applied sciences

MDPI

Article

Research on the Simulation Method of HTTP Traffic Based on GAN

Chenglin Yang, Dongliang Xu * and Xiao Ma

School of Computer Science and Technology, Shandong University, Weihai 264209, China; 202137596@mail.sdu.edu.cn (C.Y.); maxiaosdu@mail.sdu.edu.cn (X.M.)
* Correspondence: xudongliang@sdu.edu.cn

Abstract: Due to the increasing severity of network security issues, training corresponding detection models requires large datasets. In this work, we propose a novel method based on generative adversarial networks to synthesize network data traffic. We introduced a network traffic data normalization method based on Gaussian mixture models (GMM), and for the first time, incorporated a generator based on the Swin Transformer structure into the field of network traffic generation. To further enhance the robustness of the model, we mapped real data through an AE (autoencoder) module and optimized the training results in the form of evolutionary algorithms. We validated the training results on four different datasets and introduced four additional models for comparative experiments in the experimental evaluation section. Our proposed SEGAN outperformed other state-of-the-art network traffic emulation methods.

Keywords: GAN; HTTP stream; traffic feature mimicry; data synthesis; network data

Citation: Yang, C.; Xu, D.; Ma, X. Research on the Simulation Method of HTTP Traffic Based on GAN. *Appl. Sci.* **2024**, *14*, 2121. https://doi.org/10.3390/app14052121

Academic Editor: Ugo Vaccaro

Received: 23 January 2024
Revised: 21 February 2024
Accepted: 28 February 2024
Published: 4 March 2024

1. Introduction

With the explosive development of the Internet, significant challenges have emerged in the field of security. These challenges arise from the need to protect personal information as well as the monitoring of massive network traffic. Additionally, there is an increasing threat posed by statistical analysis-based attacks, known as traffic analysis attacks [1]. Such attacks exploit network [2] traffic to analyze the trajectory and behavior of network communication users from the perspectives of statistics, real-time data behavior analysis, and big data. On the other hand, in the field of network maintenance and detection [3–5], a large amount of network data is typically required for various model training and validation purposes. During the model validation phase, trained classification models are used to differentiate between genuine and fake data. The method proposed in this paper precisely addressed the problem of generating high-quality network traffic.

Existing traffic generators cannot guarantee the accuracy and authenticity of the synthesized traffic, often resulting in significant deviations. The quality and effectiveness of traffic are evaluated based on the correctness of the data types and the appropriateness of the labels. Even in real data, there are challenges in manually labeling a large amount of self-captured data, and the use of publicly available datasets may obscure sensitive information such as port numbers and IP addresses due to privacy concerns. This undoubtedly presents challenges for the research and development of network security.

Currently, generative adversarial networks (GANs) have achieved excellent results in the field of image processing and audio/video domains [6,7]. The advantage of GANs lies in their ability to learn from complex data distributions in a detailed manner. They can even generate high-resolution images that are indistinguishable from real ones based on pixel-level distribution patterns [8–12]. From the perspective of network traffic generation, our goal was to explore whether GANs can generate effective data-level features.

Traffic obfuscation techniques involve concealing encrypted information within legitimate data packets to bypass protocol inspection. This involves disguising a certain protocol

(source protocol) as another well-known protocol (target protocol) that is widely present in the network environment. As a result, it becomes challenging for attackers to identify hidden network traffic from a vast amount of normal network traffic. The term "obfuscation" originally appeared in the field of biology and refers to a species actively imitating certain characteristics of another species during the process of continuous biological evolution, in order to deceive predators or natural enemies.

Traffic obfuscation techniques are new methods that significantly differ from traditional tunneling techniques [5]. Tunneling techniques involve encapsulating one network layer protocol into another protocol to traverse the network and reach another router. On the other hand, traffic obfuscation techniques transform known traffic data into another known data traffic through reshaping and variation, thus evading traffic analysis attacks. Traffic obfuscation techniques can achieve communication goals without the knowledge of regulators, which is the greatest advantage of this technology. Tunneling techniques, obviously, cannot guarantee this aspect. We summarize the main contributions of this paper in the following points:

- A method based on Gaussian mixture models (GMM) was proposed for processing network data.
- A method for generating data at the flow level was proposed.
- The Swin Transformer was introduced into the field of network data generation.
- The efficiency of the generative adversarial network architecture was further improved by combining evolutionary algorithms, leading to the proposed SEGAN (Swin evolutionary generative adversarial network) model.
- The effectiveness of the generated data from the model was validated through neural network classification algorithms.

2. Related Research

The feasibility of using GANs to generate network traffic data has been recognized [13]. Although GAN-based data generation is a relatively new approach, several methods have shown promising results in different aspects. Here are a few notable examples:

Ring et al. [14] proposed a data generation method based on Wasserstein GAN with gradient penalty (WGAN-GP) using a dual time update rule. The main contribution of the paper lies in the introduction of a single hidden layer IP2Vec method for network data processing. This approach leverages techniques from natural language processing, such as Word2Vec, to transform traditional network data into fixed-length vector representations. IP addresses, destination ports, and transport protocols are treated as words and encoded as one-hot vectors. However, this method has limitations as it can only generate existing data by recombining real data, rather than generating entirely new data.

Adriel Cheng et al. [15] explored the possibility of directly generating packet files using the PAC-GAN model. Unlike Ring et al [14]., Adriel Cheng focused on simulating individual packets. Although the authors achieved a high accuracy of 0.99 in fitting a particular type of DNS packet, the success rate of fitting HTTP packets was only around 0.7. Moreover, the method is limited to simple packet structures and cannot specify the content of the generated packets. The content of the packets generated by GANs is uncontrollable, which hinders their practical application for effective data communication.

Maria Rigaki et al. [16] proposed a GAN-based method for fitting data at the data flow level. In contrast to the previous methods mentioned, the authors focused on shaping malicious traffic generated by malware into normal network traffic to evade detection. They input malicious traffic into the GAN and dynamically adjusted the output traffic by simulating Facebook chat traffic. The authors also envisioned a scenario where intercepted shaped data is fed back to the GAN for modification. This method achieved good practical results in testing. However, it has clear limitations, such as only simulating traffic from specific apps. If the app is not widely used in certain regions, the method becomes ineffective. Additionally, generating a large amount of Facebook-like data in a network environment may raise suspicions.

PcapGAN [17] is a GAN model designed to generate enhanced pcap data. The model consists of an encoder, a data generator, and a decoder. The encoder divides the network data into four parts. The generator's task is to generate data for each segmented part, and the decoder combines these scattered data into a cohesive whole, representing real network data for transmission. The entire pcap file is transformed into a directed graph G by creating an IP flow graph. The time intervals between data packets are converted into a layer sequence, and then processed in a mixed structure. However, this approach also faces challenges in generating data in more complex network scenarios. Although the generated data packets have high quality, they are mainly suitable for analyzing network flow graphs and timestamps.

ZipNet-GAN [18] is a GAN architecture that combines a new neural network called ZipNet with super-resolution targeting mobile traffic. It consists of several modules, including a spatiotemporal feature module for extracting mobile traffic features, upsampling blocks, and a core Zipper convolution block, forming a deep ZipNet architecture. The architecture specifically includes a 3D upsampling block as input, followed by a 3D deconvolution layer, three 3D convolution layers, a batch normalization layer, and an activation function layer. Without adding additional parameters, the architecture utilizes a set of additional skip connections to achieve ResNet-like architecture with enhanced performance, accelerating the overall training process and outperforming the convolution-based neural network (SRCNN [19]) in terms of prediction accuracy. However, similar to ZipNet-GAN, this architecture is designed for mobile traffic inference and analysis of corresponding patterns, rather than traffic generation.

3. Materials and Methods

In the following section, we will introduce the relevant techniques used in the model and their specific implementation. The Gaussian mixture model (GMM) is a mathematical method used to model the data, while the autoencoder (AE) is employed for data dimensionality reduction and achieving data diversity. We have also incorporated the architecture of Swin Transformer to redesign the generator architecture and utilized evolutionary algorithms for the overall construction and training of the model.

3.1. The Gaussian Mixture Model (GMM)

Due to the discrete nature of network data itself, which poses significant challenges for downstream tasks, it is necessary to standardize network data for efficient processing. Common data standardization methods include min-max normalization, z-score normalization, etc., which are not suitable for network data. For example, while the maximum value for port numbers is 65,535, another feature such as packet count has a maximum value not exceeding 100. The significant difference in magnitudes would lead to nearly zeroing the port number data if min-max normalization were used, causing substantial difficulties for downstream tasks. Additionally, since downstream tasks require mapping the distribution patterns of real data using AE (autoencoder), traditional data standardization methods are not suitable.

The main role of the Gaussian mixture model (GMM) in this paper was to model the data. The specific approach involved modeling each column of the data separately using GMM and saving the index of the selected submodel. This ensured the accuracy of data modeling.

A mixture model is a probability model that includes k component distributions to represent the overall distribution. When calculating the probability of observed data in the overall distribution, a mixture model does not require information about the component distributions from the observed data.

