Policy Research Journal
ISSN (E): 3006-7030 ISSN (P) : 3006-7022 Volume 4, Issue 2, 2026

ADVERSARIAL TRAFFIC GENERATION FOR IDS EVASION: A COMPARATIVE STUDY OF AIDAE AND SGAN-BASED APPROACHES

Sana Mukhtar[1], Fazilat Bibi[2], Rafia Nouman[3], Talha Riaz[*4], M Inam-ur-Rehman[5], Muhammad Shaheer[6], Minahil Fatima[7]

[1,2,3, *4,5,6]*Faculty of Engineering Sciences and Technology, Hamdard University Islamabad, Pakistan*
[7]*Department of Artificial Intelligence and Data Science, FAST National University Islamabad, Pakistan*

[*4]talha.riaz@hamdard.edu.pk

DOI: https://doi.org/10.5281/zenodo.18657119

Keywords
Intrusion Detection Systems, Adversarial Attacks, Generative Adversarial Networks, AIDAE, SGAN-IDS, Self-Attention Mechanism, Machine Learning Security, NSL-KDD Dataset, Cybersecurity, Random Forest.

Article History
Received: 17 December 2025
Accepted: 01 February 2026
Published: 16 February 2026

Corresponding Author: *
Talha Riaz

Abstract

Machine learning-based Intrusion Detection Systems (IDS) have become essential com- ponents of modern cybersecurity infrastructure yet remain vulnerable to adversarial attacks that can compromise their effectiveness. This research presents a comprehensive compara- tive analysis of two state-of-the-art adversarial generation frameworks: Anti-Intrusion De- tection Autoencoder (AIDAE) and Self-Attention Generative Adversarial Network for IDS (SGAN-IDS). Traditional IDS approaches, including signature-based and anomaly-based methods, suffer from significant limitations such as inability to detect zero-day attacks and high false alarm rates. While machine learning-based IDS have addressed some of these shortcomings, they remain susceptible to carefully crafted adversarial examples that can evade detection. This study establishes a uniform experimental framework to systemati- cally evaluate and compare the adversarial effectiveness of AIDAE and SGAN-IDS under identical conditions. Both methods are tested against a Random Forest classifier using the NSL-KDD dataset, enabling direct comparison of their adversarial generation capabilities, training dynamics, and attack strategies. AIDAE combines autoencoder reconstruction with GAN-based adversarial training to generate semantically consistent adversarial sam- ples, while SGAN-IDS leverages self-attention mechanisms to capture long-range depen- dencies and produce globally consistent adversarial traffic. Through quantitative evaluation metrics including accuracy degradation, fooling rate, and confusion matrix analysis, this research provides critical insights into the strengths, weaknesses, and trade-offs of each approach. The findings reveal fundamental vulnerabilities in machine learning-based IDS and highlight the urgent need for adversarial-robust architectures. This study establishes important benchmarks for adversarial traffic generation and contributes to the development of more resilient intrusion detection systems capable of defending against sophisticated at- tacks.

1 Introduction

In today's growing world, digital platforms are the source of connectivity between people, businesses, government, and all kinds of institutions, where we can instantly share information across the world. These platforms have greatly promoted communication, but it has made our digital data, infrastructure, and personal information more vulnerable. We can say that not only individuals, but the entire nation is constantly at risk of cyber-attacks, data leakage, and disruptions on different scales [24, 25].

To avoid such situations, IDS (Intrusion Detection System) has become an important part of cybersecurity. Intrusion Detection System plays an important role in identifying all the ma- licious traffic. It plays an important role in protecting information and systems against such threats [34]. They are generally divided into two types: Signature-based IDS and Anomaly- based IDS. Signature-based IDS detects attacks by analyzing the

activity of the system or net- work and comparing it with the known signatures of previous attacks in the data [25].

They are good at detecting known attacks with non-significant errors and do not produce much false alarms, but they are not very effective in detecting unknown or zero-day attacks. That is why they must be continuously updated with new signatures.

To remove this weakness, researchers developed Anomaly-based IDS. This IDS first ob- serves the basic activities of the network or system as a baseline, but then any activity deviating from this baseline is flagged as anomaly. They cover the weaknesses of signature-based IDS, but while they are effective in detecting unknown and zero-day attacks, this strength brings a new problem. These unusual activities are not necessarily malicious. That is why they also label benign but unusual activities as malicious, which produces large numbers of false alarms [25]. Since both traditional IDS have significant weaknesses, therefore, after discovering the shortcomings of previous works, researchers use Machine learning-based IDS to analyze or estimate network traffic and system activities [21, 24]. They learn from data how to distinguish between normal and malicious or abnormal behavior, and they reduce some shortcomings of signature-based and anomaly-based IDS, such as reducing false alarm rate and detecting un-
known attacks.

In practice, different machine learning models have been used for intrusion detection like Support Vector Machine (SVM) [19] for classification, Random Forest (RF) [18] for large and complex datasets, K-Nearest Neighbor (KNN) [20] for pattern recognition, Convolutional Neural Network (CNN) and Deep Learning [21, 22] for capturing non-linear relationships in traffic. These models have proven to be very important and effective in detecting both known and new and advanced types of attacks [23].

However, despite better detection performance, machine learning-based IDS are not com- pletely foolproof and still have vulnerabilities to adversarial attacks [15, 16, 33]. In an Adver- sarial attack, a malicious actor deliberately creates such network traffic that is called adversarial traffic or adversarial examples to deceive machine learning models. They do this by slightly altering the patterns of traffic signals or network packets so that the machine learning model misclassifies, such as labeling malicious packets as benign [15]. Therefore, since machine learning-based IDS rely entirely on learned patterns, their functioning is based on these pat- terns, which makes them sensitive to such attacks.

To further understand these weaknesses and improve performance, researchers have used different machine learning models to prepare adversarial samples, and malicious samples are made by bringing perturbation in traffic, such as changing the content of packets, timestamps, or length of packets unusually, or perturbations are calculated to exploit the weaknesses of IDS models so that misclassification can happen [16, 17]. Machine learning-based IDS help in identifying these weaknesses.

Researchers have used a powerful method, the method which we call Generative adversarial network (GAN) [7]. This is an effective way to create sophisticated adversarial samples for testing IDS models. Instead of relying on simple rule-based perturbations, they use data-driven learning to make flexible and effective adversarial samples [10, 11]. These prove valuable for testing IDS against advanced threats and for making them more robust through further adversarial training.

However, GAN also has notable weaknesses in IDS. They often suffer from training insta- bility. They face difficulty in capturing long-term dependencies in network traffic and some- times prepare such samples that look statistical but do not have semantic consistency with those attacks in the real world [26].

To overcome these shortcomings, researchers have introduced improved frameworks such as Anti Intrusion Detection Autoencoder (AIDAE) [2] and Self-Attention GAN, which is called SGAN-IDS [1]. AIDAE [2] combines the generative abilities of GAN with the reconstruction power of Autoencoder. This can capture both continuous and discrete traffic features, which makes samples not only statistically realistic but also semantically consistent. SGAN-IDS [1] uses the self-attention mechanism [6, 8], which lets models focus on global dependencies instead of long patterns. This captures long-range relations and produces more sophisticated and diverse adversarial traffic that better simulates complex attack behaviors.

These both advancements have made adversarial generation more realistic and difficult, which makes machine learning-based IDS stronger. Beyond GAN-based frameworks, several other techniques have also been used in adversarial traffic generation, such as Particle Swarm, Genetic Algorithm (GA), Adaptive Adversarial Packet Manipulation (A2PM), Optimization (PSO), and Monte Carlo (MC) approaches [11,13]. Although these techniques perturb network traffic in different ways, their objective is the same: to produce adversarial examples so that IDS performance can be tested and evaluated better.

This research establishes a uniform framework to evaluate the effectiveness of AIDAE [2] and SGAN-IDS [1] in a controlled environment. By testing their adversarial generative abili- ties against Random Forest [18] classification on the NSL-KDD dataset [3], this work provides important insights into the trade-off of these advanced methods and sets a new important bench- mark for preparing realistic machine learning-based IDS.
The primary objective of this research is not to promote cyber-attacks but to reveal the weaknesses of machine learning-based IDS [34], and our main aim is to highlight these vulner- abilities so that in the future they can be secured and strengthened.

The main contributions of this research are summarized as follows:

- We propose a uniform evaluation framework that brings AIDAE [2] and SGAN-IDS [1] together under the same experimental setup and specifically both methods are tested on the NSL-KDD dataset [3] with Random Forest classifier [18] under identical conditions.
- We conduct comparative analysis of adversarial sample generation, which highlights the strengths and weaknesses of AIDAE [2] and SGAN-IDS [1]. This explains what aspects make each method more powerful or weaker.
- We provide quantitative evaluation metrics to systematically assess adversarial effective- ness and to test trade-off. This helps us understand how and where these models work in real-world intrusion detection scenarios.

