

Research Article

Enhanced Network Intrusion Detection through Conditional GAN-based Minority Class Augmentation

Rawaa M. Aljbouri¹, Esraa Muneam Abdulhussein², Zainab A. Abdulazeez^{3,*}

¹ Presidency of University of Babylon, University of Babylon, Babylon, Iraq

² Kufa Technical Institute, AL-Furat Al-Awsat Technical University, Al-Najaf, Iraq

³ College of Education for Human Sciences, University of Kerbala, Karbala, Iraq

ARTICLE INFO

Article History

Received 13 Jul 2026
Revised 22 Aug 2026
Accepted 19 Sep 2026
Published 23 Oct 2026

Keywords

Network Intrusion
Detection,
Generative Adversarial
Networks,
Class Imbalance,
Synthetic Data
Generation,
Cybersecurity.



ABSTRACT

Severe class imbalance makes it difficult for network intrusion detection systems to detect rare attack classes. We introduce Cyber-Synth-CGAN, a conditional Wasserstein GAN with gradient penalty for synthesizing minority attack patterns, combining signed log-transform scaling, spectral normalization, minority-focused training cycles and a sample-size-gated augmentation policy. The scaler is fitted on training rows only, and all synthetic data are generated from the training partition. On CIC-IDS-2017, recall on Web Attack: Brute Force improved from 68.77% to 98.34% in a representative run, and by 29.90 points across five seeds (95% CI of the change [27.80, 32.00], $p < 0.0001$), with precision rising from 0.6509 to 0.6965. Average precision rose more modestly (0.7919 to 0.8340), indicating that much of the recall gain reflects a shift of the decision boundary rather than improved ranking. The five other classes with more than 1,000 training samples were statistically unchanged, whereas Web Attack: XSS (522 samples) lost 51.69 points; SMOTE showed the opposite pattern. These results suggest, but do not establish, a sample-size boundary between 522 and 1,206 training samples on this dataset, which we report as a dataset-specific observation rather than a validated threshold.

1. INTRODUCTION

1.1. Motivation

Network intrusion detection systems (NIDS) are important protection mechanisms against changing cyber threats. Nonetheless, practical network traffic contains an extreme imbalance of classes with benign traffic and typical attacks prevalent in datasets and rare attacks grossly under-represented [1], [2]. This imbalance presents a basic dilemma: machine learning models trained on this type of data have high accuracy overall but are unable to identify rare but potentially disastrous attacks.

A detection system with 99% overall accuracy may fail to detect zero-day exploits, advanced persistent threats, or targeted attacks, which constitute less than 0.1% of network traffic. According to recent research, minority attack categories in benchmark datasets, such as CIC-IDS-2017, have as few as 11–36 samples [3], [4], which is barely enough to train an efficient model using traditional techniques.

Traditional methods to overcome class imbalance (e.g., SMOTE oversampling, cost-sensitive learning, and ensemble models) are less effective when the imbalance ratio is extreme (1:1000) [5]. These techniques either add excessive noise with mere replication or cannot reflect the multidimensional phenomenon of network traffic patterns.

1.2. Research Gap

Current developments in generative adversarial networks (GANs) have shown impressive performance in producing lifelike synthetic data in different fields [6], [7]. Nevertheless, they have specific difficulties in their application to network intrusion detection.

Scale disparity: Network traffic has many orders of magnitude (between bytes and billions), which must be pre-processed using specialized means.

*Corresponding author. Email: zainab.abdulhameed@uokerbala.edu.iq

High dimensionality: Currently, data on network flows consist of 70–80 or more features that have intricate interrelations.

Sample scarcity of minorities: Some attack classes have fewer than 50 samples from which to learn.

Difficulty in validation: Evaluating the quality of synthetic network traffic requires domain-specific quantitative metrics rather than visual inspection alone.

Several studies have examined GANs in the field of cybersecurity [8], [9], [10], but the following questions remain open:

1. How many actual samples can be used to perform effective GAN-based augmentations?
2. How can it be verified that synthetic network traffic retains valid attack attributes?
3. What are the preprocessing techniques with extreme ranges of network values?

More broadly, do conditional GANs have the potential to enhance the performance of specific minority attack types without worsening the performance of other types?

1.3. Contributions

This study makes several contributions to the literature.

Preprocessing for extreme value ranges: we use a signed log-transform followed by min–max scaling, fitted on the training partition only, to bring network-flow features spanning –32 billion to +2 billion into $[-1, 1]$ for stable GAN training.

Sample-size-gated augmentation: We introduce a conditional GAN framework with adaptive training, minority-oriented learning cycles, a sample-size gate that decides which classes are augmented, and diagnostic fidelity assessment (KS, MMD) of the synthetic data for network intrusion detection (NID).

Empirical observation of a sample-size boundary: on CIC-IDS-2017, generative augmentation degraded the one augmented class with fewer than 1,000 training samples (Web Attack: XSS, 522) and improved or left unchanged every class above that level. These observations bracket a boundary between 522 and 1,206 training samples but do not locate it precisely, and they rest on a single dataset; we therefore present the boundary as a hypothesis for further validation rather than a validated threshold.

Performance improvement under a leakage-controlled protocol: our method increased recall on Web Attack: Brute Force by 29.57 points in a representative run (from 68.77% to 98.34%; 296 of 301 test flows detected against 207 at baseline) and by a mean of 29.90 points across five independent seeds (95% confidence interval of the change in recall [27.80, 32.00]; Holm-corrected $p < 0.0001$), while precision on the same class improved from 0.6509 to 0.6965. The effect only affected this class; the five other classes with more than 1,000 training samples were statistically unchanged, while the one augmented class below this boundary was made worse.

Comparison with classical oversampling: on the same partition and classifier (in a separate run on a different training subsample; Section 4.9), generative augmentation improved the class with 1,206 training samples (Web Attack: Brute Force) and degraded the class with 522 (Web Attack: XSS), whereas SMOTE and random oversampling showed the opposite pattern. This suggests that the choice of augmentation method may depend on the number of real samples per class, a suggestion that rests on two classes and should be calibrated on each new dataset.

The remainder of this paper is structured as follows: Section 2 presents the literature review; Section 3 describes our methodology; Section 4 presents the experimental findings; Section 5 discusses the implications and limitations; and Section 6 concludes with future directions for research.

2. RELATED WORKS

2.1. Class Imbalance in Network Intrusion Detection

Class imbalance is a basic issue in machine learning when applied to cybersecurity problems. Zhang et al. [11] reviewed deep learning-based intrusion detection systems and identified class imbalance as one of the main limitations for successful practical implementation, and Abdelkhalek and Mashaly [2] addressed the problem in network intrusion detection with data resampling and deep learning. More generally, Haixiang et al. [12] reviewed 517 studies on learning from class-imbalanced data, covering data preprocessing, classification algorithms and model evaluation.

The current development of the CIC-IDS-2017 dataset demonstrates its level of difficulty. In this benchmark dataset, as reported by Wang et al. [13] and Milosevic and Ciric [14], the minority attack classes contain between 11 and 1,966 samples, in contrast to more than 2.2 million benign flow samples. Traditional machine learning models trained on this type of data usually have an overall accuracy of 99% or higher while failing to detect rare attack types [15].

2.2. Synthetic Data Generation for Cybersecurity

The use of generative models in cybersecurity has drawn substantial attention since 2020. Agrawal et al. [16] reviewed generative models, in particular GANs, for generating synthetic attack data in cybersecurity, and examined whether such data are realistic enough for models trained on them to detect real-world attacks.

Li et al. [1] investigated conditional GANs for producing labeled network intrusions and demonstrated that it was promising in terms of minority class augmentation but had problems with training stability and mode collapse in situations with extremely rare classes.

Recently, specialized architectures have been developed. Tian et al. [17] combined a variational autoencoder with a GAN to augment intrusion-detection data, and Alabsi et al. [18] applied a conditional tabular GAN to detect DoS and DDoS attacks in Internet of Things networks. Nevertheless, these methods failed to systematically investigate the relationship between the sample size of minority classes and the effectiveness of augmentation.

2.3. Advanced GAN Architectures

Since the initial formulation by Goodfellow et al. [6], GANs have evolved significantly. The Wasserstein GAN (WGAN) introduced by Arjovsky et al. [19] addressed training instability by replacing the Jensen–Shannon divergence with the Earth Mover distance, and Gulrajani et al. [20] replaced weight clipping with a gradient penalty (WGAN-GP), giving more stable training. These formulations have since been applied in other domains, for example to radar target classification [21] and sonar signal processing [22].

Conditional GANs (cGANs), introduced by Mirza and Osindero [23], condition generation on class labels supplied to both the generator and the discriminator. Later developments include spectral normalization of the discriminator, proposed by Miyato et al. [24], and adaptive discriminator augmentation, proposed by Karras et al. [7] to avoid discriminator overfitting on small datasets.

Although their use in network intrusion detection requires domain-related adjustments, such architectural innovations form the basis of this study.

2.4. Data Preprocessing and Feature Engineering

Deep learning on network traffic data presents distinct challenges during data preprocessing. Yang et al. [25] used an improved conditional variational autoencoder to generate samples of minority attack classes and trained a deep neural network detector on the augmented data. Andresini et al. [26] converted network flows into images and used GAN-generated samples to address class imbalance in a convolutional intrusion detector.

Kasongo and Sun [27] showed that the performance of intrusion detection models depends strongly on the input representation, using XGBoost-based feature selection on the UNSW-NB15 dataset. None of these studies addresses the preprocessing of features whose values span ten or more orders of magnitude.

2.5. Research Gaps

However, despite these achievements, several important gaps remain in the literature.

Sample size considerations: No literature has studied the minimum sample size for successfully using GAN-based augmentation for network intrusion detection.

Validation methodology: Current methods do not statistically validate the quality of synthetic network traffic using metrics other than downstream classification performance.

Practical recommendations: The literature does not offer much guidance on when augmentation based on GAN should be applied or which alternative methods should be used.

Extreme value preprocessing: Current approaches are insufficient to handle network characteristics that can have an extreme range of values (-32 billion to +2 billion).

Our study addresses these gaps through a systematic empirical analysis and domain-specific methodological choices.

3. MATERIALS AND METHODS

3.1. Framework Overview

We introduce the Cyber-Synth-CGAN framework to overcome the issue of minority-attack-pattern identification in high-dimensional network traffic data. The general workflow, as shown in Figure 1, comprises four interrelated steps that ensure the robust production of synthetic data.

1. **Data Ingestion and Adaptive Preprocessing:** The system accepts raw network flow data (e.g., CIC-IDS-2017) and applies a specialized cleaning and canonicalization pipeline to them. Crucially, a new technique of log-transform scaling is used to squash values of features within extreme orders of magnitude (from -32 billion to +2 billion) into a normalized range for training neural networks.