A single Gaussian model can be classified into two cases: one-dimensional and multi-dimensional data. When the sample data x is one-dimensional, the Gaussian distribution follows the probability density function can be defined as:

$$P(x|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \quad (1)$$

where μ represents the mean (expectation) of the data, and σ represents the standard deviation. When the sample data is multi-dimensional, the probability density distribution of the Gaussian distribution follows Equation (2):

$$P(x|\theta) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum|^{\frac{1}{2}}} \exp(-\frac{(x-\mu)^T \sum^{-1} (x-\mu)}{2}) \quad (2)$$

Similarly, where μ represents the mean (expectation) of the data, Σ represents the covariance, and D represents the data dimension. A Gaussian mixture model (GMM) [20] can be understood as a model composed of multiple individual Gaussian models. The use of multiple Gaussian mixture models is motivated by their ability to better fit the specific distribution of the data. The probability distribution function of GMM is shown in Equation (3):

$$P(x|\theta) = \sum_{k=1}^{K} \alpha_k \phi(x|\theta) \quad (3)$$

The parameter θ in the equation represents the probabilities of occurrence of multiple sub-models' expectations, variances, or covariances in the Gaussian mixture model (GMM). The specific mathematical representation is as follows. The parameter estimation method for GMM uses an iterative algorithm, with the classic algorithm being the expectation-maximization (EM) algorithm. This method involves two steps: the expectation step (E-step) and the maximization step (M-step) to compute the updated model parameters for the next iteration. The visualization of a GMM with K = 3 is shown in Figure 1.

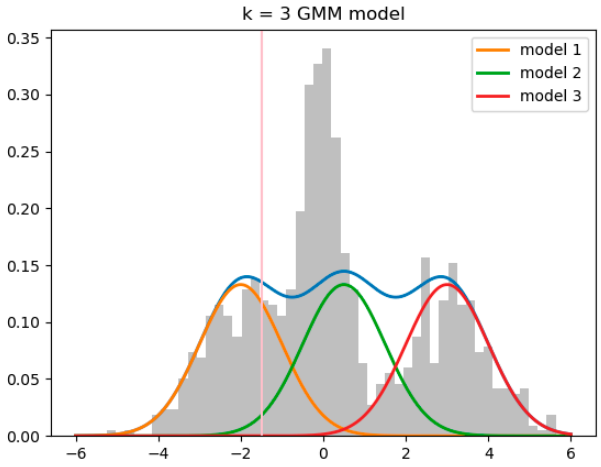

Figure 1. The outputs during training varied depending on different scenarios. A GMM model with K = 3 to describe the data distribution. The pink line represents the modeling of three sub-models when x = −1.5, and the blue line represents the estimation of the probability density function by the GMM model.

Figure 1 shows a graphical representation of a GMM model with K = 3. The red, green, and orange lines represent three sub-models, while the blue line represents the estimated probability density function of the GMM model. This blue curve can be used to describe the data distribution and the accuracy of the GMM's modeling.

Additionally, we observe a vertical pink line in the figure, indicating the modeling situation of the three sub-models at x = −1.5. Based on the illustration, we choose the model represented by the orange line to represent the data at this point, as it corresponds to the highest probability density.

Choosing the appropriate sub-models becomes crucial when using GMM for modeling. For all sub-models of continuous data, we set a predefined minimum weight ratio. If the weight of a sub-model fell below this predefined threshold, we removed that sub-model. Additionally, we reorganized all the sub-models based on their posterior probabilities, prioritizing the models with higher occurrence frequencies. We also kept track of the reordered sequence of models for inverse transformation during data generation. In Figure 1, at the position of the pink line, we chose the normalized value of the model represented by the orange line instead of the models represented by the green or red lines, as shown in Equation (4):

$$y = \frac{x - \mu_1}{4\sigma_1} \tag{4}$$

In the equation, y represents the normalized data, while x represents the x-axis coordinate of the point. μ_1 and σ_1 denote the mean and standard deviation, respectively, of the selected orange sub-model.

For categorical data, we employed one-hot encoding. The final data format combined the one-hot encoding representing the categories with the data processed through GMM for continuous variables.

3.2. AE (Autoencoder)

The autoencoder [21,22] is an unsupervised learning technique that utilizes backpropagation and optimization methods to learn a mapping relationship from input data (referred to as "data") to guide the neural network in generating a reconstructed output X that is similar or identical to the input.

An autoencoder consists of two main components: an encoder and a decoder. The encoder aims to map the original data to a hidden space h, while the decoder tries to reconstruct the original data from this latent space. Autoencoders can be understood as a form of non-linear dimensionality reduction. Variational autoencoders (VAEs) [23], which share a similar principle to autoencoders, can not only perform dimensionality transformations but also generate new data. Additionally, VAEs can be combined with generative adversarial networks (GANs) to form VAE-GANs [24].

Inspired by the principles of VAE (variational autoencoder), this paper utilized AE (autoencoder) as the pre-generator part to capture the true distribution of the data. In traditional generative adversarial network (GAN) architectures, the generator's input data is random noise, and the generator needs to map this input noise into an output that matches the real data. Throughout this process, the generator is unaware of the true distribution of the data, and parameter updates solely rely on gradients from the discriminator. The generator continuously guesses the possible distribution of correct data based on the discriminator's gradients. When the data distribution is complex, the training difficulty of the generator increases sharply. Therefore, this paper adopted the AE module to alleviate the training difficulty of the generator.

Specifically, this paper pre-trained a set of well-designed encoders and decoders to achieve dataset reconstruction. The purpose of training the AE network was to minimize reconstruction errors. In this way, the AE learned all the patterns of the true data distribution. Leveraging the representational capabilities of AE guides the generator in the GAN to learn all the distributions of the true data, preventing mode collapse where the generator can only generate a certain type of data.

3.3. Evolutionary Algorithms (EAs)

Evolutionary algorithms (EAs) [22] refer to a class of algorithms that simulate the process of natural selection, where the fittest individuals survive and reproduce while the less fit individuals are eliminated. This class includes well-known algorithms such as genetic algorithms and ant colony optimization [25–28].

Genetic algorithms, for example, explore different paths through multiple populations of individuals and iteratively converge towards the optimal solution. EAs are particularly

effective in solving highly complex, nonlinear problems, especially those with a single objective. They have achieved notable success in various computational tasks such as deep learning and machine learning.

In contrast to traditional approaches that rely on a single generator and discriminator, this paper introduced multiple generators and multiple training objectives to overcome the limitations of a single generator and objective. The discriminators simulated the natural environment and selected or retained existing generator models. During the training process, gene mutation was achieved through multiple diverse adversarial objectives. After each training iteration, the top n generators with the best fitness and quality were selected as candidates for the next generation. This method ensured that the genes of the offspring come from the optimal parents.

This paper adopted three mutation operators proposed by Chaoyue Wang [29] for asexual reproduction to generate the next generation of individuals. Each mutation operator was selected based on different training objectives, aiming to reduce the discrepancy between the generated data and the real data from different perspectives. The three operators were minimax mutation, heuristic mutation, and least-squares mutation, corresponding to different objectives. The formulas for these operators are shown in Equations (5)–(7) respectively.

$$M_G^{\text{minimax}} = \frac{1}{2} E_{z\sim p_z}[\log(1 - D(G(z)))] \tag{5}$$

$$M_G^{heuristic} = -\frac{1}{2} E_{z\sim p_z}[\log(D(G(z)))] \tag{6}$$

$$M_G^{least-square} = E_{z\sim p_z}[(D(G(z)) - 1)^2] \tag{7}$$

The minimax mutation, although prone to gradient vanishing issues during generator optimization, effectively reduces the distribution gap between the generated and real data when there is an overlap between the two distributions. On the other hand, the heuristic mutation avoids the problem of gradient vanishing but suffers from training instability, as it tends to push the two distributions apart once the JSD (Jensen-Shannon divergence) becomes negative. The least-squares mutation approaches saturation and tends to converge to a value close to zero when the discriminator output increases. However, it avoids gradient vanishing and reduces the likelihood of mode collapse in other scenarios. The three types of mutations are illustrated in Figure 2.

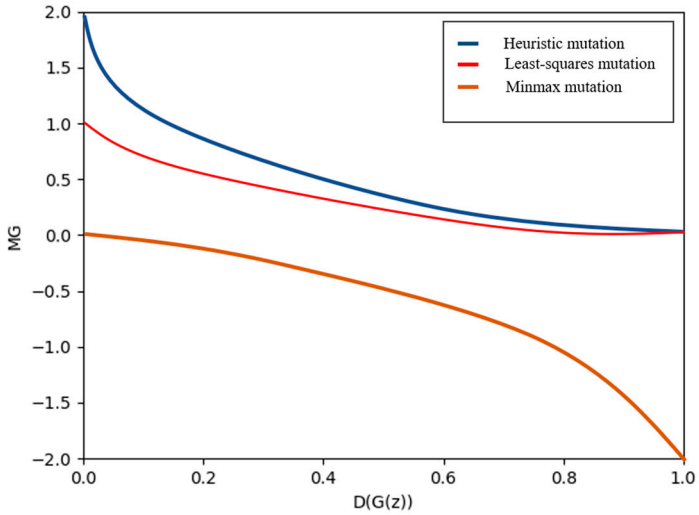

Figure 2. The three mutation operators that the generator G receives given the discriminator D.

