Distinguishing Signal from Noise in 5G MIMO Systems Using Generative Adversarial Networks

Damianos Diasakos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1084632@ac.upatras.gr

Nikolaos Prodromos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1072549@ac.upatras.gr

Apostolos Gkamas
Department of Chemistry
University of Ioannina
Ioannina, Greece
Email: gkamas@uoi.gr

Vasileios Kokkinos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: kokkinos@cti.gr

Christos Bouras
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: bouras@upatras.gr

Philippos Pouyioutas
Computer Science Department
University of Nicosia
Nicosia, Cyprus
Email: pouyioutas.p@unic.ac.cy

2025 12th IFIP International Conference on New Technologies, Mobility and Security (NTMS) | 979-8-3315-5276-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/NTMS65597.2025.11076915

***Abstract*—In recent years, Generative Adversarial Networks (GANs) have emerged as powerful tools for improving signal processing in advanced communication systems, particularly in the context of 5G networks. In this paper, we present a novel approach for distinguishing signal from noise in 5G Multiple Input Multiple Output (MIMO) systems using GANs. Our method leverages the generative capabilities of GANs to produce realistic noise signals and the discriminative power of GANs to accurately identify real signals amidst noise. By training the GAN on a combination of real-world noisy signals and pure noise, our model achieves robust signal detection and classification. We evaluate our approach using synthetic data, demonstrating significant improvements over other techniques such as the autoencoders. Our results highlight the potential of GANs in enhancing the reliability and performance of 5G MIMO communications.**

Keywords—5G MIMO, Generative Adversarial Networks (GANs), Signal Detection, Noise Classification, Machine Learning, Wireless Communications, Autoencoder

I. Introduction

Signal detection and noise classification are critical in 5G Multiple Input Multiple Output (MIMO) networks, where accurately distinguishing signals from noise ensures reliable communication and optimal spectrum use. Over the years, researchers have explored various techniques to address these challenges, ranging from traditional signal processing methods to advanced machine learning models.

Traditional methods such as adaptive filtering, Fourier transforms, and wavelet transforms have been widely used to enhance the Signal-to-Noise Ratio (SNR). Adaptive filters like the Wiener filter are effective under stable conditions, while wavelet transforms enable multi-scale analysis to isolate noise components. However, in dynamic and complex noise environments typical of 5G systems, these approaches often fall short [1], [2]. For instance, wavelet-based methods may struggle with highly variable interference patterns in dense deployments.

Machine learning approaches, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promise in signal discrimination by learning features from noisy data. CNNs excel at identifying spatial patterns, and RNNs capture temporal dependencies in signal sequences. However, both require large amounts of labeled training data, which is often difficult to obtain in practice, and may not generalize well across diverse noise conditions [3].

Feature extraction plays a crucial role in bridging raw signal data and learning models. Short-Time Fourier Transform (STFT) is widely used to convert time-domain signals into frequency-domain representations, capturing both frequency components and their evolution over time [4]. To emphasize perceptually relevant frequency bands, Mel-spectrograms are often used, providing robust visual representations suitable for deep learning [5].

Recently, Generative Adversarial Networks (GANs) have emerged as powerful tools in machine learning, capable of generating highly realistic synthetic data. GANs consist of a generator and a discriminator trained adversarially, and have been applied in wireless communications for tasks like channel estimation, noise modeling, and data reconstruction [6]. For example, GANs have been used to simulate realistic noise environments and modify time-scale features in speech signals [7]. However, their potential in signal detection and noise classification for 5G MIMO systems remains underexplored.

In this paper, we extend GAN applications to signal detection in 5G MIMO systems, using STFT and Mel-spectrograms for feature extraction. These transformations allow the GAN to effectively distinguish and denoise signals by learning from realistic noise samples that mimic complex interference patterns typical of 5G environments. While the current study focuses on single-channel signal processing, the proposed architecture is inherently scalable and can be extended to full MIMO systems. By leveraging multiple antenna streams and feeding multi-channel spectrograms into the GAN, the model could exploit inter-antenna correlations, potentially enhancing noise suppression in dense, real-world deployments. Our GAN-based approach is adaptive, requires minimal manual intervention, and aims to outperform traditional techniques including CNN-based autoencoders in both accuracy and robustness, demonstrating strong potential for future 5G applications.

