ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

Generative Channel Modeling for POWDER Datasets using GANs

Sumit Roy, Liu Cao, Laxman Balamurugan

November 19, 2024

Contents

1 Public POWDER Datasets Description **2**
1.1 POWDER Scenario Setup . . . 2
1.2 POWDER dataset description . . . 3
1.3 Motivation & Contribution . . . 3

2 GAN-to-POWDER **5**
2.1 Generative AI Models for PHY Layer Communications . . . 5
2.1.1 Generative AI/ML Models: Recap . . . 5
2.1.2 Motivation of Choosing GAN for Channel Modeling . . . 6
2.2 Fundamentals of GANs . . . 7
2.2.1 Components of GAN . . . 7
2.2.2 Objective Functions of GAN . . . 8
2.3 Proposed GAN-to-POWDER Implementation . . . 9
2.4 A Recap of ADAM (Adaptive Moment Estimation) . . . 11

3 Performance Evaluation **13**
3.1 Evaluation Metric 1: Loss Function of Generator and Discriminator . . . 13
3.1.1 Discriminator Training Loss . . . 13
3.1.2 Generator Training Loss . . . 14
3.2 Evaluation Metric 2: Average power and variance of the vectors . . . 14
3.3 Evaluation Metric 3: CDF of IQ Samples Power . . . 16
3.4 Evaluation Metric 4: Impact of Noise Vector Size . . . 17
3.5 Training & Generation Run-time . . . 19

4 Conclusion & Future Work **21**

5 Acknowledgment **22**

6 Appendices **23**

1 Public POWDER Datasets Description

POWDER [1] (the Platform for Open Wireless Data-driven Experimental Research) [2] is an NSF funded testbed that provides a flexible infrastructure enabling a wide range of software-defined experiments on the future of wireless networks. Powder supports software-programmable experimentation on 5G and beyond, massive MIMO, ORAN, spectrum sharing and CBRS, RF monitoring, and anything else that can be supported on software-defined radios. The major goal of this project is to apply the generative AI/ML models for the insufficient over-the-air (OTA) channel sample datasets that were conducted over POWDER to generate more synthetic OTA channel samples, which provides a more friendly way for POWDER users to acquire a larger synthetic dataset for academic/industrial purpose.

In this project, we use the public datasets [2] from the over-the-air (OTA) channel measurement campaign at POWDER-RENEW using Iris SDRs [1], where the channel measurements were performed at the University of Utah campus, as is shown in Fig. 1. This channel measurement campaign aims to calculate the uplink Signal-to-Noise Ratio (SNR) based on the received pilot signals at the Base station to get an idea of which combination of client locations and base station nodes would yield the best performance (the highest received power). The Uplink SNR is obtained by computing the SNR of each pilot, from each client, at every antenna in the base station, then averaging across pilots and antennas [1].

Figure 1: POWDER Testbeds at the University of Utah campus [1].

1.1 POWDER Scenario Setup

1. A single client device as a transmitter (a single-antenna Iris) is placed at 16 different locations (Red pins in Fig. 1). Data was not collected at locations #12 and #16. Thus the total number of client locations for collected data is 14.

2. 5 "base station" (BS) nodes (Green pins in Fig. 1) as the receivers are located at 5 different rooftops (a single two-antenna Iris on each rooftop). The 5 BSs are named as follows:

   (a) ACC

[1] The link to POWDER: https://www.powderwireless.net/
[2] The public OTA channel measurement dataset that we use can be downloaded at: https://zenodo.org/records/4135078

(b) Honors
(c) MEB
(d) USTAR
(e) EBC

3. The client device (with 1 antenna) continuously transmitted 4000 frames to each BS node (with 2 antennas) in uplink. Therefore, there are a total of $14 \times 5 \times 2 = 140$ links for collected dataset (a link represents a pair of BS antenna and client antenna). At each link, BS node captured 4000 frames (each frame contains with 64 pilots, i.e., 64 Wi-Fi Long Training Sequences [3]), at each of the 14 locations where the client has been stationed. Experiments were performed at 2.5 GHz band [4]. The Sampling rate was 5 MHz [5] (i.e., 5 MHz bandwidth) and the Modulation is QPSK.

1.2 POWDER dataset description

The datasets are stored in HDF5 files containing both metadata and raw IQ samples for each link at which data was collected. The client transmit to BS with the following frame schedule: `GGGGGGGPGGGGGGG` which includes 15 slots. 14 out of 15 slots are guard intervals (G letters) while one slot includes a pilot signal (P letter). The pilot signal includes 64 LTS pilot symbols. Each LTS symbol includes 64 samples. The BS receives the pilot slot from both of its antennas. Each dataset containing 2 sub-datasets for 2 links has the following dimensions: (4000, 1, 1, 2, 8192). 4000 is the number of frames collected in each dataset, 1 is the number of BSs, 1 is the number of client antennas, 2 is the number of BS antennas, and 8192 is the number of I and Q samples in the received samples in each frame. Thus:

- Each frame contains 64 repetitions of LTS.
- FFT size = 64 $\rightarrow$ Each LTS uses 64 subcarriers;
- Total # of IQ samples per frame = 64 LTS per frame $\times$ 64 subcarriers per LTS = 4096 IQ samples per frame;
- The dimension of the provided dataset = $8192 \times 2 \times 1 \times 1 \times 4000$, where $8192 = 2 * 4096$ (2 * indicates I and Q components, and 4096 is total # of IQ samples per frame), 2 is # of BS antennas, 1 is # of BSs, 1 is # of UEs. 4000 is # of frames.