As shown in Figure 2, by combining these three types of mutations, we ensured that there was always at least one mutation operator that effectively updated the generator

during any training stage. This ensured a smooth training process without getting stuck in optimization pitfalls, allowing the generator to continue learning and improving.

3.4. Swin Transformer

Swin Transformer [30] is a novel architecture based on the transformer framework. Unlike traditional transformers [31], Swin Transformer addresses the issue of processing images with non-standard fixed sizes, reducing computational complexity. By employing a hierarchical architecture similar to CNNs, the model becomes more flexible in handling images of different scales. Additionally, the use of window self-attention helps further reduce computational complexity.

To the best of our knowledge, the application of Swin Transformer in generative adversarial networks (GANs) for network traffic synthesis represents a novel approach in the field [32–35]. While there have been numerous examples of using transformers in both generator and discriminator architectures in the GAN domain, especially in image processing, the utilization of Swin Transformer in network traffic synthesis was unprecedented.

In order to reduce the complexity of network data during training and further enhance the quality of generated data, this paper introduced the Swin Transformer structure as the fundamental component of the generator. It replaced the traditional pattern of transposed convolution with batch normalization. Transposed convolution, while emphasizing the rationality of small components, often overlooks the relationship between local and global features. Moreover, using transposed convolution to process network data can lead to the appearance of checkerboard artifacts, posing a challenge to the generation quality of multi-feature network data.

In the generation process of network data, any feature that does not meet the requirements can be accurately captured by regulatory authorities. The Swin Transformer architecture employed in this paper utilized both global self-attention mechanisms and local windowed attention mechanisms during training. This enabled the model to capture both global and local features, avoiding situations where local data was irrational. For instance, in a communication process, it is reasonable to have both 5 and 12 packets, as well as accumulated transmission bytes of 1200 and 8300. However, a combination of 5 packets with an accumulated byte count of 8300 is unreasonable because the maximum transmission unit (MTU) is typically 1500 bytes. This highlights the importance of the relationship between global and local features in network data.

The overall workflow of the Swin Transformer was as follows. Firstly, the image was partitioned into smaller blocks using the patch partition method. Then, it went through four stages, each consisting of two parts: patch merging and Swin Transformer blocks. The structure of the Swin Transformer block, with a focus on the W-MSA and SW-MSA operations, is illustrated in Figure 3.

Figure 3 illustrates the architecture of two consecutive blocks. It is important to note that in each stage, the number of blocks was even, as each stage required two different layers: one with window attention (W-MSA) and another with shifted window attention. The reason for choosing W-MSA instead of MSA was due to the task's locality. The transformer can see all elements, but the target to be processed may only be a part of the whole. By using transformer operations within a small window, a significant amount of computational resources was saved. The computational complexity of MSA is shown in Equation (8).

$$\Omega(MSA) = 4hwC^2 + 2(hw)^2C \tag{8}$$

In the Equation (8), h, w, and C represent the height, width, and number of channels of the image, respectively.

W-MSA is responsible for capturing information within the window, while the information between windows is obtained through SW-MSA. SW-MSA consists of three components: moving window, cyclic shift, and mask. The moving window continuously slides to generate new windows, which may have different sizes. The cyclic shift operation

eliminates a portion of the windows. The mask operation is applied to handle the grouping of unrelated parts during the window sliding process, resulting in the final effect.

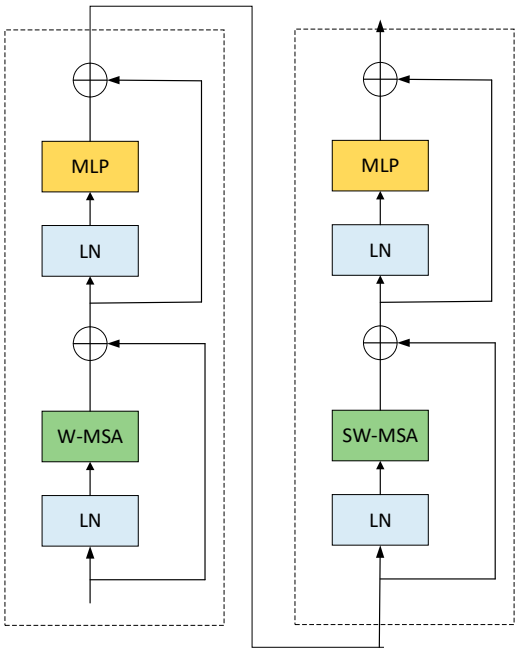

Figure 3. Structural demonstration of the W-MSA and SW-MSA in the Swin Transformer block.

4. Model Architecture

4.1. AE Module

To enable the generator to learn the patterns of the entire real data distribution, we pre-trained a simple autoencoder (AE) model. The pre-trained AE model provided the generator with the necessary noise data component. The structure of the AE model is shown in Figure 4.

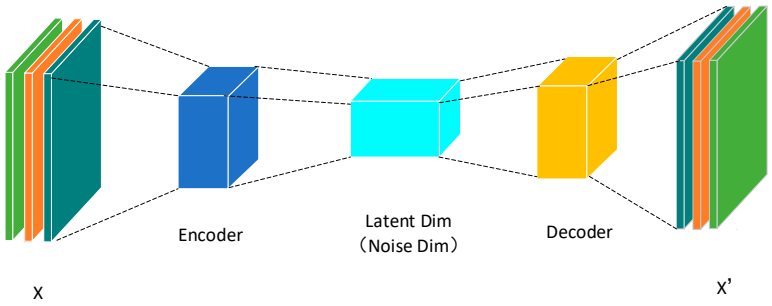

Figure 4. The auxiliary generator trains the AE model used to reconstruct the data, different colors represent different module structures.

Within the entire module, we utilized an encoder to encode the real data, and the encoded data resides in the latent dim as shown in the figure. The dimension of the latent space was the same as the noise dimension required by the generator. During the training of the AE, we used the following Formula (9) to iteratively optimize the encoder and decoder, where the decoder employed a sigmoid activation function:

$$L(x,y) = -\sum_{i=1}^{d_z} x_i \log(y_i) + (1 - x_i) \log(1 - y_i) \tag{9}$$

In the final pre-trained model, our AE achieved an accuracy of over 98%, demonstrating its effectiveness in guiding and assisting the training of the generator. By sampling

from the latent dim, the generator was able to generate a diverse range of samples, thereby avoiding the issue of mode collapse and enhancing the quality of the generated data.

4.2. GMM Data Modeling Module

We used GMM to model each column of the data separately, as mentioned in Section 3. We divided the data into two parts: continuous data and discrete data, and handled them differently.

For the continuous data part, we employed a GMM with 10 Gaussian sub-components. After 500 iterations, we removed sub-models with weights below 0.001 to ensure model simplicity. Then, we assigned each data sample to the most probable Gaussian distribution and calculated the frequency of each sub-Gaussian distribution label in the data. The process of selecting the most probable Gaussian distribution is detailed in Section 3.

Next, based on the frequency of sub-component labels, we further determined which distributions were relevant to the data. The specific approach was as follows:

- Sort the labels based on their frequency of occurrence to obtain a set of labels in descending order of occurrence count.
- Iterate through each mixture component index and check if the component is in the retained set of labels and if its weight is above a threshold. If these conditions are met, consider the component to be related to the data component.
- Represent the related components as Boolean values (True or False) to indicate their relevance. Then, use a list to record each output dimension along with its corresponding activation function and the index of the sub-component used.
- Generate one-hot encodings based on the selected patterns and reorder them in descending order of selection frequency. Store the normalized feature values and the reordered one-hot encodings together for future use and easy retrieval.
- The effectiveness of GMM is illustrated in Figures 5 and 6 as shown below:

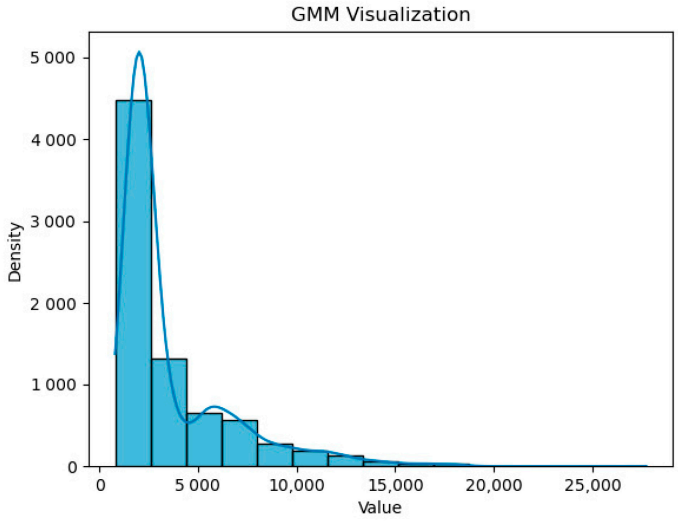

Figure 5. The GMM modeling effect of a certain feature was obtained for the experimental data, The blue curve represents the GMM density curve.

Figure 5 illustrates the overall effect of GMM modeling on a specific feature. The y-axis represents the probability density values, and the blue curve represents the GMM density curve. From the graph, we can observe that the modeling effect was satisfactory, and the fit was good.

In Figure 6, we showcase the matching of data samples with their corresponding labels. The z-axis represents the quantity of data assigned to a particular sub-distribution, while the x-axis represents the index of the sub-distribution. It should be noted that some categories in the graph do not have numerical values. This was because, after the iterations

of GMM modeling, those categories' weight proportions fell below the threshold and were removed in advance.

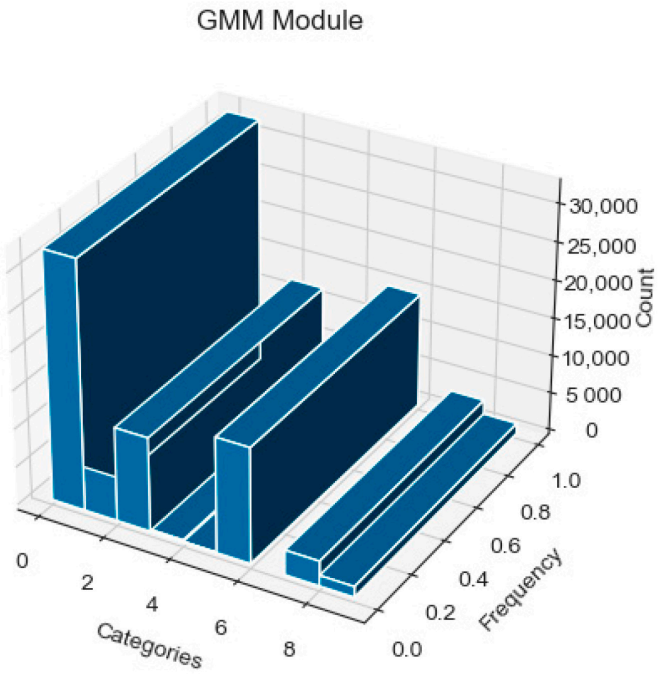

Figure 6. Mixed index label distribution of real data.

4.3. Generator Architecture

This paper innovatively combines the AE (autoencoder) with conditional vectors as inputs to the generator, in contrast to the traditional approach where generators directly obtain input data from the noise space. Similarly, in terms of the internal architecture of the generator, we creatively introduced the Swin Transformer block module as the foundational component of the generator in the field of network data generation.

Unlike traditional GAN architectures with a single generator and discriminator, at the overall level of the generative adversarial network (GAN), we treated the improved generator as an independent entity. Multiple entities were simultaneously created, and an evolutionary algorithm was employed for continuous evolution. Through this process of natural selection, the best-performing offspring were gradually retained as the final survivors.

The structure of the generator is illustrated in Figure 7a. It consists of a combination of Swin Transformer blocks (referred to as blocks) and patch expand. The initial green layer represents an MLP layer, which transforms the input data to the required input size of the blocks. The final convolutional layer reduced the number of channels in the image without changing its size.

The input of the generator consisted of a conditional vector and noise data generated by the AE module. These inputs were integrated through a linear layer and served as the input for the subsequent modules. The purpose of using a conditional vector was to allow the generator to generate specific desired data instead of generating randomly. Additionally, in the later classifier module, the classification of the generated data can further enhance the data generation performance. The type of conditional vector was determined by the number of categories in the original data. It was generated in the form of one-hot encoding, ensuring accurate classification information and facilitating vectorization and matrix operations in subsequent steps.

When using the Transformer as a generator to generate data, the phenomenon of checkerboard artifacts and ghosting often occurs [30,32,36]. This phenomenon is particularly prominent in image processing. We also observed this problem in network traffic

datasets using the same approach. After extensive experimentation with different architectures based on Swin Transformer blocks, we chose the block architecture shown in Figure 7b. We added additional activation layers after the MSA layer and MLP layer of the standard block. By setting the window size M to 2, Swin improved the local quality of the data through its unique local window structure, and the window shifting helped with attention exchange. With these modifications, we successfully eliminated ghosting issues without introducing a significant number of additional parameters.

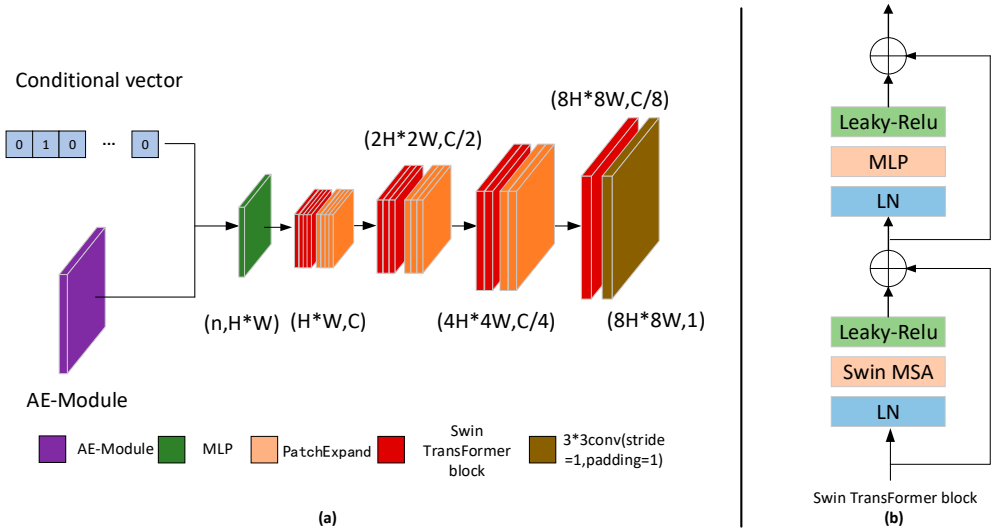

Figure 7. Figure (**a**) represents the overall structure of the generator, while figure (**b**) showcases the Swin Transformer module within it.

As shown in Figure 7a, in the main architecture part, each block except the last one was followed by a patch expand layer. This layer expanded the output features of the block to increase the resolution and expressive power of the features, and performed normalization. The input dimension of the patch expand layer was (B, H∗W, C), where B represents the batch size, H and W represent the height and width of the image, and C represents the number of channels. In the patch expand module, the input feature x was expanded to obtain a tensor of shape (B, L, 2C), where L is the size of the input feature. Then, the tensor was reshaped to (B, H, W, C) and tensor reshuffling was performed. Specifically, each pixel block was expanded to a size of 2×2, and the number of channels was reduced to one-fourth of the original. The tensor was then reshaped to $(B, -1, C/4)$, where -1 automatically calculated the dimension size. Finally, the expanded features were normalized and returned.

In each block and patch expand layer, we doubled the image's H and W dimensions and halved the number of channels. This allowed us to generate the image without using transpose convolution. In the last layer, we used a convolutional layer with a kernel size of 3, stride of 1, and padding of 1 to transform the dimensions. This convolutional layer with the specified parameters kept the overall height and width of the data unchanged while changing the number of channels. It also had the ability to extract certain data-specific features, which contributed to improving the quality of the generated data. Therefore, placing it in the last layer was beneficial for enhancing the data generation quality.

4.4. Discriminator Architecture

The discriminator adopted the classical structure of a convolutional neural network. It is used to distinguish between real and fake data and update the generator through backpropagation of gradients. In addition, in our proposed model, the discriminator also served as a natural environment to guide the selection of the generator.

As shown in Figure 8, the blue parts represent convolutional layers, the green parts represent pooling layers, and the gray parts represent activation layers. In the diagram, 'C' represents the number of channels. The numbers in parentheses, taking (4,2,1) as an example, indicate the parameters of the convolutional layer: the convolutional kernel size is 4, the stride is 2, and the padding is 1.

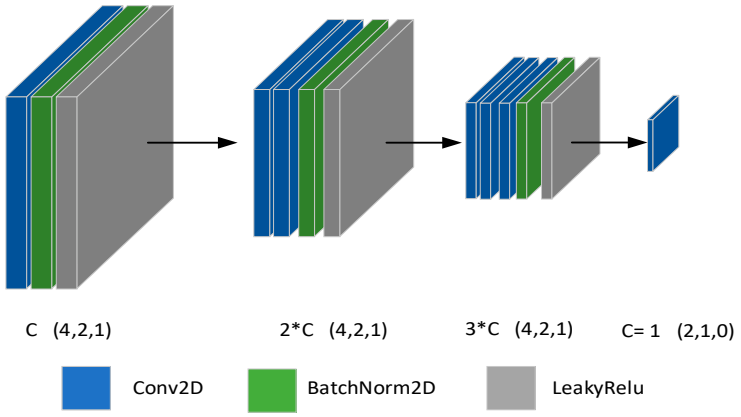

Figure 8. Discriminator architecture.

The discriminator utilizes an approach of increasing the number of channels while continuously reducing the size of the images to obtain the final output. During the training of the entire model, after all the generator models have completed training, the discriminator's discrimination results and the quality of the generated samples were used as criteria for selecting the generator models to be retained. Once the generator models for producing offspring were selected, the discriminator was trained accordingly.

It is worth mentioning that we did not choose a dual-scale learning rate for training the combination of the generator and discriminator. This was because the dual-scale approach did not achieve the expected results in experiments; it only delayed the emergence of poor results without actually improving the quality of the generated data. The formula for the discriminator is shown as Equation (10).

$$L_D = -E_{x \sim P_{data}}[\log D(x)] - E_{y \sim P_g}[\log(1 - D(y))] \tag{10}$$

In the formula, x and, y represent different real data and generated data from the generative adversarial network (GAN), respectively. This allowed the generator to continuously learn the distribution of real data and generate increasingly realistic samples.

The classifier adopted an MLP architecture and served as an auxiliary component to the discriminator, enhancing the quality of the generated data by the generator. Its structure is depicted in Figure 9, where the numbers represent the input and output dimensions of each Linear layer. In the last layer, no additional activation or dropout layers were added. The loss function of our classifier is formulated as shown in Equation (11).

$$L_{class}^{G} = E\left[|l(G(z)) - C(fe(G(z)))|\right]_{z \sim p_{(z)}} \tag{11}$$

In the equation, l(*) represents the target labels, while fe(*) represents the input features of a given instance. Specifically, we used real data as input and computed the difference between the classification values obtained by the classifier and the corresponding labels of the real data. This difference was used to train the classifier. As for the generated data, since the input included the conditional vector, we fed the generated data into the classifier. The resulting classification values were then compared with the conditional vector to compute the loss. This loss was used to backpropagate gradients and assist the generator in learning the real data distribution more effectively.

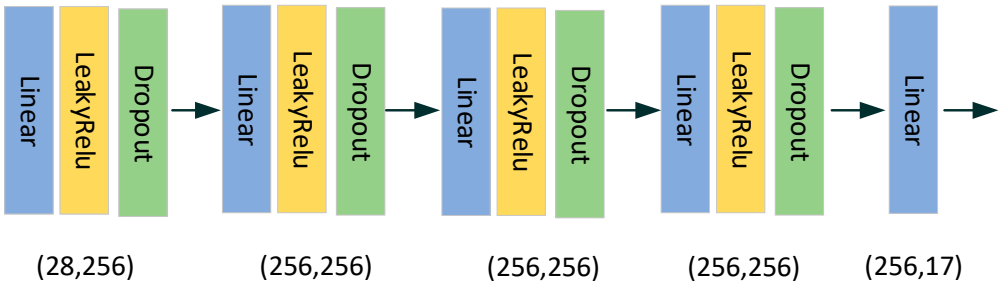

Figure 9. Classifier architecture.

4.5. Application of EVS in Modles

In the overall training process of our model, we employed the evolutionary algorithm (EA). The population size was set to three, and we retained two populations in each iteration. After multiple iterations, we selected the best offspring for preservation. During training, we had three different generators serving as parents to generate offspring in the population. The individual structure of each generator is shown in Section 4. Here, we present the framework of the entire model as depicted in Figure 10, Algorithm 1 demonstrates the specific application process of our evolutionary algorithm in the model.

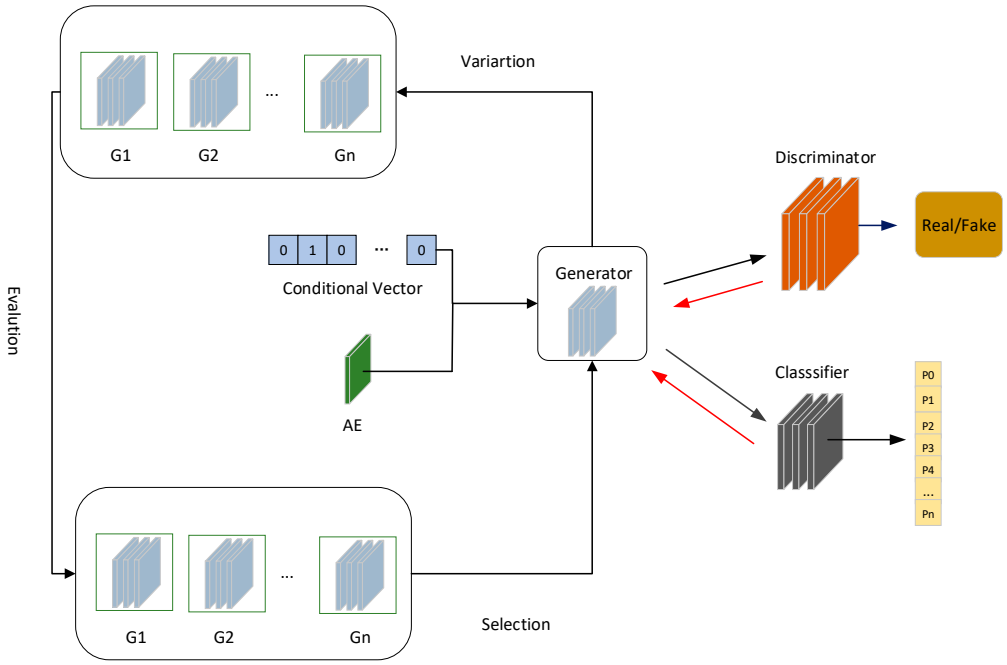

Figure 10. Model overall architecture. In the EV algorithm, we trained the entire model by using the discriminator as the environment, and the role of multiple generators was to produce high-quality offspring individuals. The black arrows in the figure represent the generation path of the generator, while the red arrows represent the process of gradient backpropagation.

The left part of Figure 10 illustrates the entire process of the evolutionary algorithm, which consisted of three stages: selection, mutation, and evaluation. In the selection stage, we chose the generator with the best fitness (evaluated via the discriminator acting as the natural environment) among different generators to be the preserved parent. It is worth noting that different fitness values were not interchangeable between parents and offspring since the natural environment was also evolving.

Algorithm 1 Evolutionary Algorithm Main Thread

Inputs:
Standardized Data x
Conditional vector t
target sample y//Target sample
Output Targeted sample x_{tar}
/* Initialize the population required for evolutionary algorithms.
Population1 ←Initialize Population(x,t)
Population2←Initialize Population(x,t)
Population3←Initialize Population(x,t)
Population(x)
iter = 0
i = 0
while iter < max_iter do
repeat
train_batch(Population)
scores1 ←Discriminator (population, y)
scores2 ← Classifier (y,t)
until TP(scores1, scores2) and TN(scores1, scores2) are in range
Select probs = Sort(scores1, scores2)
Next population = {}
while i < size +1 do
Select select probs TOP 1 as parent
Mutation1(parent)
Mutation2(parent)
Mutation3(parent)
Next population = parent∪child
population = Next population
iter = iter + 1
return x_{tar}

In the mutation stage, we applied the Equations (5)–(7) mentioned in Section 3 for mutation operations. Additionally, for the generator corresponding to the LSGAN formulation in Formula (7), the loss function for the discriminator was different from the other two. It used the mean squared error, so the discriminator loss was defined as shown in Equation (12).

$$L_D^{LSGAN} = \int_X \frac{1}{2}(p_{data}(x)(D(x)-1)^2 + p_g(x)D(x)^2)dx \tag{12}$$

Through this transformation, the form of the equation was unified with the loss values of the other two generators corresponding to their respective discriminators.

During the evaluation phase, we used two metrics to assess the generated data. One metric represented the quality of the generated data, and the other metric represented the diversity of the generated data. The quality metric was given by the discriminator and was calculated by inputting the generated data directly into the discriminator and computing the average of its output. The corresponding formula is shown in Equation (13).

$$F_D = E_z[D(G(z))] \tag{13}$$

where F_m represents the diversity metric of the generated data. It is important to note that although we have incorporated the AE module in the input part of the generator to mitigate the possibility of mode collapse, we still needed to select generators with good diversity when filtering the ones to be retained. This allowed us to obtain generators with improved generation performance as quickly as possible.

$$F_m = -\log||\nabla_D - E_x[\log D(x)] - E_z[\log(1 - D(G(z)))]|| \tag{14}$$

In the evaluation process, the negative gradient norm of the discriminator was used to measure the diversity of generated samples. If there was a higher generator value, the generated samples tended to have better diversity. In this case, the discriminator's gradient was small, which meant it did not make significant adjustments to the structure of the generator, ensuring that the generator continued to develop towards further diversity. The final metric is defined as follows in Equation (15):

$$F_{all} = F_D + \lambda F_m \tag{15}$$

The parameter α was a positive real number in this context, and it balanced the relationship between the quality and diversity of generated data.

5. Evaluation

5.1. Data Processing

The experiment used network traffic captured by Wireshark, which was deployed on a switch, capturing all network traffic requests within the entire LAN, as well as the CICIDS2017 dataset. The data was then analyzed and processed using CICFlowMeter. Since the experimental focus was on network traffic analysis, only HTTP traffic was retained. Each network flow was stored with its corresponding feature values as shown in Table 1. In total, the experiment collected 20,000 network data instances, consisting of 114,147 packets.

Table 1. Characteristics at the data flow level.

Feature	Data Type
Total number of packets	Numeric
Data stream number	Numeric
Source port	Numeric
Destination port	Numeric
Total bytes	Numeric
Flow duration	Numeric

For problematic packets in the experiment, a direct discard method was used. Similarly, for outliers in the data, a boxplot approach was employed to remove data points that deviated beyond the upper and lower quartiles. The results of the data processing are shown in Figure 11.

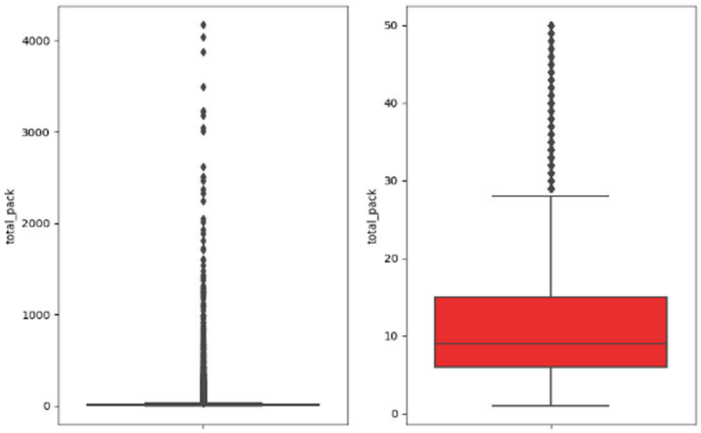

Figure 11. Comparison chart before and after packet processing. The red portion in the figure represents the main distribution range of the processed data.

On the left side is the distribution of packet quantities before processing, while on the right side is the distribution of packet lengths after processing. It can be observed that the processed data exhibited a more reasonable distribution compared to the original data.

5.2. Implementation Details

The detailed explanation of the generator and discriminator architectures used in this paper is provided in Chapter 4. In the actual implementation process, the parameters λ for the learning rates of the generator F_D and F_M were set to 0.5. This setting ensured a balance between maintaining image quality and preserving the diversity of generated images, thus achieving a compromise between the two objectives.

Regarding the number of generator populations, we chose to train three populations. After each training session, the same discriminator was used for fitness evaluation. The optimizer parameters for each generator population were consistent, with a learning rate of 1×10^{-5}. The training process for each generator population consisted of 5000 epochs.

The output dimension of the AE module, which corresponded to the noise input of the generator, was set to 50 based on multiple experiments. The batch size during training was set to 64. Using a batch size that is too large can lead to a decrease in the quality of generated data, while a batch size that is too small can result in slower training speed. Therefore, 64 was chosen as a compromise considering both factors. The specific implementation details of the generator and discriminator are described in detail in Tables 2 and 3, respectively.

Table 2. Generator architecture.

Layer Number	Neural Network Type	Input Resolution	Chanels	Depth	Headers	Window Size
1	TransformerEncoder	[1, 1]	64	4	4	4
2	PatchExpand	[1, 1]	64	-	-	-
3	TransformerEncoder	[2, 2]	32	4	4	4
4	PatchExpand	[2, 2]	32	-	-	-
5	TransformerEncoder	[4, 4]	16	4	4	4
6	PatchExpand	[4, 4]	16	-	-	-
7	TransformerEncoder	[8, 8]	8	4	4	4

Table 3. Discriminator Architecture.

Layer Num	Network Architecture
1	Conv2d(1, 64, kernel_size = (4, 4), stride = (2, 2), padding = (1, 1), bias = False)
2	BatchNorm2d(64, eps = 1×10^{-5}, momentum = 0.1)
3	LeakyReLU(negative_slope = 0.2, inplace = True)
4	Conv2d(64, 128, kernel_size = (4,4), stride = (2, 2), padding = (1, 1), bias = False)
5	BatchNorm2d(128, eps = 1×10^{-5}, momentum = 0.1)
6	LeakyReLU(negative_slope = 0.2, inplace = True)
7	Conv2d(128, 256, kernel_size = (4,4), stride = (2,2), padding = (1,1), bias = False)
8	BatchNorm2d(256, eps = 1×10^{-5}, momentum = 0.1)
9	LeakyReLU(negative_slope = 0.2, inplace = True)
10	Conv2d(256, 1, kernel_size = (2, 2), stride = (1, 1))
11	Sigmoid()

In the experiments, we explored various network architecture parameters and found that overly complex basic blocks (stacking multiple layers of Swin Transformer) did not significantly improve the quality of generated data for the generator. Instead, they increased the computational burden of the model and prolonged the computation time. After multiple trials, we determined that a generator model consisting of three layers of TransformerEncoder struck a balance between image quality and speed.

For the discriminator, a 4-layer convolutional network structure was sufficient to discriminate between real and fake data in the context of network traffic data. During the construction of the discriminator model, we chose the LeakyReLU activation function instead of the commonly used Tanh or Softmax functions. Under the guidance of these two activation functions, LeakyReLU helped to avoid the problem of gradient vanishing that may occur with the other two activation functions. This ensured that the generator

consistently learned gradient information from the discriminator, making LeakyReLU more suitable for generating network traffic data.

5.3. Baseline

We compared our model with four other models, two of which were contrast models designed with existing architectures. These included CGAN(Trans), which uses the original transformer as the basic module for the generator, and CGAN(Trans-EV), which incorporates the EV algorithm. The other two models were AE-GAN, proposed by Zhu et al. in 2022 [37], which employs an AE model for preprocessing data and constructs the generator using fully connected layers and batch normalization, and OCT-GAN, proposed by Kim et al. in 2021 [38], which utilizes a generator built on additional ODE layers and combines them with a discriminator composed of NODEs layers. To ensure fairness in the generated data, we adopted the same GMM modeling pattern for the input data of each model, ensuring that the differences in data generation were only related to the models. All models were trained for the same number of epochs, i.e., 2000 epochs, and the experiments were conducted in Table 4:

Table 4. Experiment environment.

Experimental Conditions	Model Used
Operating System	Ubuntu 20.04
CPU	I7-8700
GPU	RTX-2080
Memory	32 G (3200 Hz)
Disk	PM981
Python Version	3.8

5.4. Evaluation Results

5.4.1. Statistical Similarity

We used the following three metrics to statistically measure the similarity of the generated data:

- Difference in pairwise correlation (Diff. Corr.): this metric was used to measure the difference in correlation between two sets of data. We first calculated the correlation coefficients between the columns of the synthetic data and the real data separately. This metric had a range of $[-1, 1]$, where higher values indicated stronger correlation.
- Wasserstein distance (WD): for continuous data columns in the dataset, we used the Wasserstein distance to measure the distributional difference between the data. A smaller value indicated a smaller difference and higher similarity between the generated data and the real data. This metric was chosen as one of the evaluation indicators due to its relatively stable characteristics.
- Jensen-Shannon divergence (JSD): for categorical variables in the generated data, we used the JSD as a measure of similarity. This metric quantified the similarity between two probability distributions and had the range [0, 1]. JSD was a symmetric measure of similarity.

From Table 5, it can be observed that our proposed SE-GAN achieved the best performance in the comparison. Regarding the Corr parameter, the results of the AE-GAN, which adopts an AE structure, were second only to our SE-GAN. Our SE-GAN outperformed the AE-GAN by 16% in terms of feature similarity in generated data. This was attributed to the utilization of Swin Block modules in SE-GAN, which placed more emphasis on the local and global relationships compared to the simple, fully connected layers used in AE-GAN. As a result, SE-GAN exhibited higher feature similarity in terms of the Corr metric.

Table 5. Statistical similarity.

METHOD	MODEL				
	SE-GAN	**CGAN(Trans)**	**CGAN(Trans-EV)**	**AE-GAN**	**OCT-GAN**
Corr	0.713	0.431	0.508	0.611	0.591
WD	0.013	0.041	0.034	0.027	0.021
JSD	0.074	0.115	0.098	0.087	0.091

Regarding the WD metric, our SE-GAN also demonstrated the best performance, with our model achieving a 38% lower WD value compared to OCT-GAN. This metric adequately demonstrated the effectiveness of our proposed model in terms of similarity. Similarly, in the last parameter JSD, our model achieved an 18.7% lower value compared to OCT-GAN, which was based on the transpose convolutional structure. This was because we employed an additional classifier to train our model, enabling our model to allocate more attention to the classification data module.

5.4.2. Evaluation Metrics Based on Machine Learning

To evaluate the quality of our generated data, we employed common neural network classifiers, namely MLP, SVM, Random Forest, Decision Tree, and Multinomial Logistic Regression. We assessed the performance of our data using evaluation metrics such as accuracy (Acc), AUC, and F1-Score.

For MLP, we set the number of neurons to 256, while for Random Forest and Decision Tree, the depth was set to 56. Our objective was to measure the probability of our generated data being classified as fake data from the perspective of neural network classification. Figure 12 illustrates the accuracy values (Acc) of different models under the evaluation of the selected five neural network classifiers.

