{-1}$ and $\beta_0 = P_0 H_0^T t_0$.

Since ELM contains only a single hidden layer, its model is relatively lightweight, and it updates weights using an analytical solution (matrix pseudoinverse) instead of gradient descent, resulting in faster training speed. This makes it more suitable for IoT environments with limited computing resources. Additionally, ELM has a strong generalization ability, does not overfit, and avoids becoming stuck in local minima.

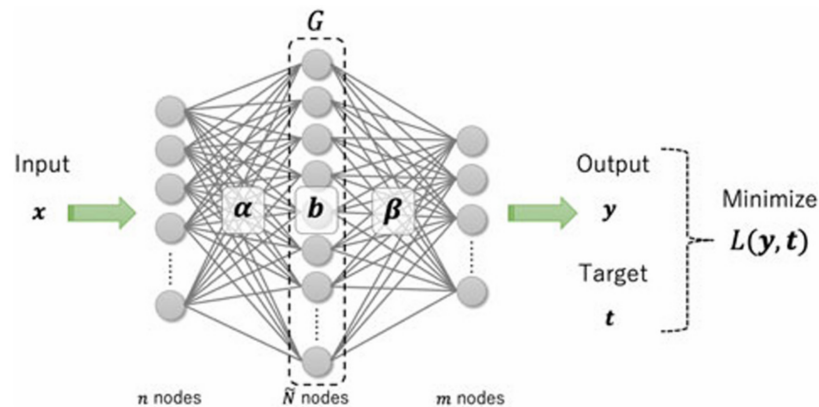


Figure 2. ELM architecture diagram.

1.2.3. Multiple Autoencoders Joint Decision-Making

In the above ONLAD scheme, researchers trained an autoencoder using normal data and used this autoencoder for intrusion detection. However, different types of attacks have different sensitivities to different features. For example, a feature can contain a lot of information about DoS attacks but almost no information about probe attacks. In this case, if the same features are used to train the model indiscriminately, the model's generalization ability will be poor. The solution is to train a separate autoencoder model for each type of attack. This way, each autoencoder can focus on detecting a specific type of attack and be trained based on the best features for that type of attack.

Therefore, in the paper [15], faced with the NSL-KDD dataset [16] (which includes normal data and four types of attacks: DDoS, Probe, R2L, and U2R), four corresponding autoencoders were trained for each type of attack. Each autoencoder was trained using the same normal data but with different features. The specific features used by each autoencoder are shown in Table 1.

Table 1. Feature selection for each type of attack data.

Type of Attack Data	Selected Feature
DoS	protocol_type, flag, wrong_fragment, num_compromised, root_shell, num_shells, same_srv_rate
Probe	logged_in, num_root, num_shells, in_host_login, srv_error_rate, srv_serror_rate, srv_diff_host_rate
R2L	service, urgent, num_root, num_shells, num_access_files, is_guest_login, srv_error_rate
U2R	flag, land, hot, num_failed_login, logged_in, num_compromised, su_attempted, num_root, num_file_creations, num_outbound_cmds, is_guest_login, same_srv_rate, dst_host_srv_count

When all four autoencoders determine the data to be normal, the detection result is normal. If any one of the autoencoders determines the data to be abnormal, the data are classified as abnormal, triggering an alert.

This tailored improvement makes the model more adept at handling complex, non-single-type attacks in network traffic.

1.3. Contributions

The main contributions of this work can be briefly summarized as follows:

- Conducted research on the papers related to the field of intrusion detection, identified their shortcomings, and improved the schemes of relevant papers.
- Introduced the concept of paired autoencoders, where an autoencoder trained on attack data is paired with another trained on normal data.
- Partitioned the data into multiple regions to reduce the complexity of data distribution in each region and improve the detection performance. For this purpose, multiple autoencoders are used for initial data prediction, and based on the prediction results, the data are partitioned.
- Leveraged the threshold characteristics of the autoencoders to precisely detect data types within each region.
- Validated our proposal on the public dataset (NSL-KDD), even when dealing with traffic that contains a mix of different types of attacks.
- Boldly introduced the use of autoencoders trained on positive data to assist in the collaborative judgment of autoencoders trained on negative data and mitigate the potential side effects (such as an increased false positive rate) through data partitioning and secondary detection. According to our investigation, this approach is unprecedented.

The structure of the subsequent content is as follows: In Section 2, we discuss the reviewed relevant literature and clarify the research goals. In Section 3, we explain the research proposal in detail. In Section 4, we present the experimental setup, procedures, and results and analyze the results. Finally, in Section 5, we summarize the whole paper and outline future research directions.

2. Related Work

2.1. Literature Review

Before formally introducing the content of this research, this section will elaborate on some relevant studies concerning network intrusion detection systems in IoT environments that have been investigated in recent years.

Intrusion detection systems (IDS) can generally be classified into two types: signature-based and anomaly-based. In traditional network security, signature-based IDS are more common, such as firewalls on personal PCs. Naturally, they also have their applications in IoT environments. In a paper by Philokypros P. Iouliauou et al. [17], a novel signature-based IDS is proposed. This system consists of IDS sensors deployed near the sensor end and IDS routers responsible for running detection modules and firewalls. The IDS sensors monitor and report suspicious activities to the IDS routers, which match the forwarded packets with malicious signatures and establish firewall rules based on the matching results.

Considering the possibility of internal attacks in collaborative intrusion detection systems (CIDS) in IoT environments, Li et al. proposed a consensus framework combining blockchain technology and signature-based IDS, called CBSigIDS (collaborative blockchained signature-based intrusion detection system) [18]. They assume that in this scenario, attackers have the opportunity to control one or more nodes in the CIDS. To address this, each IDS node identifies attacks and periodically shares a set of signatures encrypted with its private key with other nodes. Before accepting these signatures, other nodes verify them against their local databases. Thus, by using blockchain technology, CBSigIDS provides a verifiable signature-sharing method for CIDS without the need for a trusted intermediary.