2. Sample Threshold Filtering: The processed data are subjected to a class-imbalance analysis, and the classes are divided according to the number of available training samples.

- Ultra-minority (<100 training samples): These are marked as “skip-augmentation” to avoid model collapse and are directly passed to the classifier.
- Augmentable classes (≥ 100 training samples and below the base target of 6,000 samples per class): these are passed to the generative process. The base target decides which classes are augmented; the final number of synthetic samples, which can exceed it, is set in Section 3.6.2.

3. Conditional GAN Augmentation: The selected classes are fed into the conditional Wasserstein GAN with gradient penalty (CWGAN-GP). The architecture consists of a generator that produces synthetic flow vectors conditioned on the attack labels and a discriminator that differentiates between real and synthetic traffic. The training had an adaptive learning rate and minority-focused cycles to ensure stability.

4. Quality Assessment and Classification: The generated samples are assessed statistically (KS test, MMD) as a diagnostic, and all generated samples are combined with the original real data to form the final augmented training set, which is then used to train the Random Forest classifier for intrusion detection.

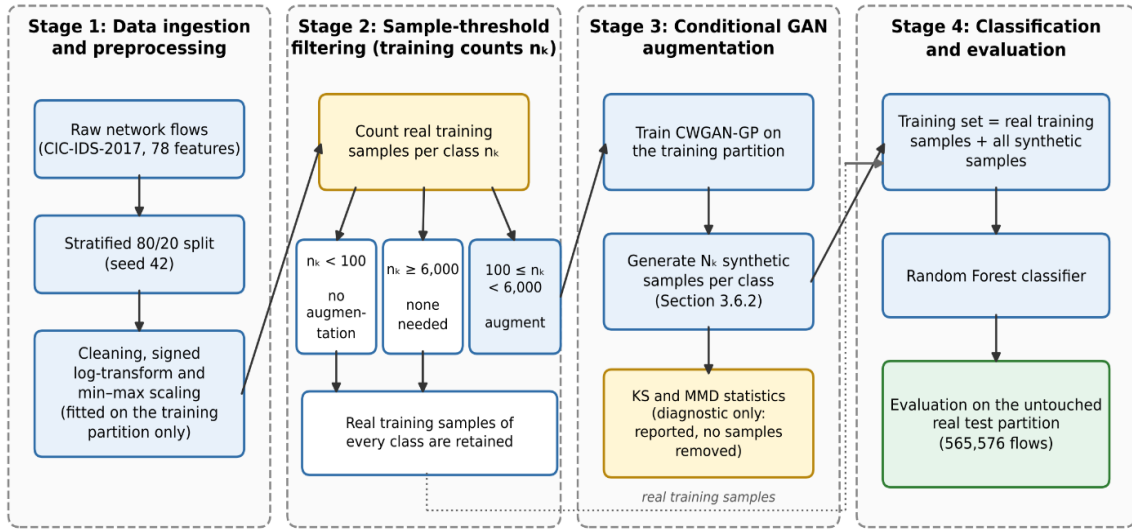


Fig. 1. System-Level Block Diagram

3.2. Problem Formulation

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ represent a network intrusion detection dataset with N samples, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes a d -dimensional network flow feature vector and $y_i \in \{1, 2, \dots, K\}$ is its class label (benign or a specific attack type). With $\mathcal{D}_k = \{(\mathbf{x}_i, y_i) : y_i = k\}$ denoting the samples of class k , the set of minority classes is defined as

$$M = \{k \in \{1, 2, \dots, K\} : |\mathcal{D}_k| < \tau N\} \quad (1)$$

where τ is a threshold (here $\tau = 0.001$). Our objective is to generate synthetic samples $\hat{\mathcal{D}}_k = \{(\hat{\mathbf{x}}_j, k)\}_{j=1}^{N_{\text{synth}}(k)}$ for each minority class $k \in M$ such that:

- 1- Distribution similarity: the class-conditional distribution of the synthetic samples approximates that of the real samples,

$$P(\hat{\mathbf{x}} | y = k) \approx P(\mathbf{x} | y = k) \quad (2)$$

- 2- Diversity: The generated samples exhibited sufficient variation to facilitate generalization.

If the traffic share of a class is below 0.1% ($\tau = 0.001$), the class is considered a minority class. This is not a random number: if the 15 classes are ranked according to the relative frequency, there is a gap in the distribution. Each of the six classes is less than 0.1% of the flows, with DoS SlowHTTPTest the next class above at 0.1943%, which is nearly three times the percentage of the previous class. The boundary is in that gap if $\tau = 0.001$.

The sensitivity of the resulting partition is presented in Table I. The set M is invariant for $\tau \in [0.00070, 0.00194]$, the interval over which the threshold count τN lies between the largest minority class (Bot, 1,966 flows) and the smallest non-minority class (DoS SlowHTTPTest, 5,499 flows). At $\tau = 0.0005$, Bot and Web Attack: Brute Force leave M ; at $\tau = 0.002$, DoS SlowHTTPTest enters M .

Two further points prevent a misreading. First, τ is a labelling threshold that identifies severely under-represented classes for analysis; it is distinct from the sample-size boundary discussed in Section 4.6, which is an empirical observation made after the experiments and was not used to decide which classes were augmented. Second, τ is not the criterion for augmentation.

TABLE I. SENSITIVITY OF MINORITY SET M TO τ

τ	Threshold count (τN)	$ M $	Classes in M
0.0005	1,415	4	XSS, Infiltration, SQL Injection, Heartbleed
0.00070	1,982	6	+ Bot, Brute Force
0.001	2,831	6	Bot, Brute Force, XSS, Infiltration, SQL Injection, Heartbleed
0.00194	5,492	6	(unchanged)
0.002	5,661	7	+ DoS SlowHTTPTest

3.3. Dataset: CIC-IDS-2017

We tested our methodology on the CIC-IDS-2017 dataset [28], which is widely used for evaluating network intrusion detection studies. The dataset contains 2,830,743 network flows collected over five days of realistic network traffic with contemporary attack scenarios.

Dataset characteristics.

- Features: 78 numerical features extracted using CICFlowMeter
- Classes: 15 classes (1 benign + 14 attack types)
- Imbalance ratio: Ranges from 1:0.000005 (Heartbleed) to 1:1 (majority class)
- Feature ranges: Values span from -32 billion to +2 billion

Minority class distribution (classes with $<0.1\%$ representation):

- Heartbleed: 11 samples (0.0004%)
- Web Attack: SQL Injection: 21 samples (0.0007%)
- Infiltration: 36 samples (0.0013%)
- Web Attack: XSS: 652 samples (0.0230%)
- Web Attack: Brute Force: 1,507 samples (0.0532%)
- Bot: 1,966 samples (0.0695%)

This imbalance is extreme, and CIC-IDS-2017 is an ideal testbed for examining minority-class augmentation methods.

3.4. Enhanced Preprocessing Pipeline

The data in network traffic are characterized by peculiarities in preprocessing because of the large range of extreme values, feature characteristics, and sparse attack representations that are specific to network traffic. We therefore designed a preprocessing method suited to GAN training.

3.4.1. Data Cleaning and Canonicalization

We implemented a three-stage data-cleaning process to ensure data integrity. First, we removed any numerical anomalies, where all infinite values were replaced with Not-a-Number (NaN) placeholders. Subsequently, rows with missing values were removed to preserve the integrity of the attack samples. Next, the class labels, the only non-numeric column, were encoded as integers. Finally, the attack labels were standardized to a consistent naming convention; for example, “Web Attack - XSS” was renamed “Web Attack: XSS”. All Heartbleed flows were assigned to a single class label, “Heartbleed”, so that each attack type corresponds to exactly one class.

3.4.2. Log-Transform Scaling

Standard min/max scaling often does not work with network traffic data because the features can have more than ten orders of magnitude. In such cases, extremely large values compress most of the feature data towards zero, where numerical instability may be encountered during the computation of the gradient, and GANs cannot learn meaningful representations. To solve this problem, we designed a log-transform scaling algorithm. The process starts with a type of forward transformation that applies a signed log transform to the extreme values to compress them logarithmically while preserving the sign information and zero values. This was followed by min-max scaling to normalize the data to fall within the range of $[-1, 1]$. Theoretically, this transformation guarantees bounded gradients to prevent gradient explosion, preserves the relative feature importance by log-space transformation, and is fully reversible, such that it can be used to generate realistic synthetic data. We empirically validated that the method works correctly by ensuring that the inverse transformation of the values returns to the original values with negligible errors.

With the network traffic data now rigorously cleaned and normalized into a stable numerical range, the data are ready for the generative model. The core of our framework is a conditional Wasserstein GAN with gradient penalty (CWGAN-GP), which learns the complicated distributions of these preprocessed minority classes. Unlike ordinary GANs, which may not be able to handle specific statistical features of network flows, our architecture is specifically designed to consider these normalized feature vectors and condition them according to the attack type. The following sections describe the specific parts of the Generator and Discriminator networks for this task.

3.4.3. Generator Network

The generator network is designed to map a noise vector and one-hot-encoded class label into a synthetic network flow vector. The architecture begins by incorporating the class label and appending it to noise inputs. This combined input passes through a series of four fully connected layers, which increase and decrease the dimensions to learn the hierarchical representations. We used layer normalization to stabilize training and address internal covariate shifts in the network. Finally, Tanh activation functions were used for the last layer to ensure that the output was in the range of $[-1, 1]$ owing to pre-processed data. The label embedding mechanism is important because it allows the network to learn different class-specific feature sets.

3.4.4. Discriminator Network

The discriminator differentiates between real and synthetic samples, which are conditioned according to their class labels. Its architecture has a flow vector and class label as the inputs. The label was embedded and concatenated with the flow features before being passed through a three-layered feature extraction network. Each linear layer was enforced with spectral normalization for the Lipschitz constraint, which is an important constraint for the stability of the Wasserstein GAN. It uses the LeakyReLU activation function to avoid dead gradients. A dropout was applied to the hidden layers to avoid overfitting. This design guarantees that the discriminator can critique the realism of the generated samples while maintaining training stability.