2 Literature Review

The concept of Intrusion Detection Systems first started from anomaly-based security models, which Denning introduced, and later Anderson improved them. The biggest problem of these early models was that they produced too many false alarms. A major update in IDS research came when Tavallaee et al. published their study "A Detailed Analysis of the KDD CUP 99 Dataset" [3]. From which the NSL-KDD dataset was developed. This dataset became a standard, but the IDS models of that time depended on simple statistical features, because of which attackers could easily evade them. When machine learning was introduced into IDS design, models such as SVM by Cortes and Vapnik [19], KNN by Cover and Hart [20], Random Forest by Breiman [18], and deep learning models such as LSTM networks by Hochreiter and Schmidhuber [22] improved the detection performance significantly.
But large surveys such as Mishra et al.'s ML for IDS [25] and Coulter et al. Intelligent Traffic Analysis [24] show that machine-learning-based IDSs are very sensitive to adversar- ial manipulation. This vulnerability was first formally exposed by Biggio et al.; their paper "Evasion Attacks Against Machine Learning at Test Time" [15] proved that classifiers can be fooled by even a small adversarial noise. Around the same time Goodfellow et al. Introduced GAN [7], which provided new methods for generating realistic synthetic data and opened the way for creating adversarial samples in IDS research.
GAN-based IDS attack research was first done by Hu and Tan in their paper "Generating Adversarial Malware Using GAN" [12], where malware behavior was changed to evade IDS. After that Usama et al. proposed GAN for Launching and Thwarting Adversarial Attacks on IDS [11], which generated adversarial network traffic. This work was pioneering, but these approaches had one major limitation: continuous and discrete features were treated in the same way, because of which the generated samples sometimes became unrealistic.
After this, Lin et al. introduced IDSGAN [10], where the generator learns from the pre- dicted labels of a black-box IDS and generates adversarial traffic near the decision boundary. But the issue with IDSGAN was that it queried the IDS too much, discrete features would get distorted, and the semantic structure was not preserved.
To solve these problems, Chen, Wu, Zhao, Sharma, Blumenstein, and Yu proposed AIDAE in 2020 in their paper "Fooling Intrusion Detection Systems Using Adversarial Autoencoder" [2]. AIDAE combines an encoder, separate continuous and discrete decoders, and a GAN. The encoder maps raw traffic into latent space. The continuous and discrete decoders reconstruct features in a realistic form, and for discrete reconstruction AIDAE uses Gumbel-Softmax. The GAN keeps the latent space natural so that synthetic latent codes appear realistic. A major advantage of AIDAE is that it generates adversarial traffic without using IDS feedback. AIDAE noticeably degraded attack detection on NSL-KDD, UNSW-NB15 [4], and CICIDS2017 [5].
Another revolution in deep learning occurred when Vaswani et al. introduced the self- attention mechanism in "Attention Is All You Need" [8]. Self-attention made it easy to capture long-range dependencies, which is very useful for complex network-traffic data. Using this concept, Aldhaheri and Alhuzali [1] proposed SGAN-IDS. SGAN-IDS uses self-attention in- side the generator so it can understand distant feature relationships and

generate globally con- sistent adversarial samples. SGAN-IDS greatly reduced detection accuracy of multiple IDS models and generated highly deceptive adversarial flows.

Overall, the research progression is clear: adversarial sample generation for IDS began with simple GAN attacks, evolved into black-box guided models such as IDSGAN, moved toward semantically consistent frameworks like AIDAE to preserve feature realism, and finally reached an advanced level with self-attention-based SGAN-IDS. In our research, both state-of-the-art frameworks, AIDAE [2] and SGAN-IDS [1], are evaluated on the common NSL-KDD [3] dataset using a Random Forest [18] classifier, allowing a fair and direct comparison of the actual strengths and weaknesses of each model.

3 Methodology

This section describes the experimental design and the procedural steps used to evaluate ad- versarial traffic generation methods. We present the conceptual framework, data preprocessing steps, baseline classifier configuration, adversarial generation techniques (AIDAE and SGAN- IDS), the experimental protocol, and evaluation metrics.

3.1 Experimental Framework

The primary objective of this research is that two state-of-the-art adversarial sample genera- tion framework AIDAE and SGAN, be systematically compared under the same experimental conditions and their effect on Random Forest based IDS detection performance be quantified. And evaluates the regression in detection performance when the baseline ideas encounter these adversarial samples. This comprehensive evaluation framework enables direct comparison of adversarial effectiveness while maintaining controlled experimental conditions throughout the study.

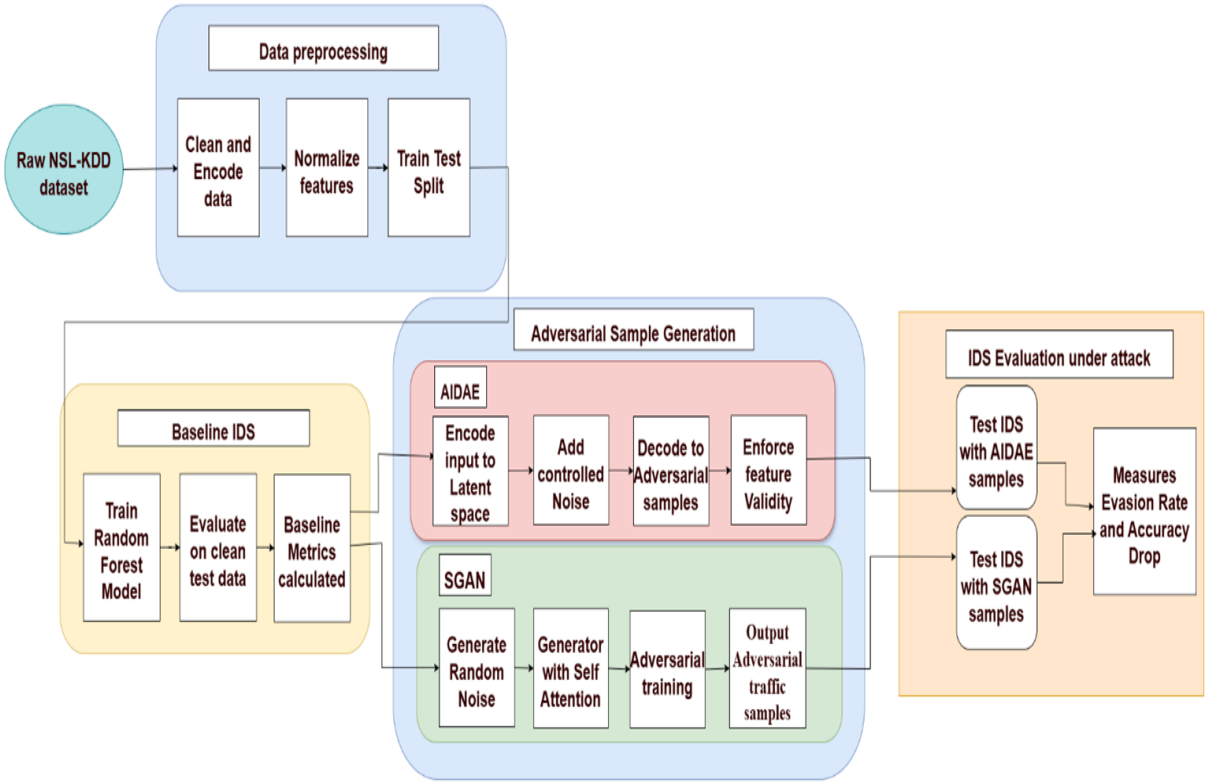

Figure 1: Adversarial sample generation framework AIDAE and SGAN

1.1 Intrusion Detection System

1.1.1 Random Forest

For IDS, we choose random forest algorithm [18] because it is fast and reliable in classification. The model is an

ensemble of decision trees, and its final prediction is based on a majority vote of the individual trees. Random forest is configured with some specific hyperparameters so that the result is reproducible, and the performance is optimal.

Formally, a Random Forest is an ensemble of T decision trees $\{h_t(x; \Theta_t)\}_T$ where Θ_t are independent and identically distributed random vectors. The final prediction for a given input $x \in R_d$ is:

$$\hat{y} = \text{majority vote}\{h_t(x; \Theta_t)\}_T \quad (1)$$

For regression tasks, the prediction is the average:

$$\hat{y} = 1 \Sigma h(x; \Theta) \quad (2)$$

For training, 80 percent of training data is used for training, and 20 percent data is set for validation. We test the model on testing data to evaluate whether the model works correctly on new and unseen data.

Table 1: Random Forest Hyperparameter Configuration

Hyperparameter	*Value*	*Justification*
Number of Estimators (n estimators)	*100*	*Balances performance and computational ef- ficiency*
Max Depth	*None*	*Allows trees to ex- pand until pure leaves are achieved*
Min Samples Split	*2*	*Standard value for un- restricted tree growth*
Min Samples Leaf	*1*	*Permits fine-grained decision boundaries*
Max Features	*sqrt*	*Reduces correla- tion among trees: max features = √n features*
Bootstrap	*True*	*Enables bagging for variance reduction*
Random State	*42*	*Ensures reproducibil- ity across experiments*

3.2 Adversarial Generation Techniques

We created adversarial samples by making small and carefully planned changes in the features [15, 16]. We made such small changes in features so that data seems normal to both humans and ML-based IDS.