The rest of the paper is structured as follows: Section II reviews recent GAN and autoencoder methods in signal detection, Section III details the mathematical model of GANs used, Section IV describes signal generation techniques, Section V presents experimental results comparing our GAN-based approach with CNN-based autoencoders, and Section VI discusses implications, applications, and limitations.

II. Model Architectures and Training Frameworks

As seen in Figure 1, GANs consist of two primary components: Generator (G): The generator G(z;θg) takes a random noise vector z as input and generates synthetic data (1). The goal of the generator is to produce data that is indistinguishable from real data.

$$G(z;\theta g) = synthetic\ data \quad (1)$$

Discriminator (D): The discriminator D(x;θd) takes either real data or synthetic data as input and outputs a probability D(x) representing the likelihood that the input data is real (2). The goal of the discriminator is to correctly classify real and synthetic data.

$$D(x;\theta d) = P(real\ data \mid x) \quad (2)$$

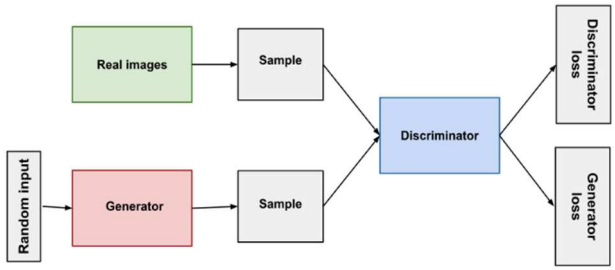

Fig. 1. Architecture of a GAN

The GAN is trained using the following minimax objective function (3):

$$minGmaxDV(D,G) = E\_x \sim pdata(x)[logD(x)] + E\_z \sim pz(z)[log(1 - D(G(z)))] \quad (3)$$

Here, pdata(x) is the distribution of real data, and pz(z) is the distribution of the noise vector z.

During training, the discriminator is updated to maximize the probability of correctly classifying real and synthetic data, while the generator is updated to minimize the probability of the discriminator identifying synthetic data as fake. This adversarial training process enables the GAN to produce highly realistic noise and train a robust discriminator capable of distinguishing between real noisy signals and pure noise. By leveraging the generative and discriminative capabilities of GANs, our model provides a significant improvement over traditional signal processing techniques and CNN-based models in distinguishing signals from noise in 5G MIMO systems [8],[9],[10].

As seen in Figure 2 autoencoder network comprises two main components:

Encoder:

- Maps the input data x into a latent space z.
- Involves a series of transformations through neural network layers, such as fully connected layers or convolutional layers, depending on the type of data.
- The encoder's function can be mathematically represented as: $z = f_{encoder}(x)$, where fencoder represents the encoder's function.
- Outputs z, a compact representation of the input data.

The decoder:

- Reconstructs the data from the latent space representation z.
- Maps z back to the original data space, aiming to approximate the input x as closely as possible.
- The decoder's function can be mathematically represented as: $\hat{x} = f_{decoder}(z)$, where $\hat{x}$ is the reconstructed data and $f_{decoder}$ represents the decoder's function.