TABLE II. ARCHITECTURE OF CYBER-SYNTH-CGAN GENERATOR AND DISCRIMINATOR

Component	Layer	Units	Activation / Normalisation	Notes
Generator	Input	100 (noise) + label dim	-	Concatenation
	FC1 + LayerNorm	256	ReLU	Xavier init
	FC2 + LayerNorm	512	ReLU	
	FC3 + LayerNorm	1024	ReLU	
	FC4 + LayerNorm	512	ReLU	
	Output	78	Tanh	Matches scaled feature space
Discriminator	Input	78 + label dim	-	Concatenation
	SpectralNorm FC1	512	LeakyReLU(0.2) + Dropout(0.3)	
	SpectralNorm FC2	256	LeakyReLU(0.2) + Dropout(0.3)	Feature-matching layer
	SpectralNorm FC3	128	LeakyReLU(0.2)	
	Output	1	Linear	Critic score
Optimisers	Adam	$lr_G = 2 \times 10^{-4}$ $lr_D = 1 \times 10^{-4}$	$\beta_1=0.5, \beta_2=0.999$	Implementation defaults; $\lambda_{base} = 10$ (Eq. (6)). Values used in the reported runs: Section 4.1

3.4.5. Network Architecture Visualization

To help the reader better understand the generative process, Figure 2 shows the internal detailed structure of the proposed Conditional Wasserstein GAN with Gradient Penalty (CWGAN-GP).

In Figure 2, the generator (left) takes a 100-dimensional noise vector (z) and a class label (y) as input. These were concatenated and fed through four dense layers (256, 512, 1,024 and 512 units) using layer normalization and rectified linear unit (ReLU) activation functions, resulting in a hyperbolic tangent-activated output that generated a 78-dimensional synthetic flow vector. The discriminator (right) receives both real and synthetic flow vectors with the corresponding labels. It processes these through three dense layers (512, 256 and 128 units) with spectral normalization and LeakyReLU (0.2) activations, with dropout (0.3) after the first two layers, to output a validity score (critic score), which is used to instruct the learning process of the generator. The gradient penalty is computed by interpolating between the real and synthetic samples to impose the Lipschitz constraint.

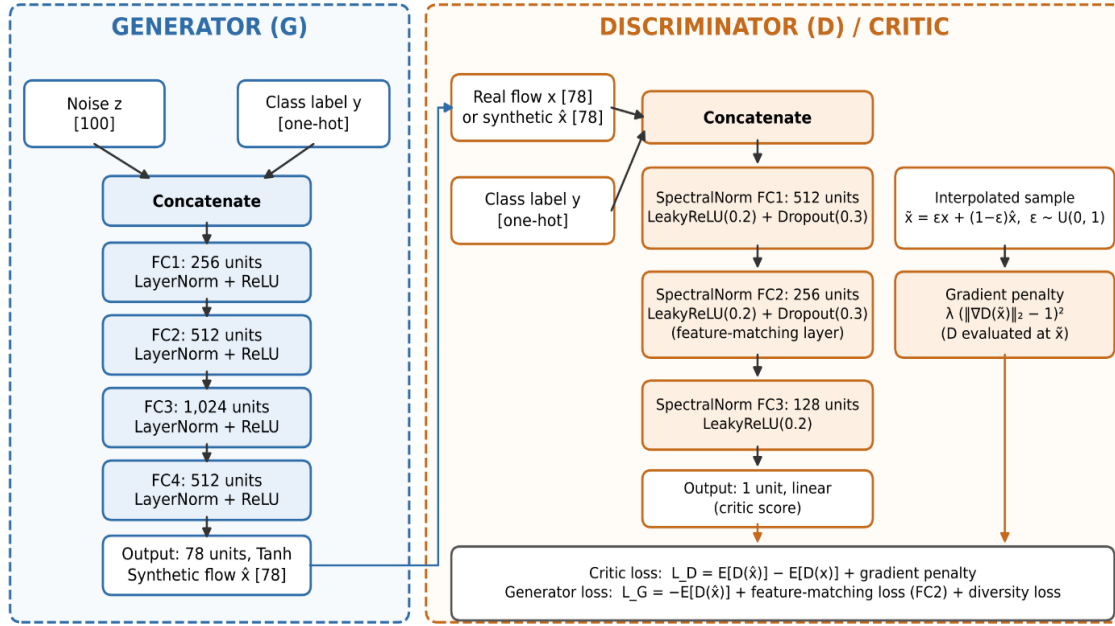


Fig. 2. Detailed Architecture of Cyber-Synth-CGAN

3.5. Advanced Training Strategy

Our training regimen includes several advanced techniques to ensure convergence and quality. We used a multi-component loss function (Section 3.5.1) that consists of a Wasserstein loss with a gradient penalty, a feature-matching loss that drives the generator to match the statistics of real data, and a diversity loss to penalize mode collapse. To overcome the class imbalance, we used a Minority-Focused Training phase. Every 10 epochs, the model goes through specialized training cycles with batches highly biased to minority classes (70% minority composition), with a higher weight given to the feature matching loss. In addition, we used an Adaptive Learning Rate that dynamically changed according to the ratio of the discriminator/generator loss to maintain a healthy adversarial competition. An Early Stopping mechanism was used to monitor the model for divergence or stagnation, and if no improvement was noticed for 1,000 epochs, the best-performing weights were restored.

3.5.1. Loss Functions

The discriminator (critic) is trained with the WGAN-GP objective [20]:

$$L_D = \mathbb{E}_{\hat{\mathbf{x}} \sim P_g} [D(\hat{\mathbf{x}}, c)] - \mathbb{E}_{\mathbf{x} \sim P_r} [D(\mathbf{x}, c)] + \lambda_{gp} GP \quad (3)$$

where P_r and P_g are the distributions of real and generated flows, $\hat{\mathbf{x}} = G(\mathbf{z}, c)$ is a generated sample obtained from the noise vector $\mathbf{z}$, c is the class label used for conditioning, and GP is the gradient penalty

$$GP = \mathbb{E}_{\tilde{\mathbf{x}}} \left[\left(\|\nabla_{\tilde{\mathbf{x}}} D(\tilde{\mathbf{x}}, c)\|_2 - 1 \right)^2 \right] \quad (4)$$

evaluated at interpolated samples

$$\tilde{\mathbf{x}} = \epsilon \mathbf{x} + (1 - \epsilon) \hat{\mathbf{x}}, \quad \epsilon \sim U(0,1) \quad (5)$$

Adaptive gradient penalty: the penalty strength depends on the epoch t as

$$\lambda_{gp}(t) = \lambda_{base} \left(1 - 0.3 \frac{t}{T}\right) \quad (6)$$

where T is the total number of epochs, so that the penalty strength decreases from its full value to 70% of it over training, allowing the discriminator more flexibility as training progresses.

The generator minimizes the multi-component objective

$$L_G = -\mathbb{E}_{\hat{\mathbf{x}} \sim p_g} [D(\hat{\mathbf{x}}, c)] + \lambda_{fm} L_{FM} + \lambda_{div} L_{div} \quad (7)$$

where λ_{fm} and λ_{div} weight the feature-matching and diversity losses. The feature-matching loss

$$L_{FM} = \left\| \mathbb{E}_{\mathbf{x} \sim p_r} [\phi(\mathbf{x})] - \mathbb{E}_{\hat{\mathbf{x}} \sim p_g} [\phi(\hat{\mathbf{x}})] \right\|_2^2 \quad (8)$$

encourages the generator to match the statistics of the real data, where $\phi(\cdot)$ denotes the intermediate discriminator features (the output of the second discriminator layer, Table II). The diversity loss

$$L_{div} = \mathbb{E}_{i \neq j} \left[\exp \left(-d(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) \right) \right] \quad (9)$$

penalizes mode collapse by encouraging sample diversity, where $d(\cdot, \cdot)$ denotes the Euclidean distance.

3.5.2. Minority-Focused Training

Every 10 epochs, a special training stage is run for the minority classes:

Minority Training Phase (three repetitions of this phase):

1. Sample class labels with a strong bias towards minority classes (cubed inverse frequency):

$$p(c_k) \propto \left(\frac{1}{n_k}\right)^3 \quad (10)$$

where n_k is the number of real training samples of class k .

2. Create minority-comprising batches (70% minority, 30% majority)
3. Compute the feature-matching loss on real minority samples:

$$L_{FM}^{\min} = \left\| \mathbb{E}[\phi(\mathbf{x}_{\min})] - \mathbb{E}[\phi(G(\mathbf{z}, c_{\min}))] \right\|_2^2 \quad (11)$$

where $\mathbf{x}_{\min}$ are real minority-class samples and $G(\mathbf{z}, c_{\min})$ are generated samples conditioned on the same classes.

4. Update the generator with a larger feature-matching weight ($\lambda_{fm} = 0.5$) than in the standard updates.

This specialized training prevents the neglect of minority classes in normal adversarial training.

3.5.3. Adaptive Learning Rate Adjustment

We also decreased and increased the learning rates in terms of the discriminator-generator balance as follows:

Every 100 epochs:

$$\rho = \frac{L_D}{L_G + \delta} \quad (12)$$

where δ is a small constant that prevents division by zero.

If $\rho > 3.0$ (discriminator too strong):

$$\text{lr}_D \leftarrow 0.9 \text{lr}_D, \quad \text{lr}_G \leftarrow 1.05 \text{lr}_G \quad (13)$$

If $\rho < 0.5$ (generator too strong):

$$\text{lr}_D \leftarrow 1.05 \text{lr}_D, \quad \text{lr}_G \leftarrow 0.95 \text{lr}_G \quad (14)$$

This maintains a balanced adversarial competition throughout training.

3.5.4. Early Stopping with Divergence Detection

Monitor from epoch 500:

Divergence check:

If $L_D > 100$ or $L_G > 50$:

STOP - training diverged

Convergence check:

Track best generator loss

If no improvement occurs for 1,000 epochs:

STOP - restore best model

Safety check:

Maximum training budget: 2,000 epochs. In the reported seed-42 run, early stopping was triggered at epoch 1,531 (no improvement in generator loss for 1,000 epochs), and the weights from approximately epoch 531, where the best generator loss was recorded, were restored.

3.6. Hybrid Augmentation Strategy

We use a decision-based augmentation strategy that determines how each class is handled according to its number of real training samples. Classes with fewer than 100 training samples are skipped to avoid unstable generator training, and classes with 100 or more training samples that fall below the base target of 6,000 samples are augmented with the conditional GAN. For each augmented class, the number of synthetic samples is set by the planning rule of Section 3.6.2 (Eqs. (15)–(18)), which starts from the base target and then applies the boost multipliers and their caps. After generation, the synthetic data of each class are assessed with the Kolmogorov–Smirnov (KS) and Maximum Mean Discrepancy (MMD) statistics. This assessment is diagnostic: in the experiments reported here no synthetic samples were discarded or reduced on the basis of it (see Section 3.6.3). The implementation also contains a SMOTE branch for classes with fewer than 50 samples, but because every such class falls below the 100-sample floor and is skipped first, this branch was never executed in the reported experiments.

3.6.1. Sample Threshold Policy

- The policy applied in all reported experiments has two tiers, evaluated on the number of real samples of class k in the training partition, $n_k = |\mathcal{D}_k^{\text{train}}|$:
- If $n_k < 100$: skip augmentation (insufficient samples for reliable GAN training).
- If $n_k \geq 100$: augment with the conditional GAN, using the sample counts of Section 3.6.2.