First, adversarial samples were prepared in such a way that the record's features were given small changes. These changes were made in such a manner that they do not appear much, but they are enough to deceive the system. The purpose of making these changes is that the system understands it in a wrong way.

Mathematically, given a legitimate sample $\mathbf{x} \in \mathbb{R}^d$ with true label y, we seek an adversarial sample $\mathbf{x}_{adv}$ such that:

$$\mathbf{x}_{adv} = \mathbf{x} + \boldsymbol{\delta}, \qquad \|\boldsymbol{\delta}\|_p \leq \varepsilon \tag{3}$$

$$f(\mathbf{x}_{adv}) \neq y, \quad \text{while} \quad f(\mathbf{x}) = y \tag{4}$$

where $f : \mathbb{R}^d \rightarrow Y$ is the IDS classifier, $\boldsymbol{\delta}$ is the perturbation vector, $\|\|_p$ is the L_p-norm, and ε is the perturbation budget.

3.2.1 AIDAE (Anti Intrusion Detection Autoencoder)

This AIDAE [2] is a model in which the Auto-Encoder and the Generative Adversarial Network (GAN) [7] are combined to design a system that generates adversarial network traffic which confuses the Intrusion Detection System (IDS). Its main purpose is to create synthetic traffic features that look like normal data but are not actually real.

AIDAE's structure consists of two parts: The **Encoder** compresses the original network traffic data (both continuous and discrete features) into a small latent representation. The **De- coder** then reconstructs that compressed data back into its original form; however, instead of making an exact copy, the model slightly modifies the data so that it appears like normal traffic but behaves slightly differently during the detection phase. The structure of the AIDAE is pre- sented in Figure 2. AIDAE's adversarial component comes from its GAN integration [7], where the generator produces random latent codes, and the discriminator checks whether these gener- ated feature codes come from the real distribution or from the synthetic one. This adversarial training helps the model learn the distribution of normal data more accurately and generates traffic that appears normal but is actually adversarial in nature. To guide this learning, AIDAE uses two types of loss functions [2].

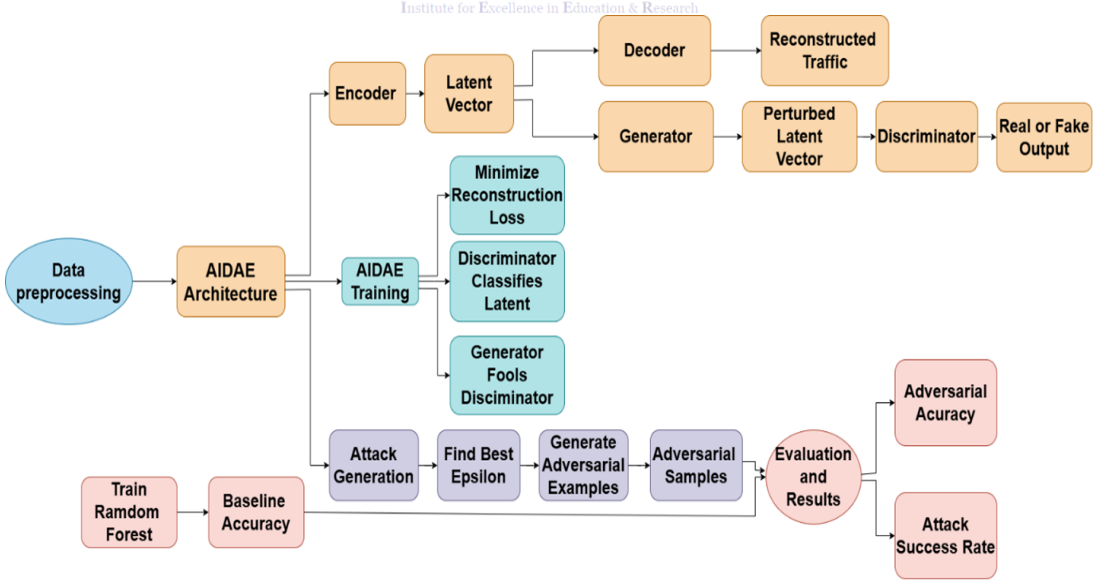

Figure 2: The framework of the AIDAE (anti-intrusion detection autoencoder)

Reconstruction Loss measures how accurately the autoencoder has reconstructed the input data. It ensures that the generated features remain close to the original distribution. **Adversar- ial Loss** helps the generator to create features that can fool the IDS, making it classify malicious traffic as normal.

By combining both losses, AIDAE [2] produces adversarial samples that are close to real traffic but effectively deceive the IDS. In this project, AIDAE was applied to the NSL-KDD dataset [3] and tested on a pre-trained Random Forest [18] IDS model. The results showed that when the IDS was tested with AIDAE generated data,

its accuracy dropped, which proves that the model has the ability to mislead detection systems.

3.2.2 Mathematical Formulation of AIDAE

The training objective of the AIDAE model combines the **Autoencoder reconstruction loss** and the **Adversarial loss** from the GAN component.

Let $\mathbb{R}^d$ be the input space. The encoder $E : \mathbb{R}^d \to \mathbb{R}^l$ maps input $\mathbf{x} \in \mathbb{R}^d$ to a latent representation $\mathbf{z} = E(\mathbf{x}) \in \mathbb{R}^l$, where l is the latent dimension. The decoder $D : \mathbb{R}^l \to \mathbb{R}^d$ reconstructs the input from the latent representation: $\hat{\mathbf{x}} = D(\mathbf{z}) = D(E(\mathbf{x}))$.

The generator $G : \mathbb{R}^l \to \mathbb{R}^l$ produces perturbations in the latent space: $\boldsymbol{\delta} = G(\mathbf{z})$. The adversarial latent representation is:

$$\mathbf{z}_{adv} = \mathbf{z} + \varepsilon \cdot G(\mathbf{z}) \quad (5)$$

where ε is the perturbation budget.

The discriminator $Dis : \mathbb{R}^l \to [0, 1]$ distinguishes between real latent codes $\mathbf{z}_{real} = E(\mathbf{x})$ and fake latent codes $\mathbf{z}_{fake} = \mathbf{z}_{adv}$.

The overall objective function can be expressed as:

$$\min_{G,E} \max_{D} E_{\mathbf{c}\sim P(\mathbf{c})}[\log Dis(\mathbf{c})] + E_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - Dis(G(\mathbf{z})))] + \lambda L_{rec} \quad (6)$$

where:

- G: Generator
- *Dis*: Discriminator
- *E*: Encoder
- L_{rec}: Reconstruction loss
- $P(\mathbf{c})$: Real data distribution
- $P(\mathbf{z})$: Latent distribution from encoder
- λ: Weight coefficient balancing the losses

The network structure of AIDAE is shown in Table 2, where input-dim is the dimension of the input features, $\mathbf{z}$ is representing the 64-dimensional latent vector, $\boldsymbol{\delta}$ is the perturbation generated in the latent space, λ is the weight coefficient balancing MSE and BCE losses, and *lr* is the learning rate of the Adam optimizer set to 0.001 to fool the target IDS classifier.

Table 2: The network structure of the AIDAE

Component	Encoder (E)	Decoder (D)	Perturbation Generator (G)	Discriminator (Dis)
Role	Converts X to latent z	Reconstructs X' from latent z	Generates pert. $\boldsymbol{\delta}$ in latent space	Distinguishes real/fake z
Architecture	Linear(in → 256) Linear(256 → 128) Linear(128 → 64)	Linear(64 → 128) Linear(128 → 256) Linear(256 → in)	Linear(64 → 128) Linear(128 → 128) Linear(128 → 64)	Linear(128 → 64) Linear(64 → 1)
Input Size	input dim features	64-dim latent	64-dim latent	64-dim latent
Output Size	64-dim latent	input dim features	64-dim pert.	Probability [0, 1]
Activation	ReLU ReLU None	ReLU ReLU Sigmoid	ReLU Tanh	LeakyReLU(0.2) Sigmoid
Loss Function	MSE	MSE	BCE +λ · MSE	BCE
Optimizer	Adam (lr=0.001)	Adam (lr=0.001)	Adam (lr=0.005)	Adam (lr=0.001)
Training Data	Normal only	Normal only	Normal only	Real + Fake latent

3.2.3 Training Procedure

The AIDAE framework is trained for 50 epochs with a batch size of 128 using only normal traffic samples. The ADAM optimizer is used for all components, with learning rates of 0.001 for the Encoder, Decoder, and discriminator, and 0.005 for the Generator, while keeping the default momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The reconstruction loss weight is set to $\lambda = 0.5$, the latent dimension is 64, and the training epsilon is fixed at $\varepsilon = 0.3$. During training, three optimization steps are performed in an alternating manner.