In summary, there are a total of 140 separate POWDER datasets for 140 separate links, while the dimension of the dataset for each link is 8192×4000**, where 4000 is # of frames, and 8192 is the total # of I and Q samples in the received 4096 samples in each frame.**

One caveat is that the client does not perform over-the-air synchronization with the BS in this particular dataset. But rather the client is continuously sending LTS pilots (not following the BS frame schedule) which means the base station will receive a snapshot (4096 samples) from the continuous transmission by the client. The reason for not using OTA synchronization (which is the not default behavior in the sounder software) is because the main goal of this channel measurement campaign was to calculate SNR at each location and not CSI. Thus the synchronization issues during data collection were not addressed. The consequence is that due to the CFO between BS and client there will be a sample drift through time and thus the beginning received sample at each frame will not match the first sample of an LTS pilot. Thus, we propose to apply an AI/ML approach (i.e., Generative Adversarial Network) on the received channel samples to learn all the complex imperfections and then produce a channel model that can characterize the operating conditions with these imperfections.

1.3 Motivation & Contribution

We implemented Generative Adversarial Networks (GAN) on the public POWDER datasets to generate the synthetic dataset. More specifically, 140 separate GANs are needed to train 140 different links' datasets, and each link's dataset includes the channel samples of that link. To generate the synthetic channel samples of each link, after the training process, only the generator (a set of parameters) for each link needs to be stored for POWDER users for future use [6]. Thus, the parameters of a total of 140 generators need to be stored. Afterward, the POWDER users can obtain more synthetic channel samples for a specific link that they want by selecting the corresponding stored generator. The main contributions are summarized as follows:

[3] The sounder software is designed to be configurable to include the desired number of pilot symbols as needed.
[4] The provided dataset is not to replicate WiFi standard though Wi-Fi LTS pilots are used (in a non-standard way) for channel estimation. This is why the used frequency band, sampling rate and the number of LTSs do not align with the Wi-Fi standard.
[5] The 5 MHz rather than 20 MHz bandwidth is used due to some limitations in the client Iris SDR.
[6] Since this project is to generate the synthetic channel samples dataset, the discriminator for each link is not required to be stored.

- The received OTA channel samples in the provided public datasets [1] are affected by complex imperfections without any correction, we propose to apply an AI/ML approach (i.e., Generative Adversarial Network) on the received channel samples to learn all the complex imperfections and then produce a channel model that can characterize the operating conditions with these imperfections.
- The provided public datasets are insufficient, containing only 4000 OTA frames per link dataset. Our proposed GAN-to-POWDER can generate more synthetic OTA frames for each link for POWDER users to use.
- POWDER users have to reserve the limited OTA resources [7] and then configure the experiment profiles over Linux for the OTA data collection, which is typically time-consuming and complicated. Our proposed GAN-to-POWDER can achieve fast and high-fidelity OTA data acquisition without reserving and using the POWDER testbed resources, which benefits POWDER users.

[7] Any POWDER OTA experiments require admin-approved reservations for use. The reservations are reviewed to ensure the reserved frequency ranges are declared appropriately and the reserved software (client and BS Iris SDR) is configured to only use frequencies in the range(s).

2 GAN-to-POWDER

2.1 Generative AI Models for PHY Layer Communications

The recent evolution of generative artificial intelligence (AI)/machine learning (ML) model leads to the emergence of groundbreaking digital content production applications such as ChatGPT. Beyond digital content creation, generative AI/ML model's capability in analyzing complex data distributions offers great potential for wireless communications, particularly amidst a rapid expansion of new physical (PHY) layer communication technologies [12]. In this section, we first investigated the fundamentals of generative AI/ML models for PHY layer communications as well as compare their strengths, weaknesses, and differences [12]. The discussed generative AI/ML models include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Diffusion models, and Normalizing Flows (NFs). Afterward, we discuss the motivation of using GAN to the given POWDER datasets.

2.1.1 Generative AI/ML Models: Recap

Fig. 2 shows the general structures of generative AI/ML models used for PHY layer communications.