The results in Section 4.6 suggest that a further tier near 1,000 training samples, below which generative augmentation should be avoided or replaced by interpolation-based oversampling, may be beneficial. This additional tier was not part of the pipeline evaluated here; we present it as a recommendation derived from the results rather than as a component that was tested.

3.6.2. Augmentation Planning

For the classes approved for augmentation, we determined the synthetic sample counts as follows:

Base target per class:

$$T_{\text{base}} = \max(0.10 \cdot n_{\text{maj}}, 200) \quad (15)$$

Base allocation of synthetic samples:

$$N_{\text{synth}}(k) = \min(T_{\text{base}} - n_k, C_{\text{max}}) \quad (16)$$

where $C_{\text{max}} = 15,000$ is a per-class cap that prevents computational overload.

Boost multipliers, applied after the base allocation:

If $n_k < 0.0005 N_{\text{train}}$ (0.05% of the capped training partition):

$$N_{\text{synth}}(k) \leftarrow \min(3.0 \cdot N_{\text{synth}}(k), 2 C_{\text{max}}) \quad (17)$$

If $k \in M$ (minority set):

$$N_{\text{synth}}(k) \leftarrow \min(2.0 \cdot N_{\text{synth}}(k), 1.5 C_{\text{max}}) \quad (18)$$

Here n_{maj} is the majority-class count in the training partition after the 60,000-per-class cap on real training data, and N_{train} is the size of the capped training partition; hence $T_{\text{base}} = 6,000$. In the reported experiments the $\times 3$ boost for ultra-rare classes was not triggered for any augmented class, and the cap C_{max} was never reached. The synthetic counts in Table III

follow directly from Eqs. (15)–(18): classes outside M receive $6,000 - n_k$ samples (SSH-Patator 1,283; DoS slowloris 1,363; DoS SlowHTTPTest 1,601), and classes in M receive twice that amount (Bot 8,870; Web Attack: Brute Force 9,588; Web Attack: XSS 10,956), for a total of 33,661. FTP-Patator (6,350 training samples) exceeds the base target and receives none. The value of 6,000 is therefore a base target, used to decide whether a class receives synthetic samples and to compute its initial allocation; it is not an upper limit on the final number of samples per class. Because the boost multipliers are applied after the base allocation, a boosted class can exceed 6,000 samples: with the $\times 2$ multiplier, the training sets contain $1,565 + 8,870 = 10,435$ samples for Bot, $1,206 + 9,588 = 10,794$ for Web Attack: Brute Force and $522 + 10,956 = 11,478$ for Web Attack: XSS. The synthetic allocation is bounded by $C_{\max} = 15,000$ for the base allocation, $1.5 \cdot C_{\max} = 22,500$ after the $\times 2$ multiplier and $2 \cdot C_{\max} = 30,000$ after the $\times 3$ multiplier; none of these bounds was reached.

3.6.3. Quality Validation

For each augmented class, we computed the following statistics as a diagnostic of distributional fidelity:

Kolmogorov–Smirnov (KS) test per feature j :

$$D_{KS}^{(j)} = \sup_x \left| F_{\text{real}}^{(j)}(x) - F_{\text{synth}}^{(j)}(x) \right| \quad (19)$$

where $F^{(j)}$ denotes the empirical cumulative distribution function of feature j . A class is flagged as failing if $p_{KS}^{(j)} < 0.05$ for most features.

Maximum Mean Discrepancy (MMD):

$$\text{MMD}^2 = \mathbb{E}[\kappa(\mathbf{x}, \mathbf{x}')] + \mathbb{E}[\kappa(\hat{\mathbf{x}}, \hat{\mathbf{x}}')] - 2 \mathbb{E}[\kappa(\mathbf{x}, \hat{\mathbf{x}})] \quad (20)$$

where $\mathbf{x}, \mathbf{x}'$ are real samples and $\hat{\mathbf{x}}, \hat{\mathbf{x}}'$ are synthetic samples of the same class, and $\kappa(\cdot, \cdot)$ is a Gaussian kernel with median-heuristic bandwidth, computed on the normalized feature space. A class is flagged as failing if $\text{MMD}^2 > 0.1$.

Statistical moment matching (per feature):

$$|\mu_{\text{real}} - \mu_{\text{synth}}| < 0.5 \sigma_{\text{real}} \quad (21)$$

$$|\sigma_{\text{real}} - \sigma_{\text{synth}}| < 0.3 \sigma_{\text{real}} \quad (22)$$

These statistics were used for assessment and reporting only, not as a filter. In the reported experiments all six augmented classes were flagged as failing both tests (Table VII), and all of their synthetic samples were retained. Applying the criteria as a filter would have removed every synthetic sample and reduced the pipeline to the baseline. We retained the samples because the quantity of interest is classification performance on the untouched real test partition, which is measured directly in Section 4.5; the fidelity statistics are reported so that readers can judge how far the synthetic data depart from the real distributions.

After developing the architectural parts and strategic rules for selective augmentation, it is imperative to empirically verify the effectiveness of the framework in realistic intrusion detection scenarios. We must determine not only whether the model can produce data that are statistically similar, but also whether the data it produces will improve detection rates without increasing false alarms. To this end, the following section covers the design of our experiment, which specifies the particular training scenarios, strict baselines, and performance criteria that were used to quantify the effect of our synthetic data on minority class recall.

3.7. Evaluation Methodology

3.7.1. Experimental Design

We compared two training scenarios, both of which were evaluated on the same untouched real test partition:

- 1- Baseline: Classifier trained on the original imbalanced training partition.
- 2- Augmented: classifier trained on the training partition plus the synthetic minority samples.

In both scenarios, no synthetic samples entered the test partition.

3.7.2. Classifier Configuration

We employed a Random Forest as the downstream classifier for consistent evaluation.

RandomForestClassifier:

n_estimators: 200

max_depth: 20

```

min_samples_split: 5
min_samples_leaf: 2
max_features: 'sqrt'
class_weight: 'balanced_subsample'
bootstrap: True
max_samples: 0.7

```

Random Forest provides:

- Resistance to overfitting
- Built-in feature importance
- Robustness to class imbalance
- Non-linear decision boundaries

3.7.3. Performance Metrics

Primary metrics:

- Recall per class: Essential for minority-attack detection.
- Precision per class: the proportion of flows labelled as a class that truly belong to it, which reflects false alarms.
- F1-score: the harmonic mean of precision and recall.

When precision and recall conflict, we give more weight to recall on minority classes, because missing a rare attack is usually more costly than a false alarm.

4. EXPERIMENTAL RESULTS

Statistical variation. All results presented in Tables III, IV, V and VII and in Figures 3, 4 and 5 come from a single run with a fixed seed (seed = 42) and are therefore consistent with each other for a given class. Table VIII combines this run with an auxiliary run of the classical baselines on the same partition and seed but on a different training subsample, as explained beneath that table. Since GAN training is stochastic, we replicated the entire pipeline five times, using the same fixed partition but different seeds. The resulting distribution is reported in Section 4.5.3 and Table VI; the representative run is within the resulting distribution for all classes.

All experiments had a strict train/test separation. In the first step, the CIC-IDS-2017 data were split into an 80% training partition and a 20% held-out test partition. Cyber-Synth-CGAN is trained only on the training split, and synthetic samples are generated only based on the trained generator and class distribution of the training set. All classification metrics (confusion matrix statistics and AUC) were evaluated on the entire test set, which was not used for training the GANs or for creating the augmented training set.

The final experiments used the hyper-parameters listed in Table II. The implementation specifies defaults of generator learning rate 2×10^{-4} , discriminator learning rate 1×10^{-4} , base gradient-penalty weight $\lambda_{\text{base}} = 10$ (Eq. (6)), $n_{\text{critic}} = 5$ discriminator updates per generator update and feature-matching weight $\lambda_{\text{fm}} = 0.1$. When CPU-only execution is detected, these are automatically adjusted for stability, and all experiments reported here used the adjusted values: generator learning rate 1.5×10^{-4} , discriminator learning rate 8×10^{-5} , $\lambda_{\text{base}} = 5$, $n_{\text{critic}} = 3$ and $\lambda_{\text{fm}} = 0.05$. Unchanged: Adam $\beta_1 = 0.5$, $\beta_2 = 0.999$ and diversity weight $\lambda_{\text{div}} = 0.01$. The batch size and noise dimension are listed in Section 4.1.

4.1. Experimental Setup: Hardware, Software and Hyper-parameters

Hardware: All experiments were conducted on a CPU (Intel Core i7) to demonstrate accessibility without specialized GPU resource requirements.

Hyperparameters (CPU-adjusted, as used in all the experiments):

- Epochs: 2,000 (with early stopping)
- Batch size: 128

- Noise dimension: 100
- Generator learning rate: 0.00015
- Discriminator learning rate: 0.00008
- Base gradient-penalty weight λ_{base} : 5.0
- Feature-matching weight λ_{fm} : 0.05
- n_{critic} : 3 (discriminator updates per generator update)

Training duration: early stopping at epoch 1,531 of the 2,000-epoch budget, with the best weights (from approximately epoch 531) restored; the complete pipeline, including training, generation and evaluation, took 47.6 minutes on a standard CPU.

Data split: 80% training and 20% testing with stratified sampling. The real data were capped at 60,000 samples per class to ensure computational feasibility.

4.2. Preprocessing Effectiveness

Our log-transform scaling successfully handled the extreme value ranges as follows.

Before transformation:

- Range: [-32,212,234,632 to 2,070,000,000]
- Max value: 2.07×10^9

After log transformation:

- Range: [-24.20 to 21.45]
- Compressed by factor of $\sim 10^9$

After min-max scaling:

- Range: [-1.000000, 1.000000]

The original sample max and inverse transform max had negligible round-trip errors, confirming the validity of the transformation. With this preprocessing, GAN training on features spanning this dynamic range was stable in all reported runs. We did not run the pipeline without the transform, so we do not report its effect quantitatively.

4.3. Training Dynamics

Convergence behavior:

- Training stopped at epoch 1,531 of the 2,000-epoch budget (early stopping triggered); the best weights, from approximately epoch 531, were restored
- Generator loss: $0.06 \rightarrow 10.63$
- Discriminator loss: $4.13 \rightarrow -12.11$
- Gradient penalty: $4.18 \rightarrow 0.15$

Adaptive learning rate adjustment:

- The mechanism intervened between one and three times per run, typically within the first 400 epochs, and in response to the discriminator becoming too strong, with loss ratios between 3.0 and 15.7.
- One seed showed the opposite condition (generator too strong) at epoch 100.
- No further intervention occurred after the early phase, and the divergence guard was never triggered.