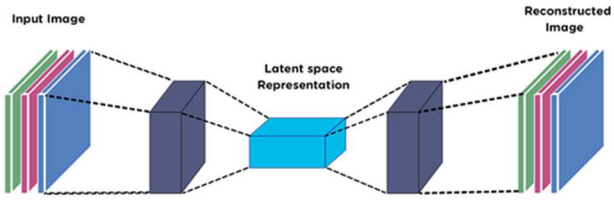

Fig. 2. Architecture of an autoencoder

The loss function that the autoencoder for binary classification tasks like this is usually the Binary Cross-Entropy (BCE). The mathematical model (4) is shown below:

$$L(x, x\_hat) = -(1/n) * \sum[xi * log(xi\_hat) + (1 - xi) * log(1 - xi\_hat)] \quad (4)$$

where n is the number of data points, x_i represents the i-th element of the input, and x_i_hat denotes the corresponding element of the reconstructed output [11],[12].

III. Analysis of Signal Generation and Models

To transform the signals into a form suitable for deep learning model input, feature extraction was performed using the STFT and Mel-spectrogram. The STFT was employed to convert the time-domain signals into the frequency domain, capturing both the frequency components and their evolution over time. This transformation yielded spectrograms, which are visual representations of the signal's frequency content over time. To further focus on perceptually important features, Mel-spectrograms were computed, emphasizing frequencies that are more relevant to human perception. These spectrograms were then resized to a standardized dimension of 64x64 pixels, facilitating uniformity in the input data fed into the neural networks [13],[14].

Algorithm 1 – Signal Generation

```
function generate_signal(fs, duration, frequency, noise_level=0.5):
    t = np.arange(0, duration, 1.0/fs)  # Time vector
    signal = np.sin(2 *np.pi * frequency * t)  # Pure sine wave signal
    noisy_signal = signal + np.random.normal(0, noise_level, t.shape)  # Add Gaussian noise
    return t, signal,noisy_signal
```

Normalization was an essential preprocessing step, ensuring that the spectrogram data fell within a suitable range for neural network training. This step involved scaling the spectrogram values to a range between 0 and 1, which prevents potential issues associated with unnormalized input data, such as vanishing or exploding gradients during model

training. The normalized spectrograms of the clean and noisy signals, as illustrated in the second image, visually underscore the efficacy of the preprocessing pipeline. The clean spectrograms exhibit well-defined frequency components, while the noisy spectrograms show the added noise, which obscures the clarity of these components. Algorithm 1 generates synthetic noisy signals using sine waves and added Gaussian noise. Algorithm 2 computes the STFT and Mel-spectrogram to extract features from the signals. Algorithm 3 normalizes the spectrogram data between 0 and 1, preparing it for training.Together, these three steps form the preprocessing pipeline for both the GAN and the autoencoder.

Algorithm 2 – Spectrogram Making and Feature Extraction

```
function extract_features(signal, sr):
    # Compute Short-Time Fourier Transform (STFT)
    stft = abs(STFT(signal, n_fft=2048, hop_length=512))
    # Compute Mel Spectrogram
    mel_spectrogram = MelSpectrogram(STFT_power(stft), sr, n_mels=64)

    # Convert to Decibels
    mel_spectrogram_db = PowerToDB(mel_spectrogram)
    # Resize to 64x64
    mel_spectrogram_db_resized = Resize(mel_spectrogram_db, (64, 64))1
    return mel_spectrogram_db_resized
```

Algorithm 3 – Spectrogram Normalization

```
function normalize_spectrogram(spectrogram):
    # Find the minimum value in the spectrogram
    min_value = find_minimum_value(spectrogram)
    # Find the maximum value in the spectrogram
    max_value = find_maximum_value(spectrogram)
    # Subtract the minimum value from each element in the spectrogram
    normalized_spectrogram = spectrogram - min_value
    # Divide each element in the spectrogram by the difference between the maximum and minimum values
    normalized_spectrogram = normalized_spectrogram / (max_value - min_value)
    # Return the normalized spectrogram
    return normalized_spectrogram
```

The process of generating spectrograms of pure noise follows a similar procedure. An empty list is initialized to store the spectrograms. For each batch, pure Gaussian noise is generated using the NumPy library, which provides robust tools for numerical computations, including random number generation. Specifically, the noise array is created with a length matching the product of the sampling frequency f_s and the signal duration. The spectrogram of this noise and its features are then computed by using the *extract_features* function. The generator utilizes these features to predict new noisy spectrograms, which are subsequently used to train the discriminator. This iterative process enables the generator to produce increasingly realistic results over time.