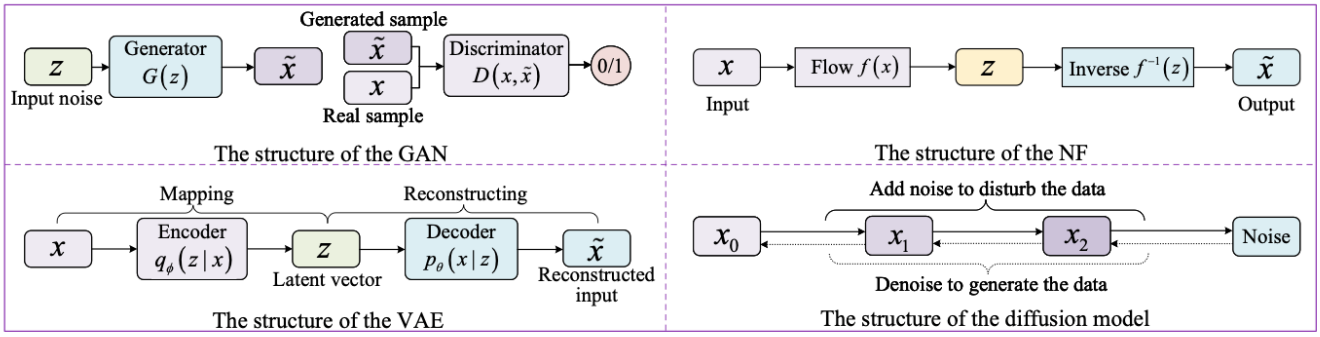


Figure 2: Structures of Generative AI/ML Models

1. **Generative Adversarial Networks (GANs)**: A GAN consists of two main elements, including (i) a generator that produces data mimicking real data and (ii) a discriminator that differentiates between the real and generated data. The training process aims for a Nash equilibrium, where the discriminator cannot differentiate between the two. Trained GANs are capable of reconstructing high-dimensional data from low-dimensional input with fewer generator function restrictions compared to other models, which makes them especially proficient in various issues in the physical layer communications such as channel estimation, channel modeling and CSI compression [15, 16, 7, 9, 5, 10, 12, 17, 6]. Despite these advantages, GANs' training complexity lies in achieving the Nash equilibrium, which is more challenging than optimizing an objective function.

2. **Variational Autoencoders (VAEs)**: VAEs are neural networks designed for compressing and reconstructing data. The VAE comprises an encoder that translates input data into a latent representation, and a decoder that rebuilds the data from this latent space. These components are typically multi-layer neural networks. VAEs optimize their parameters by minimizing a loss function that assesses reconstruction accuracy and aligns the latent space distribution with a prior distribution. Key advantages of VAEs include their ease of implementation and training, effectiveness in learning compressed data representations, and a probabilistic nature that allows for uncertainty estimation and varied outputs. As a result, VAEs are particularly effective in capturing the dynamics and uncertainty of wireless communications evidenced by a wide range of applications in channel estimation, channel modeling, and signal classification [18, 13]. However, they present challenges in training and parameter tuning, with the possibility of non-interpretable compressed representations.

3. **Normalizing Flows (NFs)**: NFs are generative models that transform simple probability distributions into complex ones using reversible transformations. The advantages of NFs lie in efficiently sampling complex distributions, managing high-dimensional data, and learning interpretable latent spaces. This is particularly useful for various tasks in the physical layer communications [4]. However, the challenges include high computational

demands, lengthy training for complex distributions, and transformation function selection. Hence, recent studies have explored optimizing architectures and training efficiency through techniques such as adversarial training and regularization, demonstrating NFs' potential in diverse applications.

4. **Diffusion Models**: Unlike the aforementioned generative AI/ML models, diffusion models start with adding noise to training samples, which is known as the forward diffusion process, and then remove the noise to generate new samples in the inverse process. They can be trained on incomplete data in a stable process. This special capability makes them highly suitable for equalizing and modeling wireless channels with limited training data and noisy conditions [11, 14]. However, diffusion models face challenges such as longer sampling times, complex training architectures, and limitations with certain data types [12].

2.1.2 Motivation of Choosing GAN for Channel Modeling

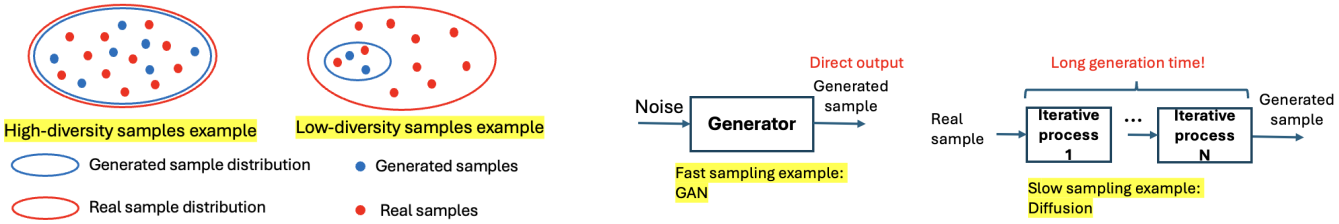
We compare the four generative AI/ML models in terms of following metrics shown in Table 1,

Table 1: Generative AI/ML Models comparison

Models	High-quality samples	High-diversity samples