First, the autoencoder (Encoder-Decoder) is updated by minimizing the reconstruction loss:

$$L_{rec} = \|\mathbf{x} - D(E(\mathbf{x}))\|^2 = \sum_{i=1}^{d} (x_i - \hat{x}_i)^2 \quad (7)$$

For mixed feature types (continuous and discrete), AIDAE employs separate decoders. For continuous features, it uses Mean Squared Error (MSE):

$$L_{cont} = \|\mathbf{x}_{cont} - D_{cont}(E(\mathbf{x}))\|^2 \quad (8)$$

For discrete features, it uses the Gumbel-Softmax relaxation for differentiable sampling:

$$\hat{x}_{disc,i} = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \quad (9)$$

where π_i are class probabilities, g_i Gumbel(0, 1) is Gumbel noise, and τ is the temperature parameter.

Then, the discriminator is updated to distinguish real latent representation $z_{real} = E(\mathbf{x})$ from adversarial latent $z_{fake} = z_{real} + \varepsilon \cdot G(z_{real})$ using the binary cross-entropy loss:

$$L_{disc} = -[\log(Dis(z_{real})) + \log(1 - Dis(z_{fake}))] \quad (10)$$

Finally, the generator is updated twice per batch to fool the discriminator while preserv- ing reconstruction consistency. Its objective combines adversarial and reconstruction terms, defined as:

$$L_{gen} = \text{BCE}(Dis(z_{adv}), 1) + \lambda\|\mathbf{x} - D(z_{adv})\|^2 \quad (11)$$

Algorithm 1 Adversarial Autoencoder-Based IDS Framework (AIDAE)

Require: Normal network traffic features $\mathbf{F} \quad \mathbf{f}_1, \{\mathbf{f}_2, \ldots, \mathbf{f}_m \quad \}$
Ensure: Trained AIDAE model, adversarial samples
Step 1: Data Preprocessing
1: Load dataset, apply one-hot encoding to categorical features
2: Normalize using Min-Max normalization: $\mathbf{x}_{norm} = \frac{\mathbf{x}\_\min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}$
3: Split data into normal and malicious samples
Step 2: Autoencoder Training
4: Encode: $z = E(\mathbf{x})$
5: Reconstruct: $\hat{\mathbf{x}} = D(\mathbf{z})$
6: Compute reconstruction loss: $L_{ae} = \|\mathbf{x} \quad \overline{\mathbf{x}}\|^2$
7: Update Autoencoder parameters
Step 3: Discriminator Training
8: Encode normal samples to obtain real latent vectors: $\mathbf{z}_{real} = E(\mathbf{x})$
9: Generate perturbations: $\mathbf{p} = G(\mathbf{z}_{real})$
10: Create fake latent vectors: $\mathbf{z}_{fake} = \mathbf{z}_{real} + \varepsilon \quad \mathbf{p} \quad \cdot$
11: Train Discriminator: real latent $\mathbf{z}_{real}$ ~~1~~ fake latent $\mathbf{z}_{fake}$ ~~0~~
12: Update discriminator using $L_{disc} =$ $[\log(Dis(\mathbf{z}_{real})) + \log(1 \quad Dis(\mathbf{z}_{fake}))]$
Step 4: Generator Training
13: Generate perturbations: $\mathbf{p} = G(\mathbf{z}_{real})$
14: Create adversarial latent vectors: $\mathbf{z}_{adv} = \mathbf{z}_{real} + \varepsilon \quad \mathbf{p} \quad \cdot$
15: Decode adversarial traffic: $\hat{\mathbf{x}}_{adv} = D(\mathbf{z}_{adv})$
16: Compute Generator loss: $L_{gen} = \text{BCE}(Dis(\mathbf{z}_{adv}), 1) + \lambda \|\mathbf{x} \quad \overline{\mathbf{x}}_{adv}\|^2$
17: Update Generator parameters
Step 5: IDS Training
18: Train Random Forest classifier on clean dataset to obtain baseline accuracy
Step 6: Adversarial Attack Generation
19: **for** malicious samples $\mathbf{x}_m$ **do**
20: Encode: $\mathbf{z}_m = E(\mathbf{x}_m)$
21: Generate perturbations: $\mathbf{p} = G(\mathbf{z}_m)$
22: Create adversarial latent vectors: $z' = z_m + \varepsilon \cdot \mathbf{p}$
23: Decode adversarial traffic: $\mathbf{x}_{adv} = D(z')$
24: Select ε with maximum evasion
25: **end for**

Step 7: Attack Evaluation
26: Evaluate attack by computing adversarial accuracy, accuracy drop, fooling rate, and attack success rate

3.2.4 SGAN-IDS

SGAN-IDS [1] is a model based on a Generative Adversarial Network (GAN) [7] that creates synthetic data. It has two parts: one part is the generator and the other is the discriminator. The generator creates synthetic data while the discriminator's job is to try to recognize these samples. In SGAN-IDS, the discriminator and generator perform opposing tasks, and a game- like process runs where the generator tries to deceive the system while the discriminator tries to catch that deception [7]. From this process the generator learns better each time and tries to fool the system; this is based on continuous learning on both sides, which is why SGAN is considered more powerful than AIDAE. The structure of the SGAN is presented in Figure 3.

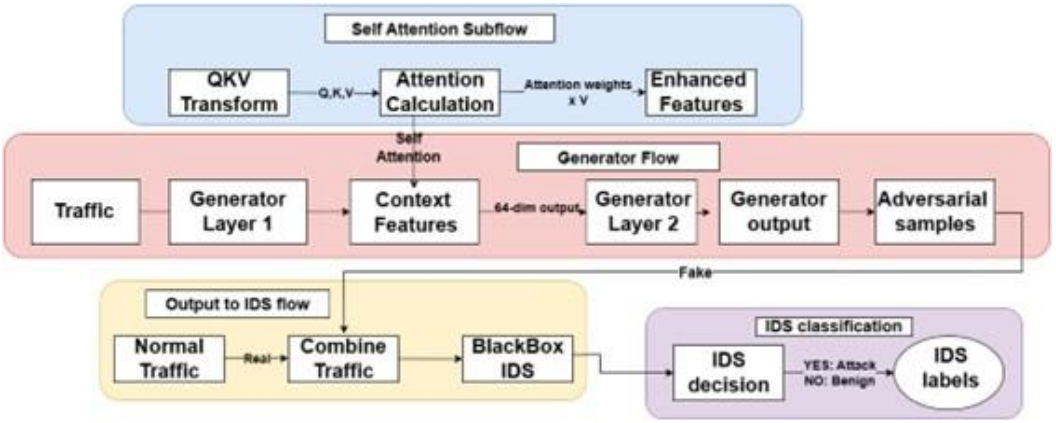

Figure 3: The framework of the SGAN (Self-attention GAN)

The SGAN-IDS model [1] uses another important mechanism called self-attention [6, 8, 9]. This is a mechanism that helps the model understand how different parts of an input are related to each other. When data is being processed the model does not depend on a single feature but understands relationships between the various features of the whole input. This helps the model capture long-range dependencies [27, 28], for example patterns in network traffic that are normally ignored when they are far apart. When the self-attention mechanism is applied in SGAN [1, 6] the model becomes more powerful because both the generator and discriminator work by understanding global feature dependencies [32]. As a result, the generator produces adversarial samples that are not only statistically realistic but also semantically similar to real traffic. This process helps generate complex and realistic adversarial traffic which is very ef- fective for testing and understanding their weaknesses.

3.2.5 Mathematical Formulation of Self-Attention

The self-attention mechanism computes a weighted sum of all elements in a sequence, allowing each element to attend to all others. Given an input feature map $\mathbf{h} \in R^{C\times N}$ where C is the number of channels and N is the number of feature locations, the self-attention mechanism transforms $\mathbf{h}$ into query, key, and value matrices:

$$\mathbf{Q} = \mathbf{W}_q \mathbf{h} \in R^{C'\times N} \quad (12)$$

$$\mathbf{K} = \mathbf{W}_k \mathbf{h} \in R^{C'\times N} \quad (13)$$

$$\mathbf{V} = \mathbf{W}_v \mathbf{h} \in R^{C\times N} \quad (14)$$

where $\mathbf{W}_q \in R^{C'\times C}$, $\mathbf{W}_k \in R^{C'\times C}$, and $\mathbf{W}_v \in R^{C\times C}$ are learned weight matrices, and $C' = C/8$ is typically chosen for computational efficiency.

The self-attention matrix is constructed mathematically as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}^T \mathbf{K}}{\sqrt{d_k}}\right)\mathbf{V} \quad (15)$$

where d_k is the dimension of the key vectors, and the scaling factor $\sqrt{d_k}$ prevents the dot products from growing too large in magnitude.

More explicitly, the attention weight $\beta_{j,i}$ indicating the extent to which the model attends to the i-th location when synthesizing the j-th location is:

$$\beta_{i,j} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}, \quad \text{where } s_{ij} = \frac{Q(:,i)^T K(:,j)}{\sqrt{d_k}} \quad (16)$$

The output of the self-attention layer is then:

$$h_j' = \sum_{i=1}^{N} \beta_{j,i} V(:,i) \in R^C \quad (17)$$

This gives the model the facility that it selectively focuses on the most relevant parts of the input when it is processing any element. Then, when this is applied in SGAN [1], this ability to understand these dependencies means that generator and discriminator both work with a complete view of the features. Consequently, the generator produces such an adverse sample which is not only statistically realistic but also semantically close to real traffic, which helps in making complex and effective adverse traffic that is used to test and understand the weaknesses of the system.

The network structures of SGAN are shown in Table 3. The generator combines real traffic features with a 10-dimensional noise vector [7], while the self-attention mechanism [6, 8] uses **Q**, **K**, **V** (Query, Key, Value) matrices to refine 64-dimensional hidden representations. The generator's loss combines BCE with twice-weighted MSE to confuse the black-box IDS [10, 11].

Table 3: The network structure of the SGAN-IDS (Part 1: Components and Roles)

Component	Architecture	Specifications
Generator (G)	FC(input+noise → 32) + BN FC(32 → 64) + BN Self-Attention(64 → 64) FC(64 → input dim)	**Role:** Generate adversarial samples **Input:** Real traffic + Noise (10-dim) **Output:** Adversarial sample (input dim) **Activation:** ReLU ReLU → Self-Attention Linear **Loss:** BCE (Disc) + 2 IDS confuse (MSE) **Optimizer:** Adam (lr=0.0002) **Training:** Normal traffic
Self- Attention	Q, K, V Linear(64 64) + Softmax	**Role:** Refine features inside generator **Input:** 64-dim hidden vector **Output:** 64-dim refined vector **Activation:** Softmax **Training:** Integrated in Generator
Discriminator (D)	FC(input dim 64) FC(64 32) FC(32 1) + Dropout(0.2)	**Role:** Distinguish real vs fake samples **Input:** Real or fake feature vector **Output:** Probability (0-1) **Activation:** ReLU ReLU Sigmoid **Loss:** BCE **Optimizer:** Adam (lr=0.0002) **Training:** Real + Generator fake samples

Table 4: The network structure of the SGAN-IDS (Part 2: IDS Components)

Component	Architecture	Specifications
Black-Box IDS (RF)	Random Forest (100 trees)	**Role:** Classify attacks vs benign **Input:** input dim features **Output:** Attack/Benign label **Loss:** CE (internal) **Training:** Real traffic only
Label Encoder	Scikit-learn LabelEncoder	**Role:** Encode string labels to numeric **Input:** String labels **Output:** 0-N encoded labels **Training:** Training/test labels

3.2.6 Training Procedure

The proposed SGAN-IDS framework is trained for 50 epochs using batch size of 128 with normal traffic samples only. The Adam optimizer is configured with learning rate lr = 0.0002 for both Generator and Discriminator networks [7]. The training follows a three-step alternating optimization process:

First, the Discriminator is updated to distinguish between real normal traffic samples and the generator-produced fake samples using binary cross-entropy loss [7]:

$$L_{disc} = \text{BCE}(Dis(\mathbf{x}_{real}), 1) + \text{BCE}(Dis(\mathbf{x}_{fake}), 0) \quad (18)$$

where $\mathbf{x}_{fake} = G(\mathbf{x}_{real}, \mathbf{z})$ with $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ being 10-dimensional Gaussian noise.

Second, the Generator is updated in two phases:

(a) To fool the Discriminator using adversarial loss [7]:

$$L_{adv} = \text{BCE}(Dis(G(\mathbf{x}, \mathbf{z})), 1) \quad (19)$$

(b) To confuse the black-box Random Forest IDS classifier [18] by minimizing the MSE between predicted labels and zero target [10, 11]:

$$L_{confuse} = \text{MSE}(RF(G(\mathbf{x}, \mathbf{z})), \mathbf{0}) \quad (20)$$

Finally, the combined Generator loss integrates both objectives with a weight factor of 2 for the IDS confusion term [10]:

$$L_{gen} = L_{adv} + 2 \times L_{confuse} \quad (21)$$

The self-attention mechanism [6,8] embedded within the Generator refines the 64-dimensional hidden representations during each forward pass to enhance the quality of adversarial perturba- tions [1].

3.3 Dataset

This section describes the NSL-KDD dataset used in this research paper, its content, type, ori- gins, and the preprocessing steps applied to prepare the data for model training and evaluation.

Algorithm 2 Self-Attention GAN-Based IDS Framework (SGAN-IDS)

Require: Normal network traffic features $\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m\}$

Ensure: Trained SGAN-IDS model, adversarial samples

Step 1: Data Preprocessing

1: Load dataset, apply one-hot encoding to categorical features

2: Normalize using Min-Max normalization: $\mathbf{x}_{norm} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}$

3: Split data into normal and malicious samples

Step 2: Generator Forward Pass with Self-Attention

4: Combine input traffic features with 10-dimensional random noise vector $\mathbf{z} \in \mathbb{R}^{10} \sim \mathcal{N}(0, \mathbf{I})$

5: Compute query, key, value matrices: $\mathbf{Q} = \mathbf{W}_q\mathbf{h}$, $\mathbf{K} = \mathbf{W}_k\mathbf{h}$, $\mathbf{V} = \mathbf{W}_v\mathbf{h}$

6: Apply self-attention mechanism:

Attention(**Q**, **K**, **V**) = softmax $\frac{\mathbf{Q}^T\mathbf{K}}{\sqrt{d_k}}\mathbf{V}$

7: Obtain refined representation: $\mathbf{h}_{refined}$ = Attention(**Q**, **K**, **V**)
8: Generate adversarial samples: $\mathbf{x}_{fake}$ = G(**x**, *z*)
Step 3: Discriminator Training
9: Train Discriminator to distinguish real normal traffic $\mathbf{x}_{real}$ from generated samples $\mathbf{x}_{fake}$
10: Compute discriminator loss: L_{disc} = BCE(*Dis*($\mathbf{x}_{real}$), 1) + BCE(*Dis*($\mathbf{x}_{fake}$), 0)
11: Update Discriminator parameters
Step 4: Generator Training (Two-Phase Update)
12: Generate adversarial samples: $\mathbf{x}_{fake}$ = G(**x**, *z*)
13: Compute adversarial loss: L_{adv} = BCE(*Dis*(G(**x**, *z*)), 1)
14: Compute IDS confusion loss: $L_{confuse}$ = MSE(*RF* (G(**x**, *z*)), 0) 15: Compute combined generator loss: $L_{gen} = L_{adv} + 2$ $L_{confuse}$ 16: Update Generator parameters
Step 5: IDS Training
17: Train Random Forest classifier on clean (normal) network traffic samples to obtain baseline detection accuracy
Step 6: Adversarial Attack Generation
18: **for** malicious samples $\mathbf{x}_m$ **do**
19: Generate adversarial traffic: $\mathbf{x}_{adv}$ = G($\mathbf{x}_m$, *z*)
20: Apply self-attention refinement
21: Select adversarial samples with maximum evasion capability
22: **end for**
Step 7: Attack Evaluation
23: Evaluate attack by computing adversarial accuracy, accuracy drop, fooling rate, and attack success rate

---

3.3.1 NSL-KDD

For this research, we used the NSL-KDD dataset because of its widespread use in network intrusion detection research. NSL-KDD is a refined version of the KDD Cup '99 dataset, ad- dressing critical issues such as redundant records and class imbalance that plagued the original dataset. The original KDD Cup '99 dataset was created from the 1998 DARPA dataset by MIT Lincoln Laboratory, collected from network data over a 9-week period.
Formally, the NSL-KDD dataset can be represented as:

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \quad (22)$$

where $\mathbf{x}_i \in R^d$ is the feature vector, $y_i \in Y$ = {Normal, DoS, Probe, R2L, U2R} . The dataset comprises 41 features, as shown in Table 4, for each network connection, di- vided into four categories: basic features (1-9) extracted from TCP/IP connections, content features (10-22) examining TCP packet payload, time-based traffic features (23-31) analyz- ing patterns within 2-second windows, and host-based traffic features (32-41) detecting attacks over longer periods [36, 37]. Among these, some features are numerical such as duration and count, while others are categorical including protocol type (3 values), service (60 values), and flag (11 values) [37]. The dataset contains records labeled as either normal or as one of 24 dif- ferent attack types grouped into four main categories: Denial of Service (DoS), Probe, Remote to Local (R2L), and User to Root (U2R) [36], as shown in Table 5, with the test data includ- ing 14 additional attack types not present in training data, making detection more realistic and challenging.
Processing the dataset involves normalizing numerical features using min-max scaling to transform values into a range between 0 and 1:

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (23)$$

and converting categorical features to numerical format through one-hot encoding. For a categorical feature with k possible values, one-hot encoding creates a k-dimensional binary vector:

$$\mathbf{x}_{onehot} = [1_{x=v1}, 1_{x=v2}, \ldots, 1_{x=vk}] \quad (24)$$

where $1_{x=vi}$ is the indicator function that equals 1 if $x = v_i$ and 0 otherwise.

This encoding expands the original 41-dimensional feature space into a 121-dimensional representation [37].

Table 5: The 41 features of the NSL-KDD dataset

No.	Feature	No.	Feature
1	Duration	22	is guest login
2	protocol type	23	Count
3	Service	24	srv count
4	Flag	25	serror rate
5	src bytes	26	srv serror rate
6	dst bytes	27	rerror rate
7	Land	28	srv rerror rate
8	wrong fragment	29	same srv rate
9	Urgent	30	diff srv rate
10	Hot	31	srv diff host rate
11	num failed logins	32	dst host count
12	logged in _	33	dst host srv count
13	num compromised	34	dst host same srv rate
14	root shell	35	dst host diff srv rate
15	su attempted	36	dst host same src port rate
16	num root	37	dst host srv diff host rate
17	num file creations	38	dst host serror rate
18	num shells	39	dst host srv serror rate
19	num access files	40	dst host rerror rate
20	num outbound cmds	41	dst host srv rerror rate
21	is host login		

Table 6: NSL-KDD attack types and classes

Attack Class	Attack Type	Sample Relevant Feature	Example
DoS	Apache2, Back, Pod, Process table, Worm, Neptune, Smurf, Land, Udpstorm, Teardrop	percentage of packets with errors, source bytes	Syn flooding
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint	source bytes, duration of the connection	Port scanning
R2L	Httptunnel, Snmpgetattack, Snmpguess, Guess Password, Imap, Warezclient, Ftp write, Phf, Multihop, Warezmaster,	number of shell prompts invoked, number of file creations	Buffer overflow
U2R	Buffer overflow, Xterm, SQL attack, Perl, Loadmodule, Loadmodule, Ps, Rootkit	service requested, connection duration, num of failed login attempts	Password guessing

3.4 Performance Evaluation Metrics

To comprehensively assess the effectiveness of the proposed adversarial attack framework, we employ several standard evaluation metrics [24, 25] that measure both the attack's impact on the IDS and the quality of generated adversarial samples.

3.4.1 Accuracy

Accuracy measures the proportion of correctly classified samples out of the total test samples [25]. It is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (25)$$

where TP (True Positives) represents correctly classified attack samples, TN (True Neg- atives) represents correctly classified normal samples, FP (False Positives) represents normal samples misclassified as attacks, and FN (False Negatives) represents attack samples misclas- sified as normal [25].

3.4.2 Precision

Precision measures the proportion of correctly predicted instances among all instances pre- dicted for a specific class [24, 25]. For each class, it is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (26)$$

High precision indicates that when the model predicts a particular attack class, it is likely to be correct, minimizing false alarms [23, 25].

3.4.3 Recall

Recall measures the proportion of correctly predicted instances among all actual instances of that class [25]. It is computed as:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (27)$$

High recall indicates that the model successfully identifies most instances of a particular attack type, minimizing missed detections [23, 25].

3.4.4 F1-Score

F1-Score is the harmonic mean of precision and recall [25], providing a balanced metric that accounts for both false positives and false negatives. It is calculated as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (28)$$

The F1-score is particularly useful for imbalanced datasets where accuracy alone may be misleading [23, 25].

For multi-class classification, we compute macro-averaged and weighted-averaged metrics.
The macro-averaged F1-score is:

$$F1_{macro} = \frac{1}{|Y|} \sum_{c \in Y} F1_c \quad (29)$$

The weighted-averaged F1-score is:

$$F1_{weighted} = \frac{1}{N} \sum_{c \in Y} N_c \cdot F1_c \quad (30)$$

where N_c is the number of samples in class c, and $N = \sum_{c \in Y} N_c$.

3.4.5 Fooling Rate

The fooling rate measures the proportion of adversarial examples that are misclassified by the IDS [15, 35]:

$$\text{Fooling Rate} = \frac{\text{Number of adversarial samples misclassified}}{\text{Total number of adversarial samples}} \quad (31)$$

3.4.6 Accuracy Drop

The accuracy drop quantifies the degradation in IDS performance when tested on adversarial samples:

$$\text{Accuracy Drop} = \text{Accuracy}_{clean} - \text{Accuracy}_{adv} \quad (32)$$

3.4.7 Confusion Matrix

The Confusion Matrix provides a detailed visualization of the classifier's performance across all attack categories [24, 25], showing the distribution of true labels versus predicted labels. Each cell (i, j) in the matrix represents the number of samples with true label i that were predicted as label j. For a K-class classification problem, the confusion matrix $\mathbf{C} \in \mathrm{R}^{K\times K}$ is defined as:

$$C_{ij} = |\{\text{samples with true label } i \text{ predicted as label } j\}| \quad (33)$$

This matrix helps identify which attack types are most successfully evaded and which are still detected by the IDS after adversarial manipulation [15, 33, 34].

3.4.8 Training Loss Curves

We track the training loss curves for all model components across epochs to monitor conver- gence behavior [7, 26]. For the AIDAE framework, we monitor:

- **Autoencoder Loss (AE)**: Reconstruction loss measuring how well the autoencoder re- constructs input features [2]:

$$L_{ae} = \|\mathbf{x} - D(E(\mathbf{x}))\|^2 \quad (34)$$

- **Discriminator Loss (D)**: Binary cross-entropy loss for distinguishing real from fake latent codes [7]:

$$L_D = -\mathrm{E}_{z\sim P(z_{real})}[\log Dis(\mathbf{z})] - \mathrm{E}_{z\sim P(z_{fake})}[\log(1 - Dis(\mathbf{z}))] \quad (35)$$

- **Generator Loss (G)**: Combined adversarial and reconstruction loss for generating effec- tive perturbations [7, 10]:

$$L_G = \mathrm{E}_{z\sim P(z)}[\log(1 - Dis(G(\mathbf{z})))] + \lambda\|\mathbf{x} - D(\mathbf{z} + \varepsilon G(\mathbf{z}))\|^2 \quad (36)$$

For the SGAN-IDS framework, we monitor:

- **Generator Loss**: $L_G = \mathrm{BCE}(Dis(G(\mathbf{x}, z)), 1) + 2 \cdot \mathrm{MSE}(RF(G(\mathbf{x}, z)), \mathbf{0})$
- **Discriminator Loss**: $L_D = \mathrm{BCE}(Dis(\mathbf{x}_{real}), 1) + \mathrm{BCE}(Dis(\mathbf{x}_{fake}), 0)$

These curves ensure stable adversarial training throughout the learning process and help identify potential issues such as mode collapse or training instability [7, 26].

4 Results and Discussion

4.1 Experimental Setup

This study evaluated two adversarial attack methods SGAN [1] and AIDAE [2] against a Ran- dom Forest based IDS [18] using the NSL-KDD dataset [3, 36] (125,973 training samples, 22,544 test samples, 122 features [36, 37], 36 attack categories [36]).

4.1.1 Baseline IDS Performance

The Random Forest classifier [18] achieved 91.38% accuracy on original test data with strong detection rates for major attacks: neptune (99.88% recall), portsweep (100%), smurf (100%), and warezclient (100%). Table 6 summarizes baseline performance [25, 36, 37].

Table 7: Baseline IDS Performance Metrics

Metric	Value
Accuracy	91.38%
Precision (weighted avg)	87.02%
Recall (weighted avg)	91.38%
F1-Score (weighted avg)	87.64%

4.1.2 AIDAE Attack Results

4.1.3 Training Dynamics

Figure 4 shows AIDAE's training loss curves over 50 epochs. The autoencoder loss decreased from 0.0010 to 0.0001, indicating effective reconstruction [2]. However, the discriminator loss collapsed to near-zero by epoch 20 while the generator loss increased to 18.70, revealing discriminator dominance [7, 26] that limited adversarial sample diversity [2, 12].

Figure 4: AIDAE training loss curve

4.1.4 Attack Performance

AIDAE reduced IDS accuracy from 91.40% to 50.90% (accuracy drop: 40.50%, fooling rate: 41.89%) [2, 15, 35]. Table 7 presents detailed metrics [25, 33, 34].