Two spectrograms made can be seen in Figures 3 and 4. The upper spectrogram corresponds to a clean signal, while the lower shows its noisy counterpart. Both are before their normalization.

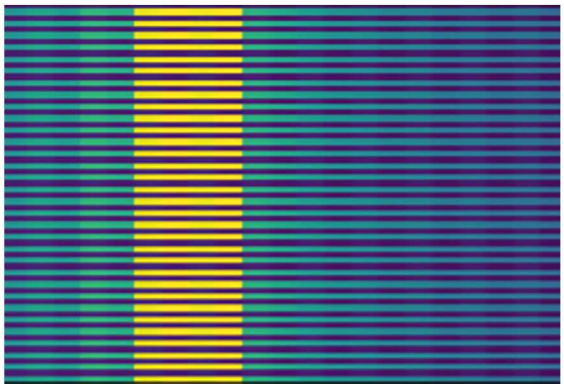

Fig. 3. Clean Spectrogram

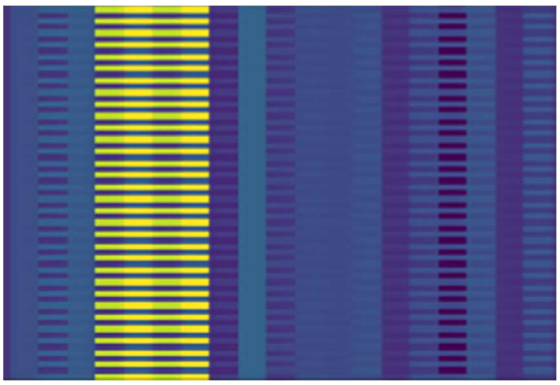

Fig. 4. Noisy Spectrogram

So, the signal generation process consists of the initialization of an empty list to store the spectrograms and for each batch, the use of generate_feature_batch function produce a noisy and a clean signal. This function has the generate_signal and the feature extraction functions as well as the normalization function. Using generate_feature_batch, the spectrogram of the produced signal is computed using the STFT through the scipy.signal.spectrogram function, which decomposes the signal into its frequency components over time. This results in a 2D array representing the spectrogram, with frequency bins along one axis and time bins along the other. The spectrogram and its features are normalized and then they are returned from the function to serve as inputs for the training models. Below in Figure 5, parts of the noisy spectrogram after normalization can be observed.

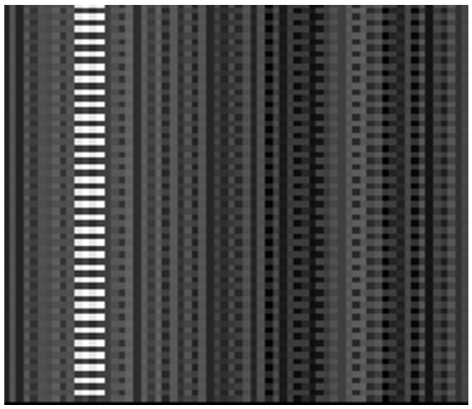

Fig. 5. Noisy Spectrogram Normalized

The architecture of the GAN implemented in this model is carefully designed to handle the task of denoising spectrograms, with the generator and discriminator playing complementary roles. The generator starts with an input layer that accepts a 64x64x1 spectrogram, representing a noisy signal. The model's first convolutional block applies a 2D convolution with 128 filters, using a kernel size of 3x3 and a stride of 2x2. This layer's purpose is to downsample the input and extract initial spatial features. By employing a LeakyReLU activation function with an alpha of 0.2, the network avoids the common issue of dead neurons, ensuring that even non-active regions maintain a small gradient, which is crucial for stable training [15]. BatchNormalization follows, stabilizing the output of this layer by normalizing the activations, thus speeding up the training process and making it more robust to changes in the data distribution.