Nazim Uddin Sheikh et al. proposed a pattern-matching algorithm to compare the DNA sequences of data to be detected with signatures in the signature database [19]. Simply put, this algorithm compares the session DNA sequences with signatures, calculating a similarity score. If the similarity exceeds a preset threshold, the session is marked as an attack.

The above are all signature-based IDS, but this type of IDS has its own drawbacks: firstly, it cannot detect unknown attacks; secondly, IoT edge devices may not be able to support large signature databases. Therefore, more people choose anomaly-based IDS as a solution for IoT environments, with machine learning being widely studied as an implementation method. Wai Weng Lo et al. proposed a novel network intrusion detection system called E-GraphSAGE [20], which is based on a GNN (graph neural network). The GraphSAGE method can capture edge features and topological information of the graph, achieving edge classification to detect malicious network traffic. This method has achieved good results on four NIDS benchmark datasets, such as BoT-IoT.

For GNN-based IDS in IoT environments, Zhou et al. proposed a new hierarchical adversarial attack generation method [21]. This method uses salient graph technology to identify key elements in the feature space and generates adversarial samples by minimally perturbing these key elements. Additionally, they used a hierarchical node selection algorithm based on random walks to find the most vulnerable nodes in IoT as attack targets. The combination of these two algorithms reduces the detection accuracy of two state-of-the-art GNN models by 30%.

Muder Almiani et al. proposed a network intrusion detection model in a fog computing environment [22]. This model adopts a two-layer detection structure, each layer using deep recurrent neural networks with different internal structures and parameter settings. The first layer mainly detects DoS attacks, while the second layer filters out hard-to-detect attacks such as R2L and U2R.

In the context of smart homes, Wang et al. proposed an intrusion detection system based on Transformer [23]. The method used by this system utilizes a self-attention mechanism to learn contextual embeddings of network features, capable of handling both continuous and categorical features simultaneously. It achieved good results on the ToN IoT dataset, with 97.95% accuracy for binary classification and 95.78% for multi-class classification.

Salam Fraihat et al. compared four feature selection algorithms—AOA, WSO, GWO, and BAT—in their paper [24] and combined them with various machine learning models such as RF (random forest) [25], NB (naive bayes), and DT (decision tree) [26]. They ultimately selected the optimal combination to build an IDS suitable for large-scale IoT networks.

Most of the machine learning models used in the above studies are supervised learning models. However, the preliminary research for this study, which is based on unsupervised learning models such as autoencoders, is described in fewer papers [13,15]. The research in these two papers has been thoroughly discussed in Section 1.2. In addition, we also investigated the application of stacked autoencoders. Although they may not be specifically used in the field of network intrusion detection, they can still serve as a reference. In their paper [27], Javaria Amin et al. introduced a deep learning model for brain tumor detection. They used two layers of sparse autoencoders (SAEs) for feature extraction and performed classification through a softmax layer. The model demonstrated superior performance on the BRATS dataset. However, this approach may not be suitable for all situations. First, the SAEs contain two hidden layers with 200 and 400 units, respectively, which makes the model large and difficult to deploy in IoT environments. Secondly, this model uses the softmax function for classification, which may be less effective for network traffic data with more ambiguous classification boundaries.

Similarly, in paper [28], Lukáš Váreka et al. used stacked autoencoders to complete the P300 detection task in brain-computer interfaces (BCIs) and compared the performance of SAEs, LDA, and MLP. However, similarly, because this SAE model has four hidden layers, and the number of units in each hidden layer varies from 50 to 130, it still appears to be relatively large in the IoT environment.

The content, methods, and limitations of the literature are uniformly recorded and summarized in Table 2.

Table 2. Literature summary table.

Research	Model/Method	Limitations
Novel signature-based IDS [17]	Composed of IDS sensors near the sensor end and IDS routers responsible for running detection modules and firewalls	Signature-based methods cannot detect unknown attacks. Moreover, in the IoT environment, edge devices may not be able to support large signature databases.
CBSigIDS (collaborative blockchained signature-based intrusion detection system) [18]	Block-chain	
DNA sequence pattern matching algorithm [19]		
E-GraphSAGE (based on a GNN) [20]	GNN (graph neural network)	
New hierarchical adversarial attack generation method [21]	GNN	All the methods adopted are supervised learning.
Network intrusion detection model in fog computing environment [22]	Two-layer detection structure with deep recurrent neural networks of different internal structures and parameter settings	
Intrusion detection system for smart homes based on Transformer [23]	Transformer	
Combining feature selection algorithms (AOA, WSO, GWO, BAT) with machine learning models like RF, NB, DT [24]	RF, NB, DT	
Deep learning model for brain tumor detection [27]	Two layers of sparse autoencoders (SAEs) and softmax layer	Large models are difficult to deploy in IoT environments. The softmax function may be less effective for network traffic data with ambiguous classification boundaries.
Stacked autoencoders for P300 detection in brain-computer interfaces [28]	Stacked autoencoders	Four hidden layers and a varying number of units make it relatively large in IoT environments.
This study	(1) a group of autoencoders is trained with attack data and another group of autoencoders is trained with normal data (2) the data are partitioned using these two groups of autoencoders (3) an autoencoder is selected for each region	

2.2. Challenges in This Study

To develop a network intrusion detection system suitable for IoT environments, we propose two key requirements: first, the model should be as lightweight as possible; second, the model's detection accuracy should be maximized. The first requirement can be achieved using the Extreme Learning Machine mentioned earlier, as long as no significant additional load is added to the model that would drastically increase time or space complexity. However, the previous research [15] mentioned in Section 1.2 has shown shortcomings in addressing the second requirement. Ideally, the distribution of data in the feature space should have clearly defined boundaries between normal and attack data. However, due to the complexity of network traffic, in most scenarios, different types of data often overlap, making it difficult to accurately detect the anomalies. Therefore, this study focuses on

reducing the complexity of data distribution in the feature space, aiming to improve the final detection performance of the model.