Minority-focused training

- Minority-focused cycles were conducted every 10 epochs during the training.

4.4. Synthetic Data Generation and Threshold Filtering

Table III summarizes how the augmentation strategy was executed. Six classes met the 100-sample floor and were augmented with the GAN, producing a total of 33,661 synthetic flows according to the planning rule of Section 3.6.2. Three ultra-rare classes fell below the floor and were not augmented: Infiltration (29 training samples), Web Attack: SQL Injection (17) and Heartbleed (9). Synthetic data are therefore generated only for classes with enough real training samples to train the generator.

TABLE III. SAMPLE THRESHOLD FILTERING AND SYNTHETIC DATA GENERATION RESULTS

Class	Full class	Training partition	Decision	Synthetic	Method
SSH-Patator	5,897	4,717	Augment	1,283	GAN
DoS slowloris	5,796	4,637	Augment	1,363	GAN
DoS SlowHTTPTest	5,499	4,399	Augment	1,601	GAN
Bot	1,956	1,565	Augment	8,870	GAN
Web Attack: Brute Force	1,507	1,206	Augment	9,588	GAN
Web Attack: XSS	652	522	Augment	10,956	GAN
Infiltration	36	29	Skip (< 100)	0	—
Web Attack: SQL Injection	21	17	Skip (< 100)	0	—
Heartbleed	11	9	Skip (< 100)	0	—
Total				33,661	

Every training-partition count corresponds to the 80% stratified training share of the full class count (to within rounding for the smallest classes), consistent with generator training on the training partition only. The Bot class contained 1,966 raw flows, of which 1,956 remained after the removal of rows with infinite or missing values.

4.5. Minority Class Detection Performance

4.5.1. Baseline Performance (Real Only)

To set up a baseline, we first trained a random forest classifier using only the original imbalanced training data. Table IV provides the details of the classification of minority classes under this condition.

TABLE IV. PER-CLASS PERFORMANCE ON THE HELD-OUT TEST PARTITION: BASELINE VERSUS AUGMENTED (SEED 42, 565,576 TEST FLOWS).

Class	Support	Base P	Base R	Base F1	Aug P	Aug R	Aug F1
BENIGN	454,265	0.9999	0.9934	0.9966	0.9999	0.9967	0.9983
DoS Hulk	46,025	0.9894	0.9993	0.9943	0.9859	0.9994	0.9926
PortScan	31,761	0.9936	0.9996	0.9966	0.9937	0.9996	0.9966
DDoS	25,605	0.9987	0.9991	0.9989	0.9988	0.9990	0.9989
DoS GoldenEye	2,059	0.9804	0.9971	0.9887	0.9790	0.9971	0.9880
FTP-Patator	1,587	0.9987	0.9994	0.9991	0.9975	0.9987	0.9981
SSH-Patator	1,180	0.9992	1.0000	0.9996	0.9983	1.0000	0.9992
DoS slowloris	1,159	0.9957	0.9974	0.9966	0.9931	0.9965	0.9948
DoS SlowHTTPTest	1,100	0.9246	0.9918	0.9570	0.9191	0.9918	0.9541
Bot	391	0.1655	1.0000	0.2841	0.4436	0.9949	0.6136
Web Attack: Brute Force	301	0.6509	0.6877	0.6688	0.6965	0.9834	0.8154
Web Attack: XSS	130	0.2420	0.5846	0.3423	0.4615	0.0462	0.0839
Infiltration	7	1.0000	0.5714	0.7273	1.0000	0.5714	0.7273
Web Attack: SQL Injection	4	1.0000	0.5000	0.6667	0.6667	0.5000	0.5714
Heartbleed	2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Accuracy	565,576		0.9943			0.9970	
Macro avg	565,576	0.8626	0.8881	0.8411	0.8756	0.8717	0.8488

4.5.2. Detailed Performance Analysis

Table V compares baseline and augmented recall. Web Attack: Brute Force shows the largest improvement, from 68.77% to 98.34%, a gain of 29.57 points, reducing missed detections from 94 to 5 of 301 test flows. The precision on the same class rose from 0.6509 to 0.6965; therefore, the improvement was not obtained by trading precision for recall.

A second and unanticipated benefit appears in the Bot class. Its baseline precision of 0.1655 at 100.00% recall indicates a classifier flagging a large number of non-Bot flows to capture all 391 true instances. After augmentation, the precision rises to 0.4436 with a recall of 99.49%, a substantial reduction in false alarms on this class. Bot was one of the augmented classes (8,870 synthetic samples, Table III); because its baseline recall was already 100.00%, the effect of augmentation appears in its precision rather than its recall.

TABLE V. EFFECT OF AUGMENTATION ON RECALL, ORDERED BY TRAINING-PARTITION SIZE (SEED = 42).

Attack Class	Full class	Training partition	Baseline recall	Augmented recall	Δ (pp)	Status
FTP-Patator†	7,938	6,350	99.94%	99.87%	-0.07	Maintained
SSH-Patator	5,897	4,717	100.00%	100.00%	0.00	Maintained
DoS slowloris	5,796	4,637	99.74%	99.65%	-0.09	Maintained
DoS SlowHTTPTest	5,499	4,399	99.18%	99.18%	0.00	Maintained
Bot	1,956	1,565	100.00%	99.49%	-0.51	Maintained
Web Attack: Brute Force	1,507	1,206	68.77%	98.34%	+29.57	Improved
Web Attack: XSS	652	522	58.46%	4.62%	-53.84	Degraded

† FTP-Patator was not augmented: its 6,350 training samples exceed the base target of 6,000, so it received no synthetic samples (Table III; Section 3.6.2). It is included in Table V as a non-augmented reference class; its small change in recall arises only because the classifier was retrained on a training set that contains synthetic samples of the other classes.

The mean change in recall across the seven classes was -3.56 percentage points, dominated entirely by the XSS degradation. Restricted to the six classes with more than one thousand training samples, the mean change was +4.82 points.

4.5.3. Run-to-Run Stability and Statistical Significance

GAN training is stochastic, and performing one GAN run does not support a performance claim. We thus repeated this entire pipeline five times with different random seeds, while keeping the same train/test split, so that all runs were evaluated on the same test flows. The mean and standard deviation of recall with 95% confidence intervals are reported for each of the seven classes along with paired two-sided t-tests and Holm-Bonferroni correction.

Two effects remain significant after correction. Recall on Web Attack: Brute Force increases by 29.90 points (95% CI [27.80, 32.00], $p < 0.0001$), and recall on Web Attack: XSS falls by 51.69 points (95% CI [-54.51, -48.88], $p < 0.0001$). The corrected p-values for the remaining five classes range from 0.0928 to 1.0000, and their absolute effects are 0.66 points or less. Only one class shows a large positive effect; the other classes above the sample-size boundary are flat.

The standard deviations are informative in their own right. Both effects were observed for every seed: standard deviations of 0.72 and 1.03 points were found for augmented Brute Force and XSS recall, respectively. The XSS degradation is therefore not run-to-run instability; it is systematic and reproducible across seeds. In addition, augmentation reduced the run-to-run standard deviation of Brute Force recall from 1.87 points at baseline to 0.72 points.

TABLE VI. PER-CLASS RECALL ACROSS FIVE INDEPENDENT SEEDS (MEAN $\pm$ SD), WITH PAIRED T-TESTS AFTER HOLM-BONFERRONI CORRECTION. THE 95% CIs ON Δ ARE UNADJUSTED PER-CLASS T-INTERVALS (4 DEGREES OF FREEDOM); THE SIGNIFICANCE COLUMN USES THE HOLM-CORRECTED P-VALUES ACROSS THE SEVEN CLASSES, WHICH IS WHY AN INTERVAL EXCLUDING ZERO (BOT) CAN STILL BE NON-SIGNIFICANT

Class	Support	Baseline recall (%)	Augmented recall (%)	Δ (pp)	95% CI	p (corr.)	Sig.
Web Attack: XSS	130	57.38 $\pm$ 1.69	5.69 $\pm$ 1.03	-51.69	[-54.51, -48.88]	< 0.0001	Yes
Web Attack: Brute Force	301	68.17 $\pm$ 1.87	98.07 $\pm$ 0.72	+29.90	[+27.80, +32.00]	< 0.0001	Yes
Bot	391	99.03 $\pm$ 0.80	98.36 $\pm$ 0.97	-0.66	[-1.15, -0.18]	0.0928	No
DoS SlowHTTPTest	1,100	99.15 $\pm$ 0.26	99.16 $\pm$ 0.23	+0.02	[-0.03, +0.07]	0.7478	No
DoS slowloris	1,159	99.26 $\pm$ 0.44	99.26 $\pm$ 0.40	0.00	[-0.08, +0.08]	1.0000	No
SSH-Patator	1,180	99.88 $\pm$ 0.08	99.85 $\pm$ 0.09	-0.03	[-0.09, +0.02]	0.7112	No
FTP-Patator	1,587	99.96 $\pm$ 0.03	99.94 $\pm$ 0.04	-0.03	[-0.07, +0.02]	0.7112	No

4.6. Sample Size Threshold Analysis

We observed an association between the number of real training samples in a class and the effect of augmentation on that class (Figure 3).

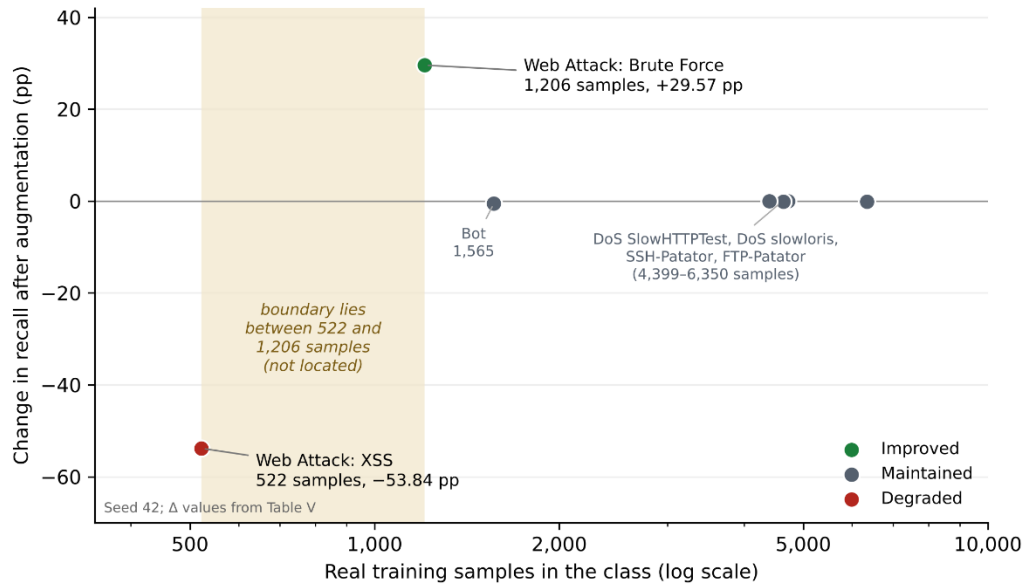


Fig. 3. Augmentation effectiveness versus sample size

Key findings:

- Among the six classes with more than 1,000 real training samples, augmentation either improved recall substantially (Web Attack: Brute Force, +29.57 points in the representative run and +29.90 points across seeds) or left it statistically unchanged. No class in this range was harmed.
- Between 1,565 and 6,350 training samples, none of the five classes (Bot, DoS SlowHTTPTest, DoS slowloris, SSH-Patator and FTP-Patator) showed a statistically significant change after correction (FTP-Patator received no synthetic samples; see the note to Table V); the changes ranged from -0.51 to 0.00 points.
- The one augmented class with fewer than 1,000 real training samples was harmed. Web Attack: XSS (522 training samples), which received the largest synthetic allocation of any class (10,956 samples), lost 53.84 points of recall in the representative run and 51.69 points across seeds. Its precision rose to 0.4615 and its false positive rate fell to 0.000012, indicating that the augmented model predicted the class far less often.
- Below 100 real training samples, no augmentation was attempted, and the test supports (2, 4 and 7 flows) are too small for the reported recall to be informative.
- The comparison in Section 4.9 shows that classical interpolation and generative augmentation succeed in opposite regimes: SMOTE improves recall on the class below the boundary by 8.46 points and reduces it on the class above by 6.65 points, while Cyber-Synth-CGAN does the reverse.

These findings should be read with three limitations. First, the boundary rests on two informative classes: one class below it degraded and one class above it improved substantially, so the data bracket the boundary between 522 and 1,206 training samples without locating it; the figure of approximately 1,000 is an interpolation, not a measured value. Second, the two classes differ in attack type as well as in sample size, so the contribution of sample size cannot be separated from class-specific factors. Third, all results come from a single dataset. We therefore report the boundary as a hypothesis that a sample-size gate should be calibrated on each new dataset, not as a validated threshold. Under the implemented pipeline, augmentation decisions were made by the 100-sample rule of Section 3.6.1 only.

4.7. Visual Performance Analysis

To move beyond tabular metrics and provide a deeper insight into the model's decision-making capabilities, we visualized the classification performance using confusion matrices and receiver operating characteristic (ROC) curves.

Confusion Matrix Analysis. Figure 4 compares the confusion matrices for the class with the largest improvement, Web Attack: Brute Force. With the baseline model, 207 of the 301 Brute Force flows in the held-out test partition were detected (recall 68.77%), 94 were missed, and 111 flows of other classes were falsely labelled as Brute Force (false positives). After augmentation, 296 flows were detected and 5 missed (recall 98.34%), with 129 false positives. The number of missed attacks therefore dropped from 94 to 5. The test partition was never used for GAN training or for building the augmented training

set; all synthetic samples were generated from the training partition. These numbers are for one run using a fixed seed; Table VI shows the distribution on five independent seeds.

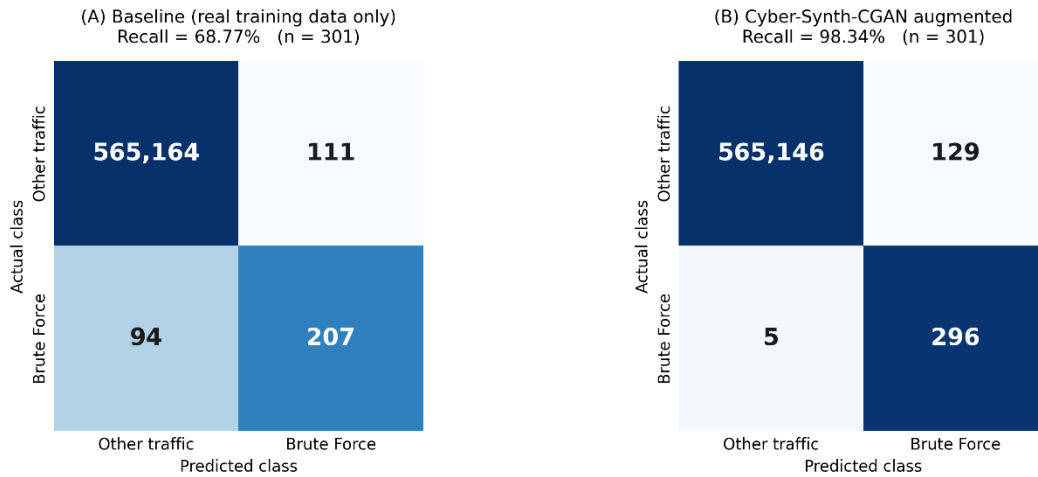


Fig. 4. Confusion matrices for Web Attack: Brute Force on the held-out test partition (301 positive flows, seed 42). (A) Baseline. (B) After Cyber-Synth-CGAN augmentation.

Threshold-Independent Evaluation. The ROC curves for the two classes on either side of the sample-size boundary are shown in Figure 5. Because the evaluation set is extremely imbalanced, with Web Attack: Brute Force accounting for 301 positive flows and 565,275 negative flows, the ratio is 1,878:1, which leads to ROC-AUC near unity for both models (0.9998 baseline, 0.9999 augmented) and makes no distinction between them. Average precision, which is useful when the classes are not well balanced, improves from 0.7919 to 0.8340 on Web Attack: Brute Force and from 0.3557 to 0.4267 on Web Attack: XSS. The baseline model already performs very well on brute-force flows; augmentation mainly shifts the decision boundary, increasing the recall from 68.77% to 98.34% while the false positive rate changes only from 0.000196 to 0.000228. This pattern means the recall improvement should not be read as an equivalent improvement in the classifier's ability to discriminate brute-force traffic. ROC-AUC is near ceiling for both models, and average precision rises by 0.0421 (0.7919 to 0.8340), whereas recall at the default decision threshold rises by 29.57 points. Most of the recall gain is therefore attributable to a shift of the decision boundary toward the minority class: augmentation changes the class balance seen by the Random Forest and hence the operating point at which brute-force flows are labelled positive. The XSS results show the same mechanism in the opposite direction, with ranking measures improving (average precision 0.3557 to 0.4267) while recall at the default threshold collapsed. It follows that part of the reported gain might also be obtained by adjusting the baseline model's decision threshold or class weights. We did not evaluate such a threshold-tuned baseline, and the reported improvement should be interpreted with this in mind.

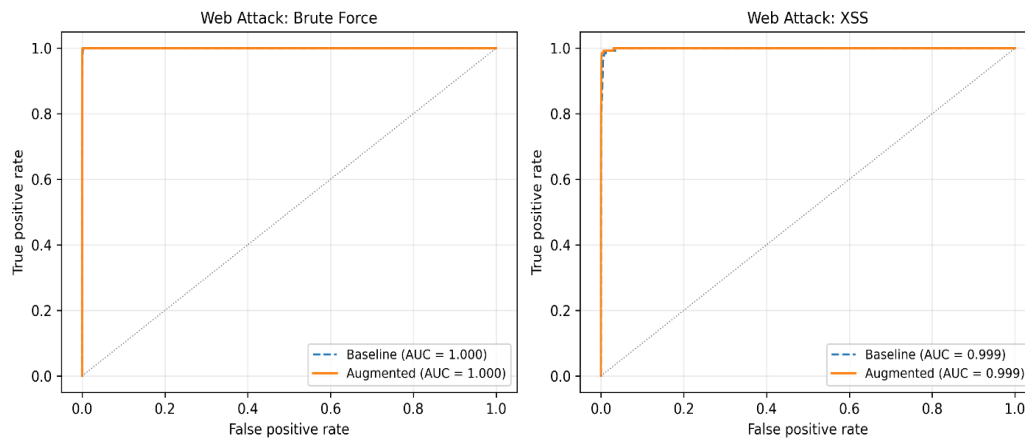


Fig. 5. ROC curves for Web Attack: Brute Force and Web Attack: XSS (seed 42). Both models achieved AUC ≥ 0.999 for either class; the curves are visually indistinguishable because the ROC-AUC saturates at the 1,878:1 class imbalance of this evaluation and cannot discriminate model quality here.

4.7.1. Quality Validation Results

To measure the statistical fidelity of the generated samples, we computed the distribution similarity measures of Section 3.6.3 (Eqs. (19)–(22)), presented in Table VII. All six augmented classes fail both criteria (KS $p < 0.05$ on all 78 features; MMD² between 0.8706 and 1.1598, above the 0.1 criterion of Section 3.6.3). Because the assessment is diagnostic rather than a filter, these samples were nevertheless used for training, and their downstream effect differs by class: they improved recall on Web Attack: Brute Force and degraded it on Web Attack: XSS.

TABLE VII. QUALITY ASSESSMENT OF GAN-GENERATED SAMPLES

Class	Training partition	Synthetic	KS p-value	MMD ² (normalized)	Validation
DoS slowloris	4,637	1,363	< 0.0001	0.8706	Fail
Bot	1,565	8,870	< 0.0001	0.9102	Fail
DoS SlowHTTPTest	4,399	1,601	< 0.0001	0.9196	Fail
Web Attack: Brute Force	1,206	9,588	< 0.0001	0.9945	Fail
SSH-Patator	4,717	1,283	< 0.0001	1.0205	Fail
Web Attack: XSS	522	10,956	< 0.0001	1.1598	Fail

MMD² is computed on the normalized $[-1, 1]$ feature space used by the networks, with a Gaussian kernel and median-heuristic bandwidth (Eq. (20)); on this space its value is bounded above by 4. No class passes the KS test at $p < 0.05$, and we make no claim of statistical indistinguishability; the benefit reported for Web Attack: Brute Force rests on the classification metrics of Sections 4.5 and 4.5.3 rather than on distributional agreement.

4.8. Computational Efficiency

Training time:

- GAN training: 1,531 epochs to early stopping
- Synthetic data generation: 33,661 samples
- Total pipeline: 47.6 minutes on a single CPU with no GPU

Memory requirements:

- Hardware memory: 16 GB RAM (peak usage was not measured separately)
- Dataset size: 2.8M samples $\times$ 78 features
- Synthetic data: 34K samples (negligible overhead)

The cost of one GAN training epoch grows linearly with the number of training rows, and generation cost grows linearly with the number of synthetic samples. We did not measure scaling empirically.

Once the generator is trained, generating samples requires only forward passes through the network. Only the downstream classifier is used for detection in operation, so its online cost is the same as that of a classifier trained with SMOTE-augmented data. The GAN stage needs to be repeated only when a new minority attack class appears or concept drift is detected.