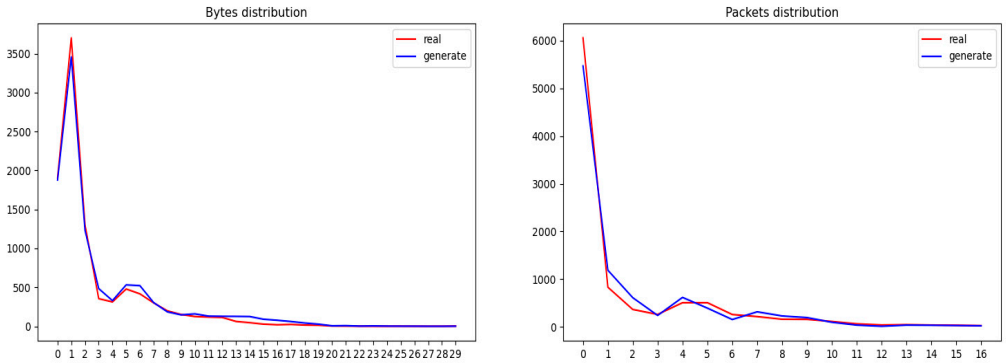

Figure 12. The number of nodes and packets is displayed.

In Table 6, the left column represents five specific methods based on neural network classification for evaluating the generated data, and the numerical values in the table represent the accuracy (Acc) of different models corresponding to their classification results. From Table 4, it can be observed that SEGAN outperformed other methods in terms of the performance measures across the five neural networks. Additionally, it was evident that the CGAN(Trans-EV) model, which incorporated the evolutionary algorithm (EV), generally exhibited lower successful identification probabilities compared to CGAN(Trans) that did not use the EV algorithm. This indicated the effectiveness of the EV algorithm proposed in this paper for network traffic data generation. Furthermore, the Acc values of AE-GAN, except for the dt column where it surpassed OCT-GAN, were consistently lower than those of OCT-GAN. This suggested that AE-GAN performed better than OCT-GAN in synthesizing network datasets.

Table 6. Different model ACC values.

METHOD	MODEL				
	SE-GAN	CGAN(Trans)	CGAN(Trans-EV)	AE-GAN	OCT-GAN
lr	10.018	12.495	11.435	10.306	10.656
dt	10.735	13.994	12.254	13.403	12.832
rf	16.031	18.741	18.142	16.655	18.342
mlp	15.781	17.253	16.634	16.139	16.551
svm	15.850	17.289	16.73	16.078	16.134

In Table 7, we present the different scores of various models on the AUC metric, where our proposed SEGAN model outperformed other models such as AE-GAN and OCT-GAN. Since the AUC value measured the reasonableness of a model's ranking of a series of samples, and because we used the same GMM network to process different data, lower data indicators indicate better model performance. Thus, it was evident that the data quality generated by the AE-GAN model was higher than that of OCT-GAN. Analyzing the underlying reasons, OCT-GAN adopted a traditional one-hot encoding plus noise input form and applied specific pattern normalization to numerical features, resulting in sparse input and excessive sensitivity to data changes. However, both AE-GAN and SEGAN employed an autoencoder to map and compress input data, thereby exhibiting stronger robustness.

Table 7. Different model AUC values.

METHOD	MODEL				
	SE-GAN	CGAN(Trans)	CGAN(Trans-EV)	AE-GAN	OCT-GAN
lr	0.036	0.051	0.045	0.055	0.061
dt	0.167	0.319	0.283	0.208	0.253
rf	0.122	0.162	0.140	0.1619	0.155
mlp	0.041	0.058	0.049	0.065	0.075
svm	0.328	0.565	0.456	0.371	0.426

As shown in Table 8, we presented the F1_score values corresponding to different models. By comparing the scores of SE-GAN and CGAN(Trans-EV), it was evident that SE-GAN achieved a significantly lower F1_score than the latter. This suggested that using the Swin Transformer block as the basic module in the generator led to better accuracy in generating different classes of network data compared to the traditional transformer structure. From the four tables presented above, it was observed that our SEGAN model consistently outperformed the state-of-the-art methods, AE-GAN and OCT-GAN. Additionally, in terms of the time required for model training to achieve similar performance, SEGAN required the shortest time, highlighting the effectiveness of our proposed approach.

Table 8. Different model F1_SCORE values.

METHOD	MODEL				
	SE-GAN	CGAN(Trans)	CGAN(Trans-EV)	AE-GAN	OCT-GAN
lr	0.195	0.232	0.207	0.210	0.242
dt	0.172	0.263	0.212	0.202	0.239
rf	0.249	0.326	0.308	0.352	0.331
mlp	0.259	0.283	0.269	0.291	0.279
svm	0.279	0.332	0.294	0.315	0.309

5.5. Experiments Comparing Different Datasets

To verify the transferability of our proposed model, after training the model using the training data, we additionally selected normal HTTP traffic data from three well-known

datasets in the field of network traffic generation for traffic emulation work. These three datasets were the CIDDS-001 [39], CSE-CIC-IDS-2018 [40], and UGR'16 datasets [41]. The results of traffic generation on different datasets are shown in Table 9.

Table 9. The traffic emulation results for different datasets.

	Training Data	CIDDS-001	CSE-CIC-IDS-2018	UGR'16
Corr	0.713	0.733	0.690	0.705
WD	0.013	0.012	0.017	0.014
JSD	0.074	0.068	0.084	0.081

From Table 9, we can observe that our proposed SEGAN model performed well across all four datasets. The correlation (Corr) values were concentrated around 0.7, indicating a high similarity between the generated and real data. Additionally, we observed that the values of the WD (Hamming distance) and JSD (Jensen-Shannon distance) metrics were both centered around 0.013 and 0.070, respectively. This suggested a small distribution gap between the generated and real data, indicating a similar data distribution status between the two.

5.6. Qualitative Results

In Figure 12, we presented the distribution of two important metrics in the traffic data: byte count and packet count. The blue line represents the generated data, while the red line represents the real data. For the total byte count feature, which is continuous, we divided both the real and generated data into 30 equally spaced bins. We then counted the number of data points falling into each bin and represented it on the y-axis. The x-axis represents the bin number.

Regarding the packet count, which is a categorical feature, we initially divided it into 17 categories, with the smallest category having a value of four. In the right half of Figure 12, the x-axis represents the categories of packet count, where category 0 corresponds to a packet count of 4, category 1 corresponds to a packet count of 5, and so on. The y-axis represents the count of packets in our data.

From Figure 12, we can observe that the distribution of total byte count in the data flow was very similar to that of the real data. Both reach their maximum in bin 1 (1700 to 2600 bytes) and then exhibited a rapid decline. There was a brief increase in bins 5 and 6 (5300 to 7100 bytes), followed by a gradual and flattening decline.

Regarding the distribution of packet count, we can also observe from the right side of Figure 12 that the highest number of packets was in bin 0 (corresponding to 4 packets), followed by bin 1 (corresponding to 5 packets). There were peaks in bins 4 and 7, followed by a relatively stable distribution. Additionally, it can be seen from the graph that both the generated and real data had the majority of packet counts concentrated within 5 packets or less.

6. Conclusions

This paper first introduces the current status of network traffic data generation based on generative adversarial networks (GANs) in recent years. In this field, existing research results are relatively scarce. To fill this gap, we propose the SEGAN model. In terms of data modeling, our model utilizes Gaussian mixture models (GMMs) to model each column of data separately, thereby improving the accuracy of data modeling. Additionally, we introduced an autoencoder (AE) mechanism to map the original data, further combating the common mode collapse phenomenon in GANs.

We utilized the latest Swin Transformer block to construct our generator architecture, which was unprecedented in the field of network traffic generation. Similarly, we designed an additional auxiliary classifier specifically for classifying generated data, which helped improve the quality of generated data. Furthermore, in the overall construction of

the network, we combined evolutionary (EV) algorithms and designed multiple population evolution and mutation algorithms to ensure the high-quality characteristics of the generated data.

Finally, we carefully designed four control models (CGAN(Trans), CGAN(Trans-EV), AE-GAN, OCT-GAN) and conducted quality tests and comparisons from both statistical and neural network classification perspectives. Through the final tests, we verified that our proposed SEGAN successfully generated network traffic data of high quality. Additionally, we validated the effectiveness of SEGAN on three different datasets, demonstrating its generalization capability to generate high-quality network traffic data across different datasets. SEGAN can contribute to network traffic emulation and generation tasks in various domains.

7. Future Work

We aim for the model proposed and implemented in this paper to be applied in the field of network traffic generation, providing higher-quality training data for neural network-based network traffic classification. However, our proposed method also has certain limitations. Currently, the network traffic generated by SEGAN is limited to HTTP traffic. Our next step is to focus on generating a more diverse range of traffic types, such as HTTPS, DNS, and others. Additionally, we plan to optimize the evolutionary algorithm used in the training process by introducing more advanced heuristic algorithms and a richer variety of population mutation methods to further enhance the diversity of generated data.

Author Contributions: C.Y.: writing—original draft preparation, D.X.: writing—review and editing, X.M.: visualization. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Shandong Provincial Natural Science Foundation (No. ZR2019PF007), basic scientific research operating expenses of Shandong University (No. 2018ZQXM004), the National Natural Science Foundation of China (No. 62076149 and No. 62376136), the Young Scholars Program of Shandong University, Weihai (No. 1050501318006), and the Science and Technology Development Plan of Weihai City (No. 1050413421912).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: CIC-IDS-2018 https://www.unb.ca/cic/datasets/ids-2018.html; UGR'16: https://nesg.ugr.es/nesg-ugr16/; CIDDS-001:https://www.hs-coburg.de/forschung/forschungsprojekte-oeffentlich/informationstechnologie/cidds-coburg-intrusion-detection-data-sets.html.

Acknowledgments: We would like to thank Panpan Li and Dong Li for their invaluable assistance in our work.

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

References

1. Chen, Q.A.; Yin, Y.; Feng, Y.; Mao, Z.M.; Liu, H.X. Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control. In Proceedings of the NDSS, San Diego, CA, USA, 18–21 February 2018.
2. Sun, Y.; Tian, Z.; Li, M.; Su, S.; Du, X.; Guizani, M. Honeypot Identification in Softwarized Industrial Cyber-Physical Systems. *IEEE Trans. Ind. Inform.* **2021**, *17*, 5542–5551. [CrossRef]
3. Zhang, H.; Gu, Z.; Tan, H.; Wang, L.; Zhu, Z.; Xie, Y.; Li, J. Masking and Purifying Inputs for Blocking Textual Adversarial Attacks. *Inf. Sci.* **2023**, *648*, 119501. [CrossRef]
4. Jia, Y.; Gu, Z.; Du, L.; Long, Y.; Wang, Y.; Li, J.; Zhang, Y. Artificial Intelligence Enabled Cyber Security Defense for Smart Cities: A Novel Attack Detection Framework based on the MDATA Model. *Knowl.-Based Syst.* **2023**, *276*, 110781. [CrossRef]
5. Sun, Y.; Tian, Z.; Li, M.; Zhu, C.; Guizani, N. Automated Attack and Defense Framework toward 5G Security. *IEEE Netw.* **2020**, *34*, 247–253. [CrossRef]
6. Bell-Kligler, S.; Shocher, A.; Irani, M. Blind super-resolution kernel estimation using an internal-GAN. In Proceedings of the Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, Vancouver, BC, Canada, 8–14 December 2019.
7. Xu, X.; Yu, C.; Qu, J.; Zhang, L.; Jiang, S.; Huang, D.; Chen, B.; Zhang, Z.; Guan, W.; Ling, Z.; et al. Imaging and clinical features of patients with 2019 novel coronavirus SARS-CoV-2. *Eur. J. Nucl. Med. Mol. Imaging* **2020**, *47*, 1275–1280. [CrossRef] [PubMed]

8. Lv, J.; Wang, C.; Yang, G. PIC-GAN: A parallel imaging coupled generative adversarial network for accelerated multi-channel MRI reconstruction. *Diagnostics* **2021**, *11*, 61. [CrossRef] [PubMed]
9. Wang, X.; Xie, L.; Dong, C.; Shan, Y. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 1905–1914.
10. Rakotonirina, N.C.; Rasoanaivo, A. ESRGAN+: Further improving enhanced super-resolution generative adversarial network. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 3637–3641.
11. Ni, Z.; Yang, W.; Wang, S.; Ma, L.; Kwong, S. Towards unsupervised deep image enhancement with generative adversarial network. *IEEE Trans. Image Process.* **2020**, *29*, 9140–9151. [CrossRef] [PubMed]
12. Xu, B.; Zhou, D.; Li, W. Image Enhancement Algorithm Based on GAN Neural Network. *IEEE Access* **2022**, *10*, 36766–36777. [CrossRef]
13. Shahid, M.; Blanc, G.; Jmila, H.; Zhang, Z.; Debar, H. Generative Deep Learning for Internet of Things Network Traffic Generation. In Proceedings of the 25th IEEE Pacific Rim International Symposium on Dependable Computing, Perth, Australia, 1–4 December 2020; pp. 70–79.
14. Ring, M.; Schlö r, D.; Landes, D.; Hotho, A. Flow-based network traffic generation using generative adversarial networks. *Comput. Secur.* **2019**, *82*, 156–172. [CrossRef]
15. Cheng, A. PAC-GAN: Packet generation of network traffic using generative adversarial networks. In Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 17–19 October 2019; pp. 0728–0734.
16. Rigaki, M.; Garcia, S. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 70–75.
17. Dowoo, B.; Jung, Y.; Choi, C. PcapGAN: Packet capture file generator by style-based generative adversarial networks. In Proceedings of the 8th IEEE International Conference on Machine Learning and Applications, Boca Raton, FL, USA, 16–19 December 2019; pp. 1149–1154.
18. Zhang, C.; Ouyang, X.; Patras, P. ZipNet-GAN: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network. In Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies, Incheon, Republic of Korea, 12–15 December 2017; pp. 363–375.
19. Dong, C.; Loy, C.C.; He, K.; Tang, X. Image super-resolution using deep convolutional networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *38*, 295–307. [CrossRef] [PubMed]
20. Zong, B.; Song, Q.; Min, M.R.; Cheng, W.; Lumezanu, C.; Cho, D.; Chen, H. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
21. Bang, D.; Shim, H. Mggan: Solving mode collapse using manifold-guided training. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 2347–2356.
22. Berahmand, K.; Daneshfar, F.; Salehi, E.S.; Li, Y.; Xu, Y. Autoencoders and their applications in machine learning: A survey. *Artif. Intell. Rev.* **2024**, *57*, 28. [CrossRef]
23. Dai, B.; Wipf, D. Diagnosing and enhancing VAE models. *arXiv* **2019**, arXiv:1903.05789.
24. Niu, Z.; Yu, K.; Wu, X. LSTM-based VAE-GAN for time-series anomaly detection. *Sensors* **2020**, *20*, 3738. [CrossRef] [PubMed]
25. Liu, H.L.; Chen, L.; Zhang, Q.; Deb, K. Adaptively allocating search effort in challenging many-objective optimization problems. *IEEE Trans. Evol. Comput.* **2017**, *22*, 433–448. [CrossRef]
26. Yang, P.; Tang, K.; Yao, X. Turning high-dimensional optimization into computationally expensive optimization. *IEEE Trans. Evol. Comput.* **2017**, *22*, 143–156. [CrossRef]
27. Dang, D.C.; Friedrich, T.; Kötzing, T.; Krejca, M.S.; Lehre, P.K.; Oliveto, P.S.; Sudholt, D.; Sutton, A.M. Escaping local optima using crossover with emergent diversity. *IEEE Trans. Evol. Comput.* **2017**, *22*, 484–497. [CrossRef]
28. Wang, Y.; Xu, C.; Qiu, J.; Xu, C.; Tao, D. Towards evolutionary compression. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 2476–2485.
29. Wang, C.; Xu, C.; Yao, X.; Tao, D. Evolutionary generative adversarial networks. *IEEE Trans. Evol. Comput.* **2019**, *23*, 921–934. [CrossRef]
30. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 10012–10022.
31. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
32. Jiang, Y.; Chang, S.; Wang, Z. Transgan: Two pure transformers can make one strong gan, and that can scale up. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 14745–14758.
33. Lee, K.; Chang, H.; Jiang, L.; Zhang, H.; Tu, Z.; Liu, C. Vitgan: Training gans with vision transformers. *arXiv* **2021**, arXiv:2107.04589.
34. Park, J.; Kim, Y. Styleformer: Transformer based generative adversarial networks with style vector. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 8983–8992.

35. Zhao, L.; Zhang, Z.; Chen, T.; Metaxas, D.; Zhang, H. Improved transformer for high-resolution gans. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 18367–18380.
36. Zhang, Z.; Zhang, H.; Zhao, L.; Chen, T.; Arik, S.; Pfister, T. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2022; Volume 36, pp. 3417–3425.
37. Zhu, Y.; Zhao, Z.; Birke, R.; Chen, L.Y. Permutation-Invariant Tabular Data Synthesis. In Proceedings of the 2022 IEEE International Conference on Big Data (Big Data), Osaka, Japan, 17–20 December 2022; pp. 5855–5864.
38. Kim, J.; Jeon, J.; Lee, J.; Hyeong, J.; Park, N. Oct-gan: Neural ode-based conditional tabular gans. In Proceedings of the Web Conference, Ljubljana, Slovenia, 19–23 April 2021; Volume 2021, pp. 1506–1515.
39. Ring, M.; Wunderlich, S.; Grüdl, D.; Landes, D.; Hotho, A. Flow-based benchmark data sets for intrusion detection. In Proceedings of the 16th European Conference on Cyber Warfare and Security, ACPI, Dublin, Ireland, 20–30 June 2017; pp. 361–369.
40. Leevy, J.L.; Khoshgoftaar, T.M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *J. Big Data* **2020**, *7*, 104. [CrossRef]
41. Maciá-Fernández, G.; Camacho, J.; Magán-Carrión, R.; García-Teodoro, P.; Therón, R. UGR '16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Comput. Secur.* **2018**, *73*, 411–424. [CrossRef]