3. Proposal: Paired-Autoencoder and 2-Layer Detection

3.1. Overviews

As shown in Figure 3, the overall process of our proposal is as follows: First, multiple autoencoders are trained using attack data and combined with the original autoencoders trained with normal data for joint decision-making. Based on the decision results, the data to be detected are partitioned into four regions: quasi-normal region, quasi-attack region, divergence region, and undetermined region. The data in the divergence region can be further divided into multiple sub-regions, which will be explained in detail later. For a specific region, the best-performing autoencoder from all previously trained autoencoders is selected as the classifier for that region, determining whether the data are normal or an attack.

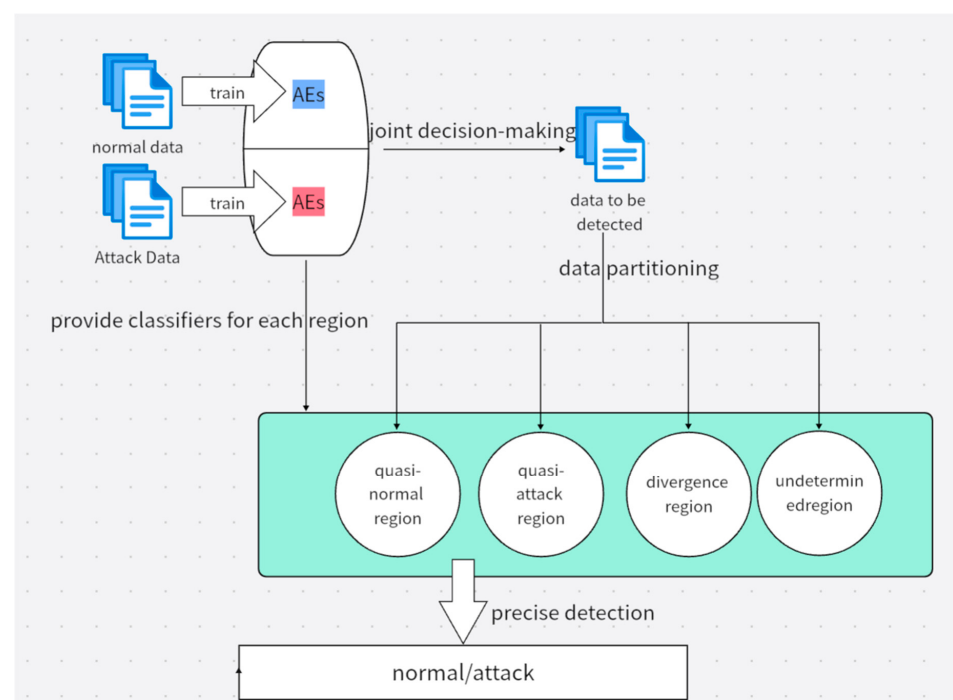


Figure 3. Proposed architecture diagram.

Next, we will provide a detailed introduction to each stage.

3.2. Training Multiple Autoencoders Using Attack Data and Normal Data, Respectively

Typically, autoencoders used for attack detection are trained only on normal (negative) data. However, the classification results of autoencoders are based on thresholds, and proper thresholds are very difficult to predefine. Thus, there will always be some anomalous data below the threshold to avoid high false-positive rates. This leads to situations where intrusion data are present but no alert is triggered, which is fatal for an intrusion detection system (IDS). To minimize this issue, in this study, we additionally trained multiple autoencoders, each with a training set consisting of various types of attack data paired with the best features for that type of attack. In other words, these autoencoders are trained on positive data. Unlike traditional autoencoders, the new autoencoder indicates an attack has been detected when the reconstruction error is less than the threshold.

Next, we trained multiple autoencoders using both normal and attack data distributions. For example, for a dataset containing normal data and three types of attacks, a total of six autoencoders need to be trained, including three models trained on normal data and three models trained on attack data. As shown in Figure 4, we used the naming

convention of “the type of attack that the autoencoder is proficient at detecting” followed by “_” and “the label of the dataset used for training” to name a specific autoencoder. For example, DoS_neg refers to an autoencoder trained on normal data whose features are more likely to distinguish DoS attacks, while DoS_pos refers to an autoencoder trained on DoS attack data, where the feature selection retains those that are more likely to distinguish DoS attacks. This naming convention shows that the models exist in pairs, so we refer to such pairs of models as “paired autoencoders” (e.g., Probe_neg and Probe_pos).

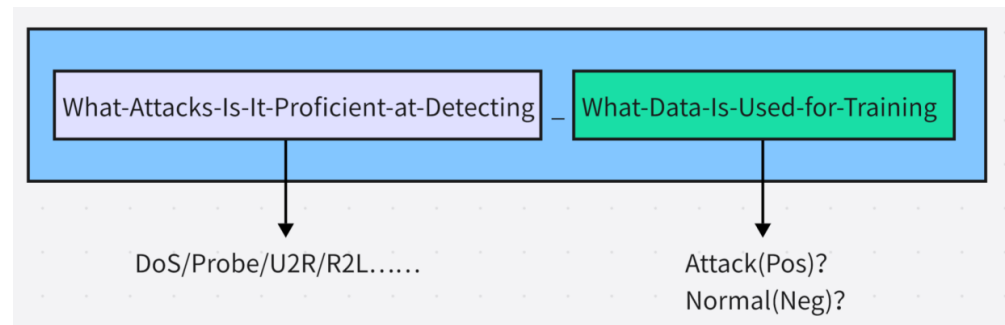


Figure 4. Naming conventions of the autoencoders.

3.3. First Layer—Data Partitioning

This section corresponds to the top right part of Figure 3, where the data to be detected are divided into four regions. As shown in Figure 5, based on the joint decision-making of these autoencoders, the data space can be divided into four regions: quasi-normal region, quasi-attack region, divergence region, and undetermined region. We use a Venn diagram to describe the behavior of a pair of autoencoders; consider the left ellipse as representing the autoencoder trained on attack data and the right ellipse as representing the autoencoder trained on normal data. If data points are distributed inside the circle, it means their reconstruction error on the corresponding autoencoder is less than the threshold of that autoencoder; conversely, if they are distributed outside the circle, their reconstruction error is greater than the threshold. In this way, the left difference set in the Venn diagram in Figure 5 represents the data that both autoencoders classify as attacks, while the right difference set represents the data that both autoencoders classify as normal. The intersection represents the data that the _neg autoencoder classifies as normal and the _pos autoencoder classifies as an attack, while the external complement set represents the data that the _neg autoencoder classifies as an attack and the _pos autoencoder classifies as normal. By observing the distribution of data in the Venn diagram, we can divide the data to be detected into different regions according to the below method (data can be inputted one by one or collected in sufficient quantity and then inputted all at once (offline)).

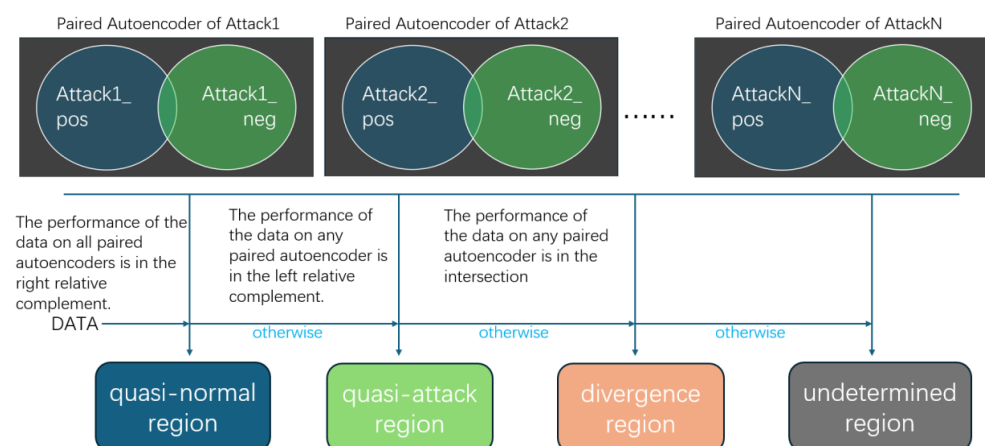


Figure 5. Four regions of data partitioning.

- If all paired autoencoders classify the data as normal, then the data are classified into the quasi-normal region.
- If any paired autoencoder determines the data to be abnormal (i.e., both the autoencoder trained with normal data and the autoencoder trained with attack data classify the data as abnormal), then the data are classified into the quasi-attack region.
- Otherwise, if the performance of the data on any paired autoencoder is in the intersection, then the data are classified into the divergence region.
- In all other cases, the data are classified into the undetermined region. For example, when there is a pair of autoencoders where the one trained on normal data classifies the data as an attack and the one trained on attack data classifies the data as non-attack.

In regions other than the divergence region, the distribution of data in the feature space is relatively simple, but the distribution of data in the divergence region remains chaotic. There is often a significant overlap in the distribution ranges of different types of data in the feature space. Therefore, in the divergence region, the data can be further divided into multiple areas according to the following rules:

- Data falls within the intersection only in one set of paired autoencoders for one kind of attack;
- Data falls within the intersection in multiple sets of paired autoencoders.

This classification method serves as an initial processing step for the detected data, so we refer to it as “data partitioning in the first layer”. This approach has several advantages: first, it essentially performs a rough classification of the data, with some partitions already having high data purity. For example, the data in the quasi-normal region are classified as normal by all autoencoders, making it highly likely that the data in this region are indeed normal; second, partitioning the data reduces the complexity of the data in each region, making it more conducive to detection in the second layer.

3.4. Second Layer—Precise Detection

After partitioning the data, the next step is to make precise predictions for the data in each region. Previously, we obtained multiple autoencoders. Now, for a specific region, we can select the autoencoder that has the best classification performance for that region’s data as the classifier for that region (classification performance is evaluated based on the accuracy on the validation set). This “optimal classifier” may be trained on negative data or positive data. For example, in the case of three types of attacks—Attack1, Attack2, and Attack3—the optimal classifier for a certain region is the one selected from Attack1_neg, Attack2_neg, Attack3_neg, Attack1_pos, Attack2_pos, and Attack3_pos. The final prediction for each region, determined by the “optimal classifier”, is what is referred to as “precise detection in the second layer”. It is important to note that due to the different ranges of data samples, even if the same autoencoder is used, the thresholds for partitioning the data in the first layer and the second layer should be different.

4. Experiments

The flow chart in Figure 6 shows the main steps of the experiment. Firstly, dataset preprocessing is carried out. Then, the model is trained, and the parameters are adjusted. After that, the model is tested and evaluated. Finally, statistical analysis is conducted on the experimental results. These steps ensure the effectiveness and reliability of the experiment. Detailed explanations of each step will be introduced in the following subsections.

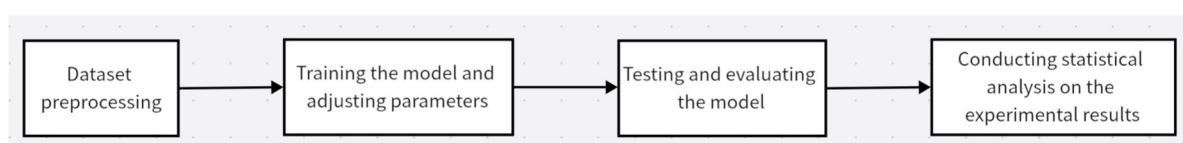


Figure 6. Experimental flow chart.

4.1. Dataset

The dataset used in this study was the NSL-KDD dataset [16]. The NSL-KDD dataset is an improved version of the KDD Cup 99 dataset [28], addressing issues such as redundant records and imbalanced data in the original dataset. This dataset is widely used in the field of network intrusion detection and aims to provide a more representative and usable dataset.

The NSL-KDD dataset is widely used not only in machine learning model evaluation and feature selection but also in testing and validating new network intrusion detection methods. Its improvements include the removal of redundant records, making the training and testing sets more balanced and representative [29,30].

Additionally, the NSL-KDD dataset includes multiple training and testing subsets of varying difficulty, such as KDDTrain+ and KDDTest+, which provide samples of different quantities and complexities to help researchers more comprehensively evaluate the performance of intrusion detection systems.