4.9. Comparison with Baseline Methods

Table VIII compares Cyber-Synth-CGAN with classical oversampling using the same train/test partition, classifier configuration and evaluation protocol. The classical methods and Cyber-Synth-CGAN come from two different runs trained on different subsamples of the training data, so each change in recall is computed against the no-augmentation baseline of its own run (see the note below Table VIII). On Web Attack: Brute Force, with 1,206 real training samples, neither classical method helped: random oversampling reduced recall by 4.32 points and SMOTE by 6.65, while Cyber-Synth-CGAN raised it by 29.57. On Web Attack: XSS, with 522 real training samples, the ordering reverses: SMOTE raises recall by 8.46 points and random oversampling by 6.16, while Cyber-Synth-CGAN reduces it by 53.84. If confirmed on further datasets, this inversion would allow a sample-size gate to decide not only whether to augment a class but which method to use. In the pipeline evaluated here, the gate decided only whether to augment.

The present study does not include an experimental comparison with recent tabular generative models such as CTGAN and TVAE. Because results reported elsewhere differ in preprocessing, partitioning and classifier, we do not compare our figures with them; a head-to-head comparison under the same preprocessing and train/test split is left for future work.

TABLE VIII. COMPARISON WITH TRADITIONAL OVERSAMPLING TECHNIQUES

(CTGAN AND TVAE ARE DISCUSSED IN THE TEXT; A FULL EXPERIMENTAL COMPARISON IS PLANNED AS FUTURE WORK.)

Method	Brute Force	Δ (pp)	XSS	Δ (pp)	Time
No augmentation	68.11%	—	58.46%	—	1 min
Random oversampling	63.79%	-4.32	64.62%	+6.16	1 min
SMOTE	61.46%	-6.65	66.92%	+8.46	1 min
Cyber-Synth-CGAN	98.34%	+29.57	4.62%	-53.84	47.6 min

Note: Rows 1–3 come from an auxiliary run and row 4 from the main run of Section 4.5. Both runs use the same 80/20 partition, classifier configuration and seed, but the 60,000-per-class cap on real training data draws a different random subsample of the majority classes in each run, so the two runs were trained on different training subsamples. The no-augmentation Brute Force recall is therefore 68.11% in the auxiliary run and 68.77% in the main run (a difference of 0.66 points, within the seed-to-seed standard deviation of 1.87 points in Table VI); the no-augmentation XSS recall is 58.46% in both. The rows are not a paired evaluation on identical training data. Each Δ is the difference between a method’s recall and the no-augmentation recall of the same run: rows 2–3 are measured against 68.11% (Brute Force) and 58.46% (XSS), and row 4 against 68.77% and 58.46% (Table V).

Key insights:

- Above the sample-size boundary (Web Attack: Brute Force), generative augmentation changed recall by +29.57 points, against -4.32 for random oversampling and -6.65 for SMOTE, each measured against its own run’s baseline.
- Below the boundary (Web Attack: XSS), the ordering reverses: SMOTE and random oversampling improve recall by 8.46 and 6.16 points, respectively, while generative augmentation removes 53.84. On these two classes, a gate selecting between the two methods would have obtained the benefit in both cases, whereas either method applied to both classes was harmful in one of them; such a selecting gate was not implemented or tested in this study.

4.10. Design Rationale of Key Components

No ablation study was performed: no component was removed and the pipeline retrained under the corrected protocol. The following summarizes the rationale for each component; only the effect of the sample-size gate is supported by measurements in this paper.

- Signed log-transform scaling. The raw features span approximately -3.2×10^{10} to 2.1×10^9 . The transform compresses this range so that gradients remain bounded during training. We did not train the pipeline without it, so its contribution is a design motivation rather than a measured effect.
- Sample-size gate. This is the only component whose effect is measured. Augmenting Web Attack: XSS (522 training samples) reduced its recall by 53.84 points in the representative run and by 51.69 points across five seeds, while the three classes below 100 training samples were protected from augmentation by the gate.
- Feature-matching and diversity losses, adaptive learning rate and minority-focused sampling. These are standard stabilization techniques included to support convergence and sample diversity. Their individual contributions were not isolated.

A quantitative ablation of each component across multiple random seeds is left for future work.

5. DISCUSSION

5.1. Principal Findings

On CIC-IDS-2017, recall on Web Attack: Brute Force improved by 29.57 percentage points in the representative run (68.77% $\rightarrow$ 98.34%) and by 29.90 points across five seeds. Among the classes with more than 1,000 training samples, Brute Force — the only one with substantial headroom for improvement — showed a large gain, while the five others were statistically unchanged. Web Attack: XSS, the only augmented class below 1,000 training samples, degraded severely; Infiltration, SQL Injection and Heartbleed fell below the 100-sample floor and were not augmented. These observations are consistent with a sample-size boundary between 522 and 1,206 training samples, but they rest on two informative classes from one dataset and do not establish a general threshold.

5.2. The Paradox of Statistical Validation vs. Classifier Utility

A notable finding in our experiments was the discrepancy between the traditional statistical validation metrics (KS test, MMD) and the actual utility of synthetic data in downstream tasks. Although the Kolmogorov-Smirnov (KS) test rejects the null hypothesis for all 78 features ($p < 0.0001$), and the Maximum Mean Discrepancy (MMD²) values, computed on the

normalized $[-1, 1]$ feature space, fall in the range 0.8706–1.1598 — substantial but bounded divergence, since MMD^2 cannot exceed 4 on this space — the Random Forest classifier's recall on brute-force attacks improved from 68.77% to 98.34%.

We theorize that this phenomenon occurs because “distributional exactness” is not a prerequisite for “classification utility.” Statistical tests such as KS and MMD measure global fidelity—how perfectly the synthetic probability density function matches the real function across the entire high-dimensional space. However, for a discriminative classifier, the global shape of the distribution is secondary to the accuracy of the decision boundaries.

As illustrated conceptually in Figure 6, our GAN likely functions as a manifold support expander.

- **Real Minority Data (sparse):** The limited real samples often fail to cover the full “shape” of the attack class, leaving gaps near the decision boundary where the classifier defaults to the majority class (False Negatives).
- **Synthetic Data (Supportive):** Even if the GAN output is slightly shifted or “noisy” (causing poor KS scores), it successfully populates these sparse regions within the correct class manifold. By generating samples that are “close enough” to the real attacks but slightly varied, the GAN forces the classifier to push the decision boundary away from the attack cluster, thereby reducing the false negatives.

In Figure 6 (left), sparse real data leave the decision boundary undefined, resulting in misclassification; on the right, synthetic samples, even if imperfect, fill the intra-class region and shift the decision boundary away from the attack cluster. The figure is schematic and does not show measured data.

One interpretation consistent with our results is that the operative benefit of augmentation in sparse security settings is regularization of the decision boundary rather than distributional fidelity. We emphasize that our experiments demonstrate the effect but do not isolate the mechanism. A related possibility is suggested by the XSS results of Section 4.7: both ROC-AUC and average precision improve for that class even as recall collapses, indicating that augmentation there produced a calibration shift in the decision threshold rather than a change in the model's underlying ranking ability. Distinguishing between a manifold-support effect and a pure calibration effect would require controlled experiments that we have not performed.

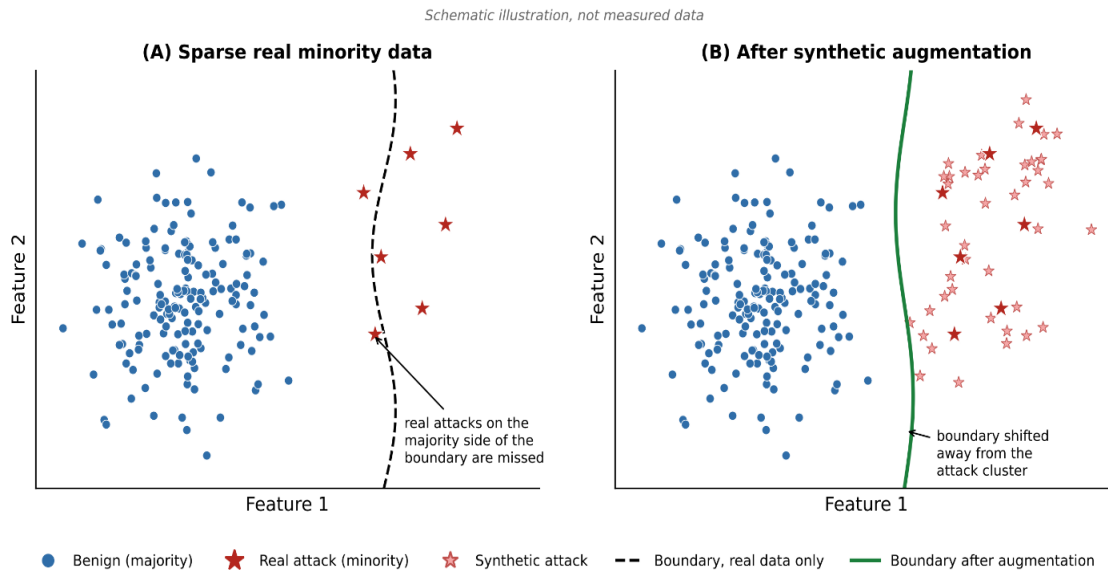


Fig. 6. Conceptual illustration of Manifold Support Expansion

5.3. Comparison with State-of-the-Art

We do not claim superiority over results reported in the literature. Published figures for minority-class recall on CIC-IDS-2017 differ in preprocessing, train/test partitioning and classifier choice, so they are not comparable with ours. The only controlled comparison in this study is Table VIII, in which all methods share a partition, a classifier and an evaluation protocol. In contrast to studies that report only aggregate performance, we examine how the effect of augmentation varies with the number of real training samples per class, and we suggest that practitioners calibrate a sample-size gate on their own data. Our results also indicate why domain-specific preprocessing such as signed log-transform scaling is needed for features spanning this dynamic range.

Tabular generative models such as CTGAN and TVAE have also been applied to intrusion-detection data. We have not compared them experimentally with our framework. These models do not, to our knowledge, include a sample-size gate that prevents harmful augmentation of very rare classes, and combining such a gate with CTGAN- or TVAE-style generators is an interesting direction for future research.

5.4. Computational, Ethical, and Security Considerations

Computationally, the framework is suitable for offline model development: the complete pipeline took 47.6 minutes on a standard CPU, without GPU hardware. Once the generator is trained, generating synthetic samples takes little time compared with training. Ethically, synthetic attack data could in principle be misused. Our generator, however, produces abstract flow-feature vectors, not packet payloads or executable exploit code, so its output cannot be used directly to carry out an attack. A remaining risk is the propagation of bias: if the original training data are biased, the synthetic data will reproduce and may amplify that bias. Organizations should therefore validate synthetic samples against known attack characteristics before deployment.