Moving deeper into the network, the second convolutional block continues the process of downsampling and feature extraction but reduces the number of filters to 64. This block also utilizes a 3x3 kernel and a 2x2 stride, further refining the spatial features while maintaining the model's ability to learn complex patterns. Again, LeakyReLU and BatchNormalization are applied, ensuring that the network remains stable and efficient during training. These layers are designed to capture increasingly abstract features from the input spectrogram, which are essential for the generator to later reconstruct a clean output.

The upsampling process begins with two Conv2DTranspose layers. These layers are crucial for the generator as they increase the spatial dimensions of the feature maps, effectively reversing the downsampling done by the earlier layers. The first of these layers uses 64 filters, while the second uses 32, both with a 3x3 kernel and 2x2 stride, progressively refining the resolution of the output spectrogram. LeakyReLU activations follow each transposed convolution, ensuring that the generator retains the non-linearity needed to model complex relationships within the data. The final output layer is another Conv2DTranspose layer, but with just one filter, outputting a single-channel spectrogram. The use of a tanh activation function in this layer is particularly apt as it outputs values in the range of [-1, 1], matching the normalized range of the spectrogram data and effectively completing the denoising transformation.

On the discriminator side, the model begins with a similar input structure, processing a 64x64x1 spectrogram that could either be real (clean) or fake (denoised by the generator). The first convolutional block applies 64 filters with a 3x3 kernel and a 2x2 stride, initiating the feature extraction process by focusing on basic patterns. LeakyReLU is again used to maintain the gradient flow throughout the network. To prevent the model from overfitting, a dropout layer with a rate of 0.3 is introduced, randomly setting some of the activations to zero during training. This regularization technique encourages the model to learn more generalized features that are not overly reliant on any specific patterns.

As the discriminator moves deeper, the complexity of the extracted features increases. The second convolutional block uses 128 filters, continuing the downsampling process and extracting more intricate features from the input spectrogram. This block also includes a 0.3 dropout rate to further combat overfitting. The third convolutional block introduces 256 filters, making the network even more capable of distinguishing between real and fake data by capturing fine-grained details. A higher dropout rate of 0.4 is used here to ensure that the model remains generalizable despite its increased complexity.

The flattened output of these convolutional layers feeds into a fully connected layer with 128 units. This dense layer distills the information down to its most essential features, using a LeakyReLU activation to preserve the network's ability to model complex data relationships. A final dropout layer with a 0.2 rate provides additional regularization before the output layer. The output layer consists of a single neuron with a sigmoid activation function, outputting a probability that indicates whether the input spectrogram is real or generated. This setup allows the discriminator to perform binary classification, determining with increasing accuracy whether the spectrogram it receives is authentic or produced by the generator.

The combination of these architectural elements in both the generator and discriminator creates a robust GAN framework capable of denoising spectrograms while maintaining the delicate balance between noise reduction and detail preservation. The careful use of LeakyReLU, BatchNormalization, and Dropout layers throughout both networks ensures that the models are well-regularized and capable of handling the complexities inherent in this task.

IV. Process Description and Model Evaluation

In this process, a time vector t is generated using the NumPy library, known for its robust numerical computation tools. A pure sinusoidal signal simulates the desired signal, while Gaussian noise is added using NumPy’s random number generation to mimic real-world conditions. The STFT extracts signal features, converting data to the frequency domain. Mel-spectrograms are computed and resized to 64x64 pixels for uniform neural network inputs, with values normalized to [0, 1] to ensure stable training. The GAN architecture consists of a generator that denoises noisy spectrograms and a discriminator that classifies spectrograms as real or generated. Both models employ convolutional layers, dropout for regularization, and LeakyReLU activation to prevent inactive neurons. Model performance is evaluated using Root Mean Squared Error (RMSE) and Mean Squared Error (MSE), with simulation parameters detailed in Table I. While the training duration for each epoch was approximately 20 minutes on a mid-range GPU, inference on a single spectrogram sample took less than 100 ms. The autoencoder offered slightly faster inference but with a small trade-off in output detail. These results suggest that the GAN is

computationally feasible for offline or near-real-time scenarios.