The records in the dataset are divided into normal and attack categories, with the attack types further divided into four major categories: Probe, DoS, U2R, and R2L. In the training set, there are 67,343 normal data, 45,927 DoS attack data, 11,656 Probe attack data, 995 R2L attack data, and 52 U2R attack data in total, while in the testing set, the numbers of normal data, DoS attack data, Probe attack data, R2L attack data, and U2R attack data are 9711, 7460, 2421, 2885, and 67, respectively. Each data sample contains 41 features, which are mainly divided into the following categories:

- **Basic features:** Describes the basic attributes of a single TCP connection, such as duration, protocol type, service type, etc. These features are derived from basic information at the IP and TCP layers.
- **Content features:** Describes features related to the contents of the data packets, such as the number of failed login attempts, number of access control files, etc. These features are mainly used to detect U2R and R2L attacks, as these attacks often involve spoofing or abnormal login behaviors.
- **Time-based traffic features:** Traffic features are calculated based on a time window, such as the time interval between connections, the number of connections to the same service within a time window, etc. These features help detect DoS and Probe attacks, as these attacks often manifest as a large number of connection requests in a short period.
- **Host-based traffic features:** Statistical features are based on the host, such as the number of connections to the same host, the number of connections to the same host within a specific time window, etc. These features are used to identify attack behaviors targeting a single host.

4.2. Data Preprocessing

Firstly, it is important to clarify that in order to determine the threshold for the autoencoder and select the best-performing classifier in each region, a validation set for parameter tuning is needed. For this purpose, randomly select 20% of the data from KDDTrain+.txt as the validation set, leaving the remaining 80% of the data to train the model (using stratified sampling, 20% of the data from each of the five categories—normal, DoS, Probe, R2L, and U2R—are randomly selected to form the validation set, while the remaining 80% of the data from all categories are combined to form the training set). The data in KDDTest+.txt will be used as the test set for the final model evaluation.

Next, to partition the training data, divide the training set into two parts based on the label: if the data label is “normal”, it is classified as normal data; if the label is not “normal”, it is classified as attack data. The attack data are further divided into four types of attacks according to their labels. Before the data are fed into the training process, feature selection is performed according to Table 1. Additionally, string feature values are encoded using one-hot encoding. It should be noted that the validation set and test set data should undergo feature selection and encoding before being input into the model. However, to ensure the

consistency of the one-hot encoding results, perform this encoding simultaneously with the training set.

4.3. Model Training and Selecting

First, train eight autoencoders (four pairs) using the training set data and adjust the thresholds of each autoencoder based on their performance on the validation set. Compare the reconstruction error of the validation set data on each autoencoder with the thresholds, and divide the data into different regions accordingly.

For each region, use different autoencoders for classification. The autoencoder with the highest accuracy is selected as the classifier for that region. The classifiers selected for each region are shown in Table 3. It is worth noting that Region1 consists of datasets that appear only in the divergence region of the paired autoencoder for DoS. Region2 contains datasets that appear exclusively in the divergence region for Probe. The data in Region3 are found solely in the divergence region for U2R, while the data in Region4 exists only in the divergence region for R2L. Finally, Region5 is composed of datasets that appear in the divergence regions of multiple paired autoencoders. As an example, the division of the validation set data for the quasi-normal region of DoS_pos is illustrated in Figure 7. We used MSE (mean squared error) as the reconstruction error, and by continuously adjusting the threshold, we found that when the threshold is at the position of the blue dashed line, it can better separate the normal data from the attack data. This achieves a better detection effect than simply judging the data in this region as normal.

Table 3. Selected classifiers for each region.

Region		Selected Autoencoder
Quasi-normal region		DoS_pos
Quasi-attack region		DoS_pos
Undetermined region		U2R_neg
Divergence region	Region1	R2L_neg
	Region2	U2R_neg
	Region3	DoS_pos
	Region4	R2L_neg
	Region5	R2L_neg

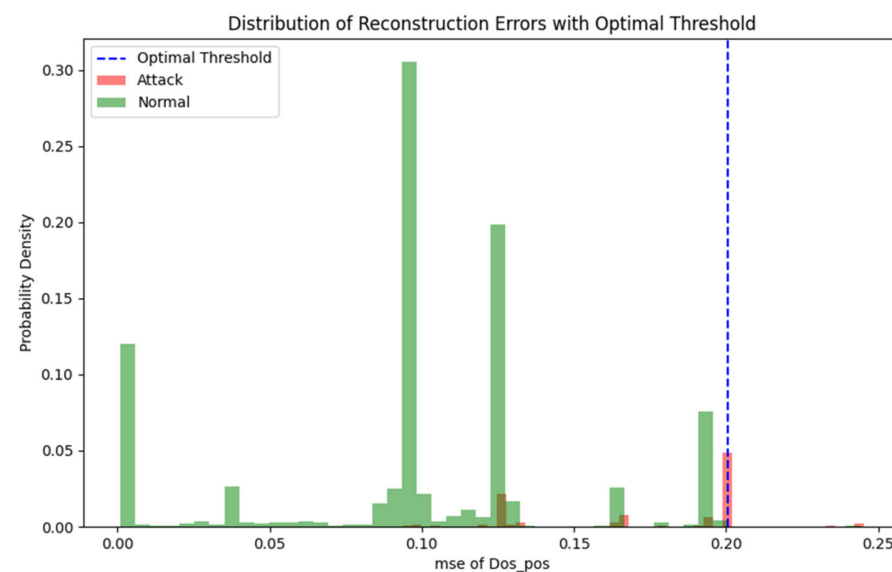


Figure 7. Precise detection in the quasi-attack region.

4.4. Evaluation Metrics

This study uses accuracy to measure the overall correctness of the model's classification and uses the F1-Score to comprehensively evaluate the model's performance on imbalanced datasets. We calculated these two metrics on all test samples and also on each of the divided data partitions. In addition, we also recorded the size of the model file and the training time to show the lightweight nature of the model. See the specific explanations below for details.