5.5. Limitations

Several limitations bound these conclusions. First, the recall gain on Web Attack: Brute Force is largely an operating-point effect (Section 4.7): average precision improves far less than recall, and a baseline with a tuned decision threshold was not evaluated. Second, the sample-size boundary rests on two informative classes from a single dataset and is not located more precisely than between 522 and 1,206 training samples. Because the two classes differ in attack type as well as in size, the effect of sample size cannot be separated from class-specific factors. A subsampling experiment on Web Attack: Brute Force (for example, 300, 600, 900 and 1,206 training samples), which would separate the two, was not performed; the boundary is therefore reported as a two-class observation. Third, the synthetic data fail standard fidelity tests for every augmented class (Table VII), so the benefit we report is a downstream classification effect rather than evidence that the generator reproduces the real distributions. Fourth, all results use one classifier family (Random Forest), and no component ablation was performed.

5.6. Future Directions

All the results in this study use a Random Forest classifier, chosen for its established strength in tabular network-flow features. Whether the benefit of augmentation transfers to other classifier families is a natural question that we have not evaluated under the corrected protocol, and we identify it as a limitation for future work.

Our study opens up several further research opportunities. Future work can investigate transfer learning to overcome the problem of ultra-rare attacks by fine-tuning pre-trained models on similar attack types. In addition, the sample size requirements can be reduced further with the application of progressively growing GANs. Federated learning also has potential, whereby organizations may share synthetic attack data to establish collaborative defenses without the need to disclose sensitive raw network traffic. Finally, creating explainable synthetic attacks would allow security analysts to perceive the individual characteristics that cause an attack to be identified, which would help bridge the gap between black-box generation and applied security intelligence.

6. CONCLUSIONS

This study introduces a new framework, known as Cyber-Synth-CGAN, that addresses a major weakness of network intrusion detection systems (NIDS): the detection of infrequent attack classes. By combining a log-transform scaling pipeline with a conditional Wasserstein GAN, we have shown that synthetic augmentation can improve the detection of a minority attack class in practice when enough real training samples are available. On CIC-IDS-2017, augmentation improved recall on brute-force attacks by 29.57 points in a representative run (from 68.77% to 98.34%) and by 29.90 points across five seeds, while it degraded the one augmented class with fewer than 1,000 training samples. These results suggest a sample-size boundary between 522 and 1,206 training samples on this dataset; confirming whether such a boundary generalizes requires further datasets and more classes near the boundary. Part of the recall gain reflects a shift of the decision boundary rather than improved ranking, as discussed in Section 4.7.

In addition to performance metrics, this work questions the so-called augment-all approach commonly found in the literature in favor of a decision-based approach, which is sensitive to the statistical shortcomings of generative models. We also demonstrated that traditional statistical validation measures, such as KS tests, are useful but should be viewed with caution in high-dimensional security settings.

The capability to model rare but devastating attacks artificially will gain increasing importance with the development of cyber threats. Future studies could build on this work by investigating transfer learning for ultra-minority classes, which were too small to augment in this study. In addition, the potential to implement the federated learning approach is promising; by training generative models cooperatively, organizations could benefit from sensitive attack data without breaching

privacy. The next important step in deploying this framework is to adapt it to concept drift in real-time operating environments.

Data and Code Availability

The complete implementation of Cyber-Synth-CGAN is publicly available at <https://github.com/zainabaabdulazeer/CyberSynth-CGAN-v.2> (release v1.0-bjai). The repository contains the preprocessing pipeline, the model definitions, the training and evaluation code, and the scripts used to produce the reported results, including the five-seed analysis, the oversampling baselines and the threshold-independent metrics. It also contains the logs of the runs reported in this paper and a list of software dependencies. All experiments use fixed random seeds (seed 42 for the representative run and seeds 42–46 for the multi-seed analysis), so the results can be regenerated. The CIC-IDS-2017 dataset is publicly available from the Canadian Institute for Cybersecurity [28] and is not redistributed in the repository.

Author Contributions

Rawaa M. Aljubouri: Software, Formal analysis, Investigation, Writing – original draft. **Esraa Muneam Abdulhussein:** Data curation, Validation, Visualization, Writing – review & editing. **Zainab A. Abdulazeer:** Conceptualization, Methodology, Supervision, Project administration, Writing – review & editing.

Conflicts of Interest

The authors declare no conflict of interest.

Funding

This research received no external funding.

Acknowledgment

We thank the University of Kerbala, the University of Babylon and AL-Furat Al-Awsat Technical University.

Ethical Approval

Not applicable. This study did not involve human participants or animals. It used only the publicly available CIC-IDS-2017 dataset, which consists of network traffic recorded in a controlled testbed environment and contains no personal data.

References

- [1] J. Li, W. Zong, Y.-W. Chow, and W. Susilo, “Mitigating class imbalance in network intrusion detection with feature-regularized GANs,” *Future Internet*, vol. 17, no. 5, Art. no. 216, 2025, doi: 10.3390/fi17050216.
- [2] A. Abdelkhalek and M. Mashaly, “Addressing the class imbalance problem in network intrusion detection systems using data resampling and deep learning,” *J. Supercomput.*, vol. 79, no. 10, pp. 10611–10644, 2023, doi: 10.1007/s11227-023-05073-x.
- [3] S. Bagui, D. Mink, S. Bagui, S. Subramaniam, and D. Wallace, “Resampling imbalanced network intrusion datasets to identify rare attacks,” *Future Internet*, vol. 15, no. 4, Art. no. 130, 2023, doi: 10.3390/fi15040130.
- [4] V. Shanmugam, R. Razavi-Far, and E. Hallaji, “Addressing class imbalance in intrusion detection: A comprehensive evaluation of machine learning approaches,” *Electronics*, vol. 14, no. 1, Art. no. 69, 2025, doi: 10.3390/electronics14010069.
- [5] M. H. A. Hamid, M. Yusoff, and A. Mohamed, “Survey on highly imbalanced multi-class data,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 6, 2022, doi: 10.14569/IJACSA.2022.0130627.
- [6] I. Goodfellow et al., “Generative adversarial nets,” in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 27, 2014, pp. 2672–2680.
- [7] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, “Training generative adversarial networks with limited data,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 12104–12114.
- [8] S. Allagi, T. Pawan, and W. Y. Leong, “Enhanced intrusion detection using conditional-tabular-generative-adversarial-network-augmented data and a convolutional neural network: A robust approach to addressing imbalanced cybersecurity datasets,” *Mathematics*, vol. 13, no. 12, Art. no. 1923, 2025, doi: 10.3390/math13121923.
- [9] W. Lim, K. S. C. Yong, B. T. Lau, and C. C. L. Tan, “Future of generative adversarial networks (GAN) for anomaly detection in network security: A review,” *Comput. Secur.*, vol. 139, Art. no. 103733, 2024, doi: 10.1016/j.cose.2024.103733.
- [10] M. Al-Ajlan and M. Ykhlef, “A review of generative adversarial networks for intrusion detection systems: Advances, challenges, and future directions,” *Comput. Mater. Contin.*, vol. 81, no. 2, pp. 2053–2076, 2024, doi: 10.32604/cmc.2024.055891.

- [11] Y. Zhang, R. C. Muniyandi, and F. Qamar, "A review of deep learning applications in intrusion detection systems: Overcoming challenges in spatiotemporal feature extraction and data imbalance," *Appl. Sci.*, vol. 15, no. 3, Art. no. 1552, 2025, doi: 10.3390/app15031552.
- [12] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Syst. Appl.*, vol. 73, pp. 220–239, 2017, doi: 10.1016/j.eswa.2016.12.035.
- [13] M. Wang, N. Yang, Y. Guo, and N. Weng, "Learn-IDS: Bridging gaps between datasets and learning-based network intrusion detection," *Electronics*, vol. 13, no. 6, Art. no. 1072, 2024, doi: 10.3390/electronics13061072.
- [14] M. S. Milosevic and V. M. Ciric, "Extreme minority class detection in imbalanced data for network intrusion," *Comput. Secur.*, vol. 123, Art. no. 102940, 2022, doi: 10.1016/j.cose.2022.102940.
- [15] Y. Yang, Y. Gu, and Y. Yan, "Machine learning-based intrusion detection for rare-class network attacks," *Electronics*, vol. 12, no. 18, Art. no. 3911, 2023, doi: 10.3390/electronics12183911.
- [16] G. Agrawal, A. Kaur, and S. Myneni, "A review of generative models in generating synthetic attack data for cybersecurity," *Electronics*, vol. 13, no. 2, Art. no. 322, 2024, doi: 10.3390/electronics13020322.
- [17] W. Tian, Y. Shen, N. Guo, J. Yuan, and Y. Yang, "VAE-WACGAN: An improved data augmentation method based on VAEGAN for intrusion detection," *Sensors*, vol. 24, no. 18, Art. no. 6035, 2024, doi: 10.3390/s24186035.
- [18] B. A. Alabsi, M. Anbar, and S. D. A. Rihan, "Conditional tabular generative adversarial based intrusion detection system for detecting DDoS and DoS attacks on the Internet of Things networks," *Sensors*, vol. 23, no. 12, Art. no. 5644, 2023, doi: 10.3390/s23125644.
- [19] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, PMLR, vol. 70, 2017, pp. 214–223.
- [20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 30, 2017, pp. 5767–5777.
- [21] X. Yao et al., "GMT-WGAN: An adversarial sample expansion method for ground moving targets classification," *Remote Sens.*, vol. 14, no. 1, Art. no. 123, 2022, doi: 10.3390/rs14010123.
- [22] Z. Wang, H. Zhang, W. Huang, X. Chen, N. Tang, and Y. An, "A Wasserstein generative adversarial network with gradient penalty for active sonar signal reverberation suppression," *Front. Mar. Sci.*, vol. 10, Art. no. 1280305, 2023, doi: 10.3389/fmars.2023.1280305.
- [23] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*, doi: 10.48550/arXiv.1411.1784.
- [24] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018.
- [25] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network," *Sensors*, vol. 19, no. 11, Art. no. 2528, 2019, doi: 10.3390/s19112528.
- [26] G. Andresini, A. Appice, L. De Rose, and D. Malerba, "GAN augmentation to deal with imbalance in imaging-based intrusion detection," *Future Gener. Comput. Syst.*, vol. 123, pp. 108–127, 2021, doi: 10.1016/j.future.2021.04.017.
- [27] S. M. Kasongo and Y. Sun, "Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset," *J. Big Data*, vol. 7, Art. no. 105, 2020, doi: 10.1186/s40537-020-00379-6.
- [28] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2018, pp. 108–116, doi: 10.5220/0006639801080116.