TABLE I. SIMULATION PARAMETERS

Parameter	*Value*
NFFT (STFT Window)	2048
Number of Mel filters	64
Generator Filters	[128,64]
Discriminator Filters	[64,128,256]
Dropout Rates	[0.4,0.3]
Activation Functions	LeakyReLU,Tanh,Sigmoid
Evaluation Metrics	RMSE,MSE

V. SIMULATION RESULTS

The experiments in this study are designed to evaluate the effectiveness of GANs and Autoencoders in denoising spectrograms derived from noisy synthetic signals. The primary objective is to assess the ability of each model to reconstruct clean spectrograms from their noisy counterparts, with the performance metrics being Root RMSE and MSE. The autoencoder training curve can be seen in Figure 6 below.

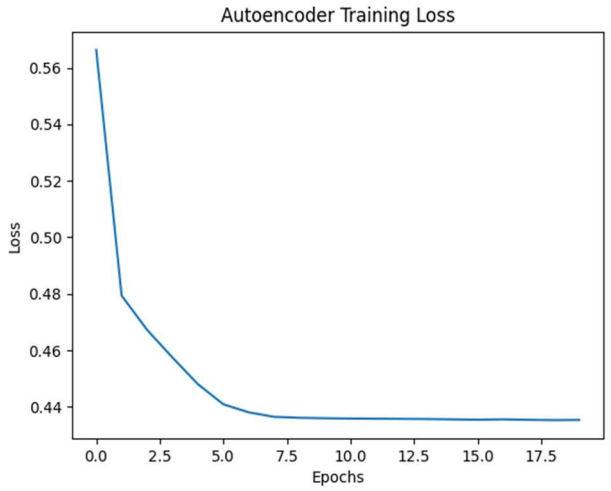

Fig. 6. Autoencoder Training Curve

The training process for the Autoencoder is conducted over 20 epochs, with the model progressively learning to minimize the binary cross-entropy loss between the input noisy spectrograms and the target clean spectrograms. The training loss curve, depicted in Figure 6, shows a steep decline in loss during the initial epochs, indicating rapid learning as the model quickly captures the basic structure of the spectrograms. However, after approximately 10 epochs, the rate of improvement plateaus, suggesting that the model has learned most of the general features necessary for denoising and is making only minor refinements in subsequent epochs.

Despite this rapid convergence, the final reconstructed spectrograms from the Autoencoder, as shown in Figure 7, reveal some limitations.

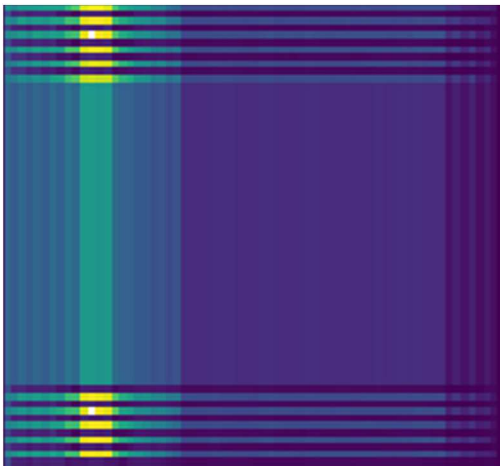

Fig. 7. Autoencoder Denoised Output

While the Autoencoder effectively reduces noise, it also introduces blurring, particularly in areas of high frequency content. This blurring effect is characteristic of the model's reliance on compressed latent space representations, which, while efficient, can lead to the loss of fine-grained details. The average RMSE of 0.155 and MSE of 0.0240 further reflect this trade-off between noise reduction and detail preservation.