- (1) Overall Accuracy and Overall F1-Score: Accuracy is used to measure the overall correctness of a model's classifications, suitable for datasets with balanced class distributions. F1-Score is useful for evaluating model performance on imbalanced datasets, particularly when balancing precision and recall is important. They can be calculated using the following formulas:

$$\text{Overall Accuracy} = (TP^{all} + TN^{all} + FP^{all} + FN^{all}) / (TP^{all} + TN^{all}), \quad (4)$$

$$\text{Overall Precision} = (TP^{all}) / (TP^{all} + FP^{all}), \quad (5)$$

$$\text{Overall Recall} = (TP^{all}) / (TP^{all} + FN^{all}), \quad (6)$$

$$\text{Overall F1-Score} = (2 * \text{Overall Precision} * \text{Overall Recall}) / (\text{Overall Precision} + \text{Overall Recall}), \quad (7)$$

where TP^{all} , TN^{all} , FP^{all} , and FN^{all} are the counts of True Positives, True Negatives, False Positives, and False Negatives, respectively, calculated over the entire test set.

- (2) Within-Region Accuracy: Assume that after the first layer of data partitioning, no precise detection is performed, and data in the quasi-normal region, divergence region, and indeterminate region are all predicted as normal (because these regions contain more normal labels than attack labels), while data in the quasi-attack region are all predicted as attacks. Record the accuracy within each region under this assumption (the ratio of correctly predicted data to the total data in the region) and compare it with the accuracy within each region after introducing precise detection to demonstrate the improvement brought by precise detection. The calculation formula is shown as follows:

$$\text{Within-Region Accuracy} = (TP^{reg} + TN^{reg} + FP^{reg} + FN^{reg}) / (TP^{reg} + TN^{reg}), \quad (8)$$

where TP^{reg} , TN^{reg} , FP^{reg} , and FN^{reg} are the counts of True Positives, True Negatives, False Positives, and False Negatives, respectively, calculated within the test set of the region.

- (3) Training Duration and Model Size: Compare the training duration and model size with several commonly used machine learning models to demonstrate the lightweight nature of the proposed model.

4.5. Experiment Results: Overall Accuracy and Overall F1-Score

Through experiments, we obtained the confusion matrices of the original model, the original model with "precise detection" added, and the method proposed in this paper, as shown in Table 4. From these three confusion matrices, we can derive the overall accuracy and overall F1-Score for each model. Table 4 (a) shows the confusion matrix for the original method [15]; (b) presents the confusion matrix obtained by using autoencoders trained only on normal data for data partitioning and selecting the best classifier (precise detection model) from these autoencoders for precise detection (see Section 3.4); and (c) displays the confusion matrix obtained from the proposed method in this study.

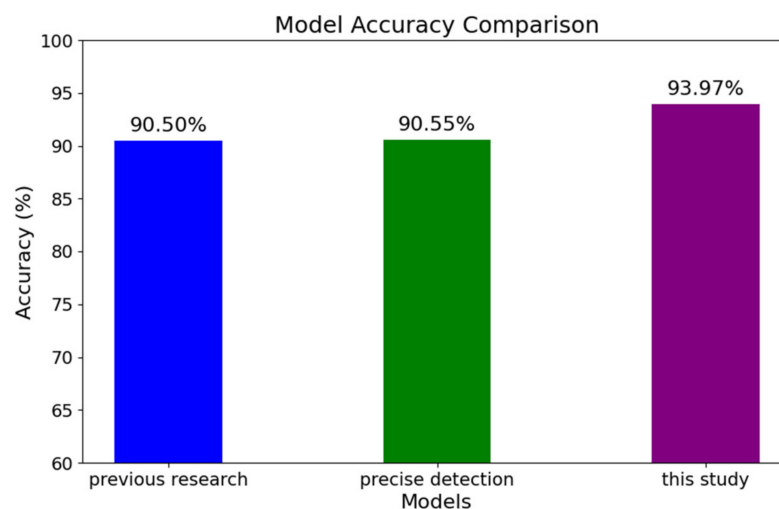
The overall accuracy and overall F1-Score of our proposed method are shown in Figure 8. Figure 8a compares the accuracy of different models, while Figure 8b compares the F1-Scores of these models. In Figure 8a, the left vertical bar (blue) represents the accuracy of the original method [15], the middle vertical bar (green) represents the accuracy after simply adding 'precise detection' to the second layer of the original method (i.e., based on

predictions from four autoencoders trained on normal data, the data are divided into quasi-normal and quasi-attack regions, and the best classifier is selected for each region to predict the final result), and the right vertical bar (purple) shows the accuracy of our proposed method, which first trains paired autoencoders and then performs precise detection within the four regions. Similarly, in Figure 8b, the blue bars represent the F1-Score of the original method [15], the green bars represent the F1-Score of the ‘precise detection’ method, and the purple bars show the F1-Score of our proposed method. From both figures, we can observe the following points:

- (1) The original method has already achieved 90.49% accuracy and 91.89% F1-Score.
- (2) Simply adding precise detection only brings a slight improvement.
- (3) By combining data partitioning from the first layer with precise detection from the second layer, the model’s performance is significantly improved.
- (4) It can also be calculated from the confusion matrix that, compared with the original case, the false negative rate (FNR) and false positive rate (FPR) of the model have decreased from 7.77% and 10.65% to 5.17% and 6.64%, respectively.

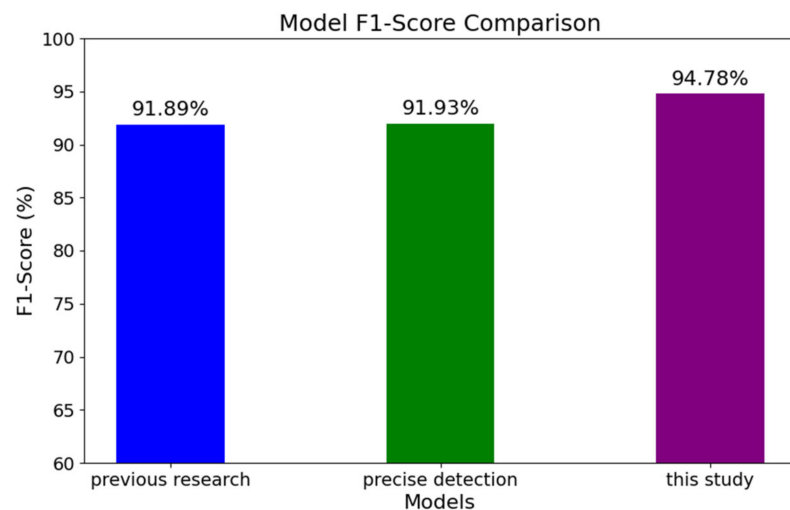
Table 4. Confusion matrix of models.