Table 8: AIDAE Attack Performance

Metric	Value
Accuracy on Original Test Data	91.40%
Accuracy on Adversarial Test Data	50.90%
Accuracy Drop	40.50%
Fooling Rate	41.89%
Training Epochs	50
Final Autoencoder Loss	0.0001
Final Generator Loss	18.70
Final Discriminator Loss	0.0000

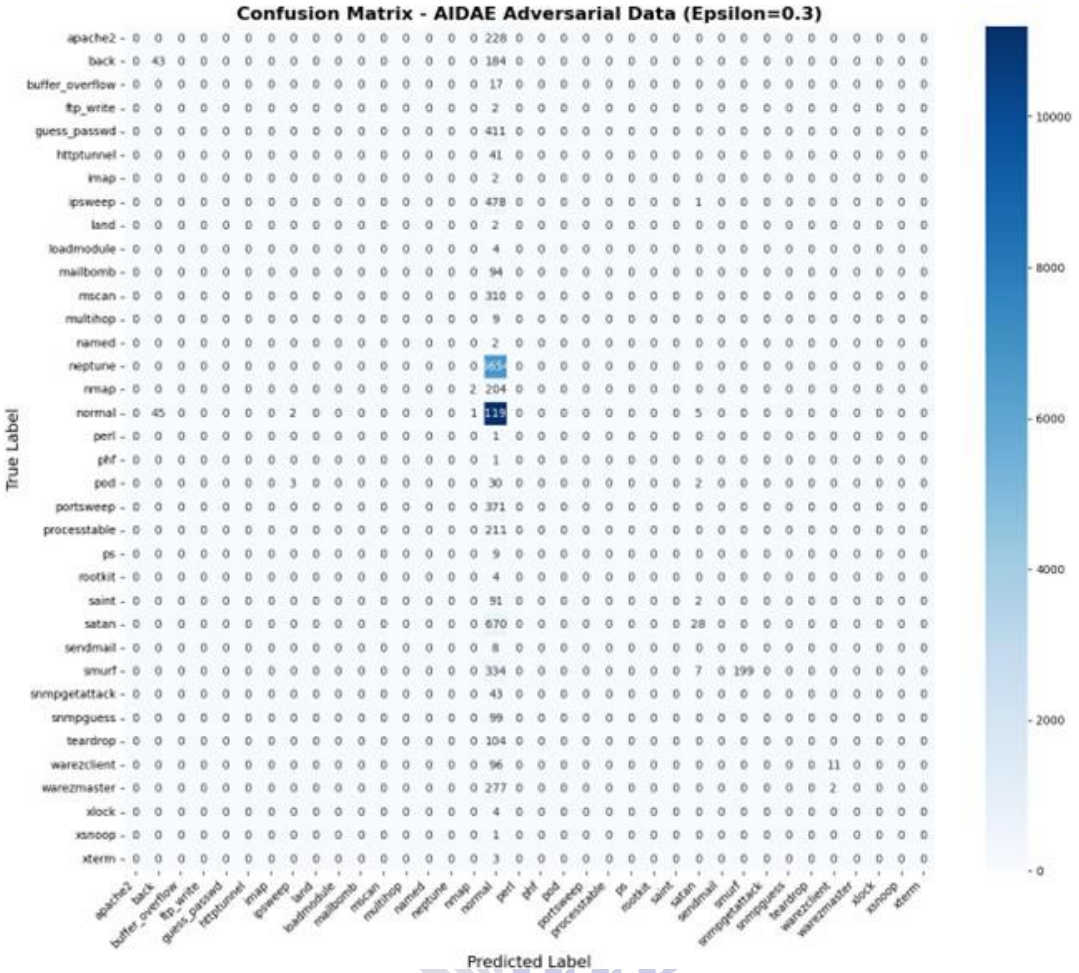

Figure 5: Confusion matrix for AIDAE

4.1.5 SGAN Attack Results

4.1.6 Training Dynamics

Figure 5 illustrates SGAN's training dynamics with dual-axis visualization. The generator loss (blue) stabilized around 220-250 while the discriminator loss (red) converged to approximately
1.29. This balanced adversarial training [1, 7], where neither component dominates, indicates healthy GAN dynamics [7, 26] and effective adversarial sample generation [1, 6].

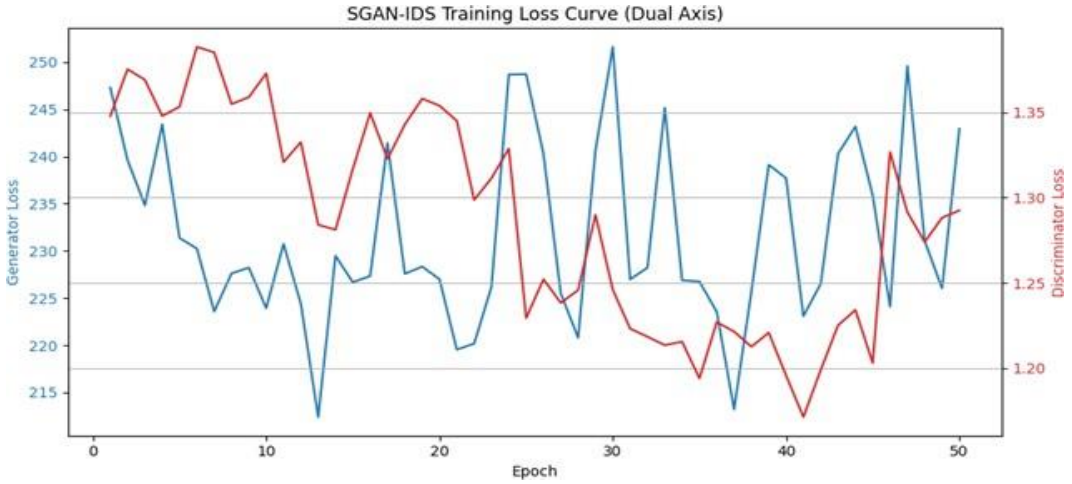

Figure 6: SGAN-IDS training loss curves

4.1.7 Attack Performance

SGAN achieved devastating effectiveness, reducing IDS accuracy from 91.38% to 21.85% (accuracy drop: 69.53%, fooling rate: ~78%) [1, 6, 10]. Table 8 summarizes SGAN's impact [1, 25, 33].

Table 9: SGAN Attack Performance

Metric	Value
Accuracy on Original Test Data	91.38%
Accuracy on Adversarial Test Data	21.85%
Accuracy Drop	69.53%
Fooling Rate	~78%
Training Epochs	50
Final Generator Loss	242.92
Final Discriminator Loss	1.29

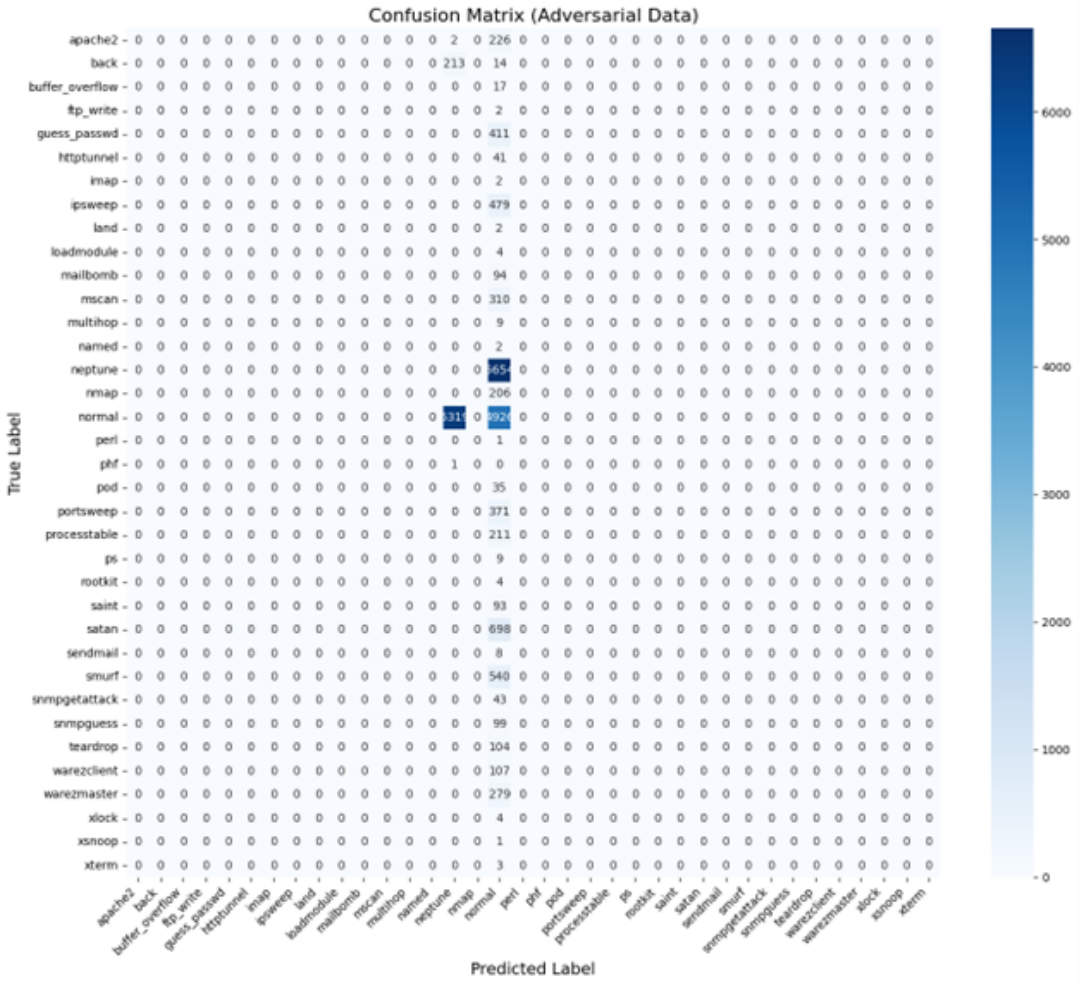

Figure 7: Confusion matrix for SGAN

4.2 Comparative Analysis

Table 9 provides direct comparison between attack methods [1, 2, 25, 33]. Table 10: SGAN vs AIDAE Comparative Performance

Performance Metric	SGAN	AIDAE	Difference
Original Test Accuracy	91.38%	91.40%	-0.02%
Adversarial Test Accuracy	21.85%	50.90%	-29.05%
Accuracy Drop	69.53%	40.50%	+29.03%
Fooling Rate	~78%	41.89%	+36.11%
Relative Effectiveness	100%	53.7%	+86%
Attack Strategy	Broad Misclassification	Normal Mimicry	—
Training Stability	Balanced	Discriminator Dominant	—

4.3 Discussion

4.3.1 Security Implications

The 78% fooling rate demonstrates critical vulnerabilities in ML-based IDS [1,15,34]. SGAN's ability to reduce accuracy to 21.85% represents a severe security gap enabling undetected net- work infiltration [1, 35]. Organizations deploying ML-based IDS must prioritize adversarial robustness [24, 25, 33].

4.3.2 Why SGAN Outperforms AIDAE

SGAN's superiority stems from:

1. **Balanced Training**: SGAN achieves Nash equilibrium in the minimax game [1, 7, 26]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(\mathbf{z})))] \quad (37)$$

This enables continuous improvement in both generator and discriminator.

2. **Self-Attention Mechanism**: The ability to capture long-range dependencies through:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (38)$$

enables modeling of complex feature interactions in network traffic [1, 6, 8].

3. **Broad Misclassification Strategy**: SGAN causes systematic confusion across all cate- gories [1, 15] that completely undermines IDS reliability versus AIDAE's single-vector approach [2].

4. **IDS Confusion Loss**: Direct optimization against the target classifier:

$$L_{confuse} = \text{MSE}(RF(G(\mathbf{x}, \mathbf{z})), \mathbf{0}) \quad (39)$$

generates perturbations that exploit fundamental feature space weaknesses [1, 31] rather than category-specific vulnerabilities.

5 Conclusion and Future Work

In this paper, we address the challenge of evaluating adversarial attacks against machine learning- based intrusion detection Systems by establishing a uniform Framework for comparing SGAN- IDS and AIDAE under identical experimental conditions. Our evaluation utilizes the NSL- KDD dataset and Random Forest classifier to assess the adversarial effectiveness of both meth- ods. Experimental outcomes reveal that SGAN-IDS achieves substantially higher attacks ef- fectiveness, reducing IDS accuracy from 91.38% to 21.85% with approximately 78% fooling rate, demonstrating 86% superior performance compared to AIDAE's 40.50% accuracy drop. These results underscore the critical vulnerabilities in ML-based IDS and highlight the ro- bustness of SGAN's self-attention mechanism and balanced training approach versus AIDAE's concentrated normal-mimicry strategy and discriminator dominance.

Mathematically, we have shown that the adversarial optimization problem:

$$\mathbf{x}_{adv} = \arg\max_{\|\boldsymbol{\delta}\|_p \leq \varepsilon} \mathcal{L}(f(\mathbf{x} + \boldsymbol{\delta}), y) \quad (40)$$

can be effectively solved by both frameworks, with SGAN achieving superior results due to its ability to model complex feature dependencies.

In **future work**, we have significant interest in developing robust defense mechanisms against these sophisticated adversarial attacks. This includes:

1. **Adversarial Training**: Incorporating adversarial examples into the training set:

$$\min_{f} \mathrm{E}_{(\mathbf{x},y)\sim D} \max_{\|\boldsymbol{\delta}\|_p \le \varepsilon} L(f(\mathbf{x}+\boldsymbol{\delta}), y) \quad (41)$$

2. **Certified Robustness**: Developing provable defenses using randomized smoothing:

$$g(\mathbf{x}) = \arg\max_{c \in Y} \mathrm{P}_{\boldsymbol{\delta}\sim N(0,\sigma^2 I)}(f(\mathbf{x}+\boldsymbol{\delta}) = c) \quad (42)$$

3. **Ensemble-Based Detection**: Combining multiple IDS architectures to improve resilience against transfer attacks.
4. **Adversarial Detection**: Developing mechanisms to identify GAN-manipulated traffic using statistical fingerprinting and feature-space analysis.
5. **Transferability Analysis**: Examining the transferability of adversarial samples across different IDS architectures:

$$\text{Transfer Rate} = \mathrm{P}(f_{target}(\mathbf{x}_{adv}) \neq y \mid f_{source}(\mathbf{x}_{adv}) \neq y) \quad (43)$$

to inform universal defense strategies.

REFERENCES

[1] S. Aldhaheri and A. Alhuzali, "SGAN-IDS: Self-Attention-Based Generative Adversarial Network against Intrusion Detection Systems," *Sensors*, vol. 23, no. 18, p. 7796, Sep. 2023.

[2] J. Chen, D. Wu, Y. Zhao, N. Sharma, M. Blumenstein, and S. Yu, "Fooling intrusion de- tection systems using adversarially autoencoder," *Digital Communications and Networks*, 2020.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl. (CISDA)*, Ottawa, ON, Canada, Jul. 2009, pp. 1–6.

[4] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in *Proc. Conf. MilCIS*, Adelaide, SA, Australia, Nov. 2015, pp. 1–6.

[5] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, Funchal, Portugal, Jan. 2018, pp. 108–116.

[6] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversar- ial networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, Jun. 2019, pp. 12744–12753.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015.

[10] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack genera- tion against intrusion detection," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, Chengdu, China, May 2022, pp. 79–91.

[11] M. Usama, M. Asim, S. Latif, J. Qadir, and A. Al-Fuqaha, "Generative adversarial net- works for launching and thwarting adversarial attacks on network intrusion detection sys- tems," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Tangier, Morocco, Jun. 2019, pp. 78–83.

[12] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," *arXiv preprint arXiv:1702.05983*, Feb. 2017.

[13] P. T. Duy, L. K. Tien, N. H. Khoa, D. T. T. Hien, A. G. T. Nguyen, and V. H. Pham, "DIGFuPaS: Deceive IDS with GAN and function-preserving on adversarial samples in SDN-enabled networks," *Comput. Secur.*, vol. 109, p. 102367, Oct. 2021.
[14] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "SynGAN: Towards generating synthetic network attacks using GANs," *arXiv preprint arXiv:1908.09899*, Aug. 2019.
[15] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić , P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discov. Databases (ECML-PKDD)*, Prague, Czech Republic, Sep. 2013,
pp. 387–402.
[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial ex- amples," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015.
[17] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in
Proc. IEEE Symp. Secur. Privacy (SP), San Jose, CA, USA, May 2017, pp. 39–57.
[18] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
[19] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.
[20] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.
[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[23] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "A novel ensemble of hybrid intrusion detection system for detecting Internet of Things attacks," *Electronics*, vol. 8, no. 11, p. 1210, Oct. 2019.
[24] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Data-driven cyber security in perspective–intelligent traffic analysis," *IEEE Trans. Cybern.*, vol. 50, no. 7, pp. 3081– 3093, Jul. 2020.
[25] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Commun. Surv. Tutor.*, vol. 21, no. 1, pp. 686–728, First Quarter 2019.
[26] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Sydney, Australia, Aug. 2017, vol. 1, pp. 298–321.
[27] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine read- ing," in *Proc. Conf. Empir. Methods Natural Lang. Process. (EMNLP)*, Austin, TX, USA, Nov. 2016, pp. 551–561.
[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidi- rectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol. (NAACL-HLT)*, Minneapolis, MN, USA, Jun. 2019, vol. 1, pp. 4171–4186.
[29] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neu- ral machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process. (EMNLP)*, Lisbon, Portugal, Sep. 2015, pp. 1412–1421.
[30] A. Kantchelian, J. D. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers," in *Proc. Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, Jun. 2016,
pp. 2387–2396.
[31] F. Zhang, P. P. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 766–777, Mar. 2016.
[32] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 7794–7803.
[33] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversar- ial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, Oct. 2018.
[34] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection sys- tems: Taxonomy, solutions and open issues," *Inf. Sci.*, vol. 239, pp. 201–225, Aug. 2013.
[35] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (ASIACCS)*, Abu Dhabi, UAE, Apr. 2017, pp. 506–519.

[36] S. Revathi and A. Malathi, "A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection," *Int. J. Eng. Res. Technol.*, vol. 2, no. 12, Dec. 2013.
[37] Y. Tang, L. Gu, and L. Wang, "Deep Stacking Network for Intrusion Detection," *Sensors*, vol. 22, no. 1, p. 25, Dec. 2021.
[38] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 656–672