In contrast, the training of the GAN model follows a more complex process due to its adversarial nature. The GAN consists of two components: the generator, which attempts to create spectrograms that mimic clean signals, and the discriminator, which seeks to distinguish between real (clean) and generated (denoised) spectrograms. The training involves iteratively improving the generator's ability to produce realistic spectrograms while simultaneously enhancing the discriminator's capability to correctly identify genuine versus generated data.

The training process for the GAN, like the Autoencoder, spanned 20 epochs. However, the GAN's dynamics are inherently more volatile due to the adversarial interplay between the generator and discriminator. Throughout training, the GAN progressively learned to produce denoised spectrograms that visually resemble the clean targets more closely than those produced by the Autoencoder. This is evident in Figure 8, where the GAN output retains more detail—particularly in high-frequency components. However, this increased detail comes with trade-offs. Some outputs include subtle artifacts, such as localized blurring or faint edge effects that are not present in the original signals. These artifacts likely result from the adversarial nature of GAN training, which can lead to overfitting when the dataset is limited or when the generator and discriminator become imbalanced. To mitigate these risks, we employed dropout layers and early stopping during training to stabilize learning and prevent mode collapse. While the artifacts observed were minor and did not significantly impair signal clarity, their presence suggests opportunities for improvement using advanced regularization techniques or more stable GAN variants.

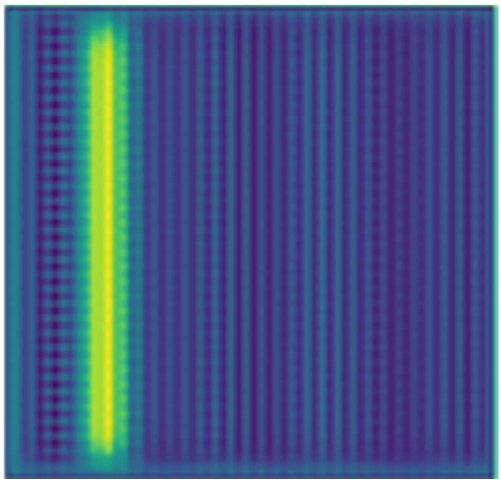

Fig. 8. GAN Denoised Output

Table II presents the RMSE and MSE scores for both models across multiple test steps (ranging from 50 to 1000), consistently showing the GAN having a slight edge over the Autoencoder. Based on these results, the GAN achieved an estimated average RMSE of 0.1530 ± 0.0022 and MSE of 0.0228 ± 0.0004, confirming its stable and reliable denoising performance across varying signal lengths and types.

TABLE II. RMSE & MSE SCORES

Testing Steps	*Model*	*RMSE*	*MSE*
50	GAN	0.150	0.0225
50	Autoencoder	0.155	0.0240
100	GAN	0.152	0.0231
100	Autoencoder	0.153	0.0243
250	GAN	0.155	0.0233
250	Autoencoder	0.157	0.0241
500	GAN	0.156	0.0229
500	Autoencoder	0.159	0.0238
1000	GAN	0.152	0.0224
1000	Autoencoder	0.153	0.0239

Quantitatively, the GAN achieved an average RMSE of 0.15 and MSE of 0.0225, slightly outperforming the Autoencoder. This improvement, albeit small, highlights the GAN's superior capability in capturing and reconstructing complex features in noisy environments, making it a more effective tool for denoising tasks in high-dimensional data like spectrograms.

VI. Conclusion and Future Work

In this study, we explored the effectiveness of GANs and Autoencoders in the task of denoising spectrograms generated from synthetic noisy signals, a problem relevant to 5G MIMO systems and other advanced communication technologies. Our experimental results demonstrated that both models are capable of significant noise reduction, with the GAN slightly outperforming the Autoencoder in terms of RMSE and MSE.