(a) Confusion Matrix of Previous Research [15]		
	Attack (Predicted)	Normal (Predicted)
Attack (Actual)	12,136	696
Normal (Actual)	1446	8265
(b) Confusion Matrix of Precise Detection Models		
	Attack (Predicted)	Normal (Predicted)
Attack (Actual)	12,143	689
Normal (Actual)	1442	8269
(c) Confusion Matrix of This Study (Both Layers)		
	Attack (Predicted)	Normal (Predicted)
Attack (Actual)	12,350	482
Normal (Actual)	878	8833



(a) Model Accuracy Comparison

Figure 8. Cont.



(b) Model F1-Score Comparison

Figure 8. Model accuracy and F1-Score comparison.

Discussion: This experiment result can be understood as follows: if precise detection is performed solely based on the original method, the distribution of data in the quasi-normal and quasi-attack regions in the feature space remains complex, making it difficult for a single autoencoder to perform the classification. The advantages of combining data partitioning with precise detection are as follows:

- (1) It provides more selectable autoencoders to serve as classifiers for precise detection;
- (2) The distribution of data in each region in the feature space becomes simpler, making classification easier.

4.6. Experiment Result: Within-Region Accuracy

The experimental results of the accuracy within each region are shown in Table 5, where each row represents the accuracy with and without precise detection in a specific region, as well as the difference between them.

Table 5. Comparison of accuracy within regions.

Region	Without Precise Detection	With Precise Detection	Increase Magnitude
Quasi-Normal Region	93.18%	96.31%	3.13%
Quasi-Attack Region	89.84%	94.66%	4.82%
Divergence Region	68.37%	89.05%	20.68%
Undertermined Region	96.48%	96.48%	0

From Table 5, it can be seen that the accuracy in each region has increased to varying degrees after adopting precise detection, with the accuracy in the divergence region increasing by approximately 20%. In the worst-case scenario, even if the classifier in the second layer does not function effectively, it will not reduce the accuracy in that region. It can be said that the introduction of precise detection does not negatively impact the model's accuracy. As for the additional time brought by precise detection, since the classifier uses existing models and does not require retraining, it is negligible.

4.7. Training Time and Model Size Analysis

As shown in Table 6, the model proposed in this study is compared with the SVM model, the random forest model, and the model from previous research.

Table 6. Comparison of training time and model size.

Model	Without Precise Detection	Model Size
RF	12.6 s	2.29 MB
SVM(kernel = linear)	20 min+	
This Study	42.3 s	484 KB
Previous Research	29.5 s	242 KB

It can be seen that the proposed model has a significant advantage in terms of model size compared to the SVM and random forest. In terms of training time, its performance is also superior to that of the SVM and is on the same order of magnitude as that of the random forest. And, compared to previous research, although the model size is twice that of the previous model, it is still relatively small, and such an increase does not cause a significant burden. Moreover, since training is not performed continuously, a slight increase in training time is acceptable. Considering the significant improvement in model accuracy, this level of sacrifice does not pose a serious problem. Overall, this lightweight characteristic makes it more suitable for resource-constrained environments like IoT compared to other machine learning models.

5. Conclusions and Future Work

In this study, we introduced an efficient classifier model for intrusion detection, leveraging autoencoders specifically designed for deployment on edge devices within IoT networks. Recognizing the limitations of traditional intrusion detection models on edge devices, such as insufficient computational resources and the need for enhanced accuracy, we proposed a lightweight model utilizing the Extreme Learning Machine (ELM) for its swift training and compact size.

Our approach involved training multiple autoencoders using different types of attack data as well as normal data paired with varying feature sets. These autoencoders collectively performed initial data prediction, partitioning data based on prediction results, and subsequently leveraging their characteristics for precise detection within each classification region. This method not only maintained the lightweight nature of the model but also improved its accuracy and F1-Score by 3.5% and 2.9%, respectively, compared to the original approach on the NSL-KDD dataset while maintaining a lightweight advantage over traditional models like random forest and support vector machine.

We evaluated the proposed model using the NSL-KDD dataset, which contains both normal and various attack types (DoS, Probe, U2R, R2L). Our experimental setup involved training eight autoencoders, including both normal and attack data, leading to the creation of “paired autoencoders”. These paired autoencoders facilitated the initial data partitioning into quasi-normal, quasi-attack, divergence, and undetermined regions. The subsequent precise detection within these regions further enhanced the model’s performance.

The experimental results demonstrated that our model achieved superior overall accuracy, particularly in the divergence region where accuracy improved by approximately 20%. Furthermore, the proposed model exhibited significant advantages in terms of model size and training duration, making it highly suitable for edge device deployment.

In summary, the binary classifier model based on autoencoders and ELM presented in this study effectively addresses the challenges faced by traditional IDS models in IoT environments. By combining data partitioning and precise detection, multiple charts derived from the final experimental results show that our method not only ensures high detection accuracy but also maintains a lightweight characteristic, which is crucial for practical implementation on edge devices.

In the future, we will use more datasets to validate our model. Additionally, we will simulate a distributed IoT environment and apply methods like federated learning to aggregate the model. Regarding the model, we will balance accuracy and model size,

appropriately reducing the number of autoencoders in data spaces where the impact on detection accuracy is minimal, to ensure further lightweight optimization.

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

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are openly available in kaggle at <https://www.kaggle.com/datasets/hassan06/nslkdd>, accessed on 8 October 2024.

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

References

1. Abdul-Qawy, A.S.; Pramod, P.J.; Magesh, E.; Srinivasulu, T. The internet of things (iot): An overview. *Int. J. Eng. Res. Appl.* **2015**, *5*, 71–82.
2. Gartner Says Worldwide IoT Security Spending Will Reach \$1.5 Billion in 2018. Available online: <https://www.gartner.com/en/newsroom/press-releases/2018-03-21-gartner-says-worldwide-iot-security-spending-will-reach-1-point-5-billion-in-2018> (accessed on 8 October 2024).
3. Antonakakis, M.; April, T.; Bailey, M.; Bernhard, M.; Bursztein, E.; Cochran, J.; Durumeric, Z.; Halderman, J.A.; Invernizzi, L.; Kallitsis, M.; et al. Understanding the mirai botnet. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 1093–1110.
4. Khan, R.; Maynard, P.; McLaughlin, K.; Laverty, D.; Sezer, S. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016, Belfast, UK, 23–25 August 2016; BSC: London, UK, 2016; pp. 53–63.
5. Farwell, J.P.; Rohozinski, R. Stuxnet and the future of cyber war. *Survival* **2011**, *53*, 23–40. [CrossRef]
6. Stanislav, M.; Beardsley, T. Hacking iot: A case study on baby monitor exposures and vulnerabilities. *Rapid7 Report* **2015**. Available online: <https://information.rapid7.com/iot-baby-monitor-research.html> (accessed on 8 October 2024).
7. Eskandari, M.; Janjua, Z.H.; Vecchio, M.; Antonelli, F. Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices. *IEEE Internet Things J.* **2020**, *7*, 6882–6897. [CrossRef]
8. Nguyen, T.D.; Marchal, S.; Miettinen, M.; Fereidooni, H.; Asokan, N.; Sadeghi, A.R. DfIoT: A federated self-learning anomaly detection system for IoT. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–9 July 2019; pp. 756–767.
9. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
10. Le Cun, Y.; Fogelman-Soulié, F. Modèles connexionnistes de l'apprentissage. *Intellectica* **1987**, *2*, 114–143. [CrossRef]
11. Choi, H.; Kim, M.; Lee, G.; Kim, W. Unsupervised learning approach for network intrusion detection system using autoencoders. *J. Supercomput.* **2019**, *75*, 5597–5621. [CrossRef]
12. Ieracitano, C.; Adeel, A.; Morabito, F.C.; Hussain, A. A novel statistical analysis and autoencoder driven intelligent intrusion detection approach. *Neurocomputing* **2020**, *387*, 51–62. [CrossRef]
13. Tsukada, M.; Kondo, M.; Matsutani, H. A neural network-based on-device learning anomaly detector for edge devices. *IEEE Trans. Comput.* **2020**, *69*, 1027–1044. [CrossRef]
14. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: A new learning scheme of feedforward neural networks. In Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), Budapest, Hungary, 25–29 July 2004; Volume 2, pp. 985–990.
15. Qin, Y.; Kondo, M. Federated Learning-Based Network Intrusion Detection with a Feature Selection Approach; IPSJ SIG Technical Report (in Japanese). In Proceedings of the 2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Kuala Lumpur, Malaysia, 12–13 June 2021; pp. 1–7.
16. NSL_KDD Dataset. Available online: <https://www.kaggle.com/datasets/hassan06/nslkdd> (accessed on 8 October 2024).
17. Ioulanou, P.; Vasilakis, V.; Moscholios, I.; Logothetis, M. A Signature-based Intrusion Detection System for the Internet of Things. Information and Communication Technology Form. 11–13 July 2018. Available online: <https://eprints.whiterose.ac.uk/133312/> (accessed on 8 October 2024).
18. Li, W.; Tug, S.; Meng, W.; Wang, Y. Designing collaborative blockchained signature-based intrusion detection in IoT environments. *Future Gener. Comput. Syst.* **2019**, *96*, 481–489. [CrossRef]
19. Sheikh, N.U.; Rahman, H.; Vikram, S.; AlQahtani, H. A lightweight signature-based IDS for IoT environment. *arXiv* **2018**, arXiv:1811.04582.

20. Lo, W.W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. E-graphsage: A graph neural network based intrusion detection system for iot. In Proceedings of the NOMS 2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022; pp. 1–9.
21. Zhou, X.; Liang, W.; Li, W.; Yan, K.; Shimizu, S.; Kevin, I.; Wang, K. Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system. *IEEE Internet Things J.* **2021**, *9*, 9310–9319. [[CrossRef](#)]
22. Almiani, M.; AbuGhazleh, A.; Al-Rahayfeh, A.; Atiewi, S.; Razaque, A. Deep recurrent neural network for IoT intrusion detection system. *Simul. Model. Pract. Theory* **2020**, *101*, 102031. [[CrossRef](#)]
23. Wang, M.; Yang, N.; Weng, N. Securing a smart home with a transformer-based iot intrusion detection system. *Electronics* **2023**, *12*, 2100. [[CrossRef](#)]
24. Fraihat, S.; Makhadmeh, S.; Awad, M.; Al-Betar, M.A.; Al-Redhaei, A. Intrusion detection system for large-scale IoT NetFlow networks using machine learning with modified Arithmetic Optimization Algorithm. *Internet Things* **2023**, *22*, 100819. [[CrossRef](#)]
25. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [[CrossRef](#)]
26. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [[CrossRef](#)]
27. Amin, J.; Sharif, M.; Gul, N.; Raza, M.; Anjum, M.A.; Nisar, M.W.; Bukhari, S.A.C. Brain tumor detection by using stacked autoencoders in deep learning. *J. Med. Syst.* **2020**, *44*, 32. [[CrossRef](#)] [[PubMed](#)]
28. Vařeka, L.; Mautner, P. Stacked autoencoders for the P300 component detection. *Front. Neurosci.* **2017**, *11*, 302. [[CrossRef](#)] [[PubMed](#)]
29. KDD Cup Dataset. Available online: <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (accessed on 8 October 2024).
30. Bala, R.; Nagpal, R. A review on kdd cup99 and nsl nsl-kdd dataset. *Int. J. Adv. Res. Comput. Sci.* **2019**, *10*, 64. [[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.