Visual comparisons revealed that while the Autoencoder reduces noise effectively, it introduces a smoothing effect that can blur high-frequency details. In contrast, the GAN retains sharper transitions and more distinct frequency bands, resulting in more detailed reconstructions. Although the GAN occasionally introduces minor artifacts due to adversarial training, its outputs remain closer to the original clean signals, demonstrating its strength in detail preservation.

The relatively narrow performance gap may reflect the Autoencoder's strong baseline capability on synthetic Gaussian noise. Future work will explore techniques to enhance the GAN's advantage, including perceptual loss functions, SSIM-based evaluation metrics, and advanced GAN variants such as conditional GANs (cGAN), Wasserstein GANs (WGAN), and Least Squares GANs (LSGAN).

To evaluate real-world applicability, future experimentation will focus on real 5G datasets, where noise characteristics are more complex and varied. Additional experiments using more complex modulated signals (e.g., QAM, OFDM) will better simulate 5G conditions and help assess model generalization. We also plan to test the model's scalability using higher-resolution spectrograms and longer signal durations.

Furthermore, to provide a broader performance context, future studies will compare the GAN not only with deep learning models but also with traditional denoising techniques such as Wiener filtering, wavelet thresholding, and spectral subtraction.

Finally, integrating these models into real-time signal processing pipelines is a key goal. This includes optimizing them for faster inference and lower computational load, while keeping time complexity low enough to support practical deployment in embedded or resource-constrained environments.

Acknowledgment

The research project was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 02440).

References

[1] Skan.ai, "How to Use Machine Learning to Separate the Signal from the Noise," [Online]. Available: https://www.skan.ai/blogs/how-to-use-machine-learning-to-separate-the-signal-from-the-noise-skan.

[2] T. Xu and Z. Wei, "Waveform Defence Against Deep Learning Generative Adversarial Network Attacks," 2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), Porto, Portugal, 2022, pp. 503-508, doi: 10.1109/CSNDSP54353.2022.9907905.

[3] E. Cohen, F. Kreuk and J. Keshet, "Speech Time-Scale Modification With GANs," in IEEE Signal Processing Letters, vol. 29, pp. 1067-1071, 2022, doi: 10.1109/LSP.2022.3164361.

[4] E. Balevi and J. G. Andrews, "Wideband Channel Estimation with a Generative Adversarial Network," *IEEE Wireless Communications Letters*, vol. 10, no. 3, pp. 592–596, 2021, doi: 10.1109/LWC.2020.3047435.

[5] Y. Xu, Q. Zhang, and J. Wu, "End-to-End Environmental Sound Classification Using Attention-Based Convolutional Neural Networks and Mel-Spectrogram Features," IEEE Access, vol. 8, pp. 31530-31541, 2020, doi: 10.1109/ACCESS.2020.2972892.

[6] Y. Huleihel and H. H. Permuter, "Low PAPR MIMO-OFDM Design Based on Convolutional Autoencoder," arXiv preprint

arXiv:2301.05017, 2023. [Online]. Available: https://arxiv.org/abs/2301.05017.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in Neural Information Processing Systems, vol. 27, 2014, pp. 2672–2680.

[8] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.

[9] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[10] Creswell, Antonia, et al. "Generative adversarial networks: An overview." IEEE signal processing magazine 35.1 (2018): 53-65.

[11] Almazrouei, Ebtesam, et al. "Using autoencoders for radio signal denoising." Proceedings of the 15th ACM International Symposium on QoS and Security for Wireless and Mobile Networks. 2019.

[12] Yildirim, Ozal, Ru San Tan, and U. Rajendra Acharya. "An efficient compression of ECG signals using deep convolutional autoencoders." Cognitive Systems Research 52 (2018): 198-211.

[13] Shen, Jonathan, et al. "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions." 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2018.

[14] Hwang, Yeongtae, et al. "Mel-spectrogram augmentation for sequence to sequence voice conversion." arXiv preprint arXiv:2001.01401 (2020).

[15] TensorFlow, "tf.keras.layers.LeakyReLU," TensorFlow Core API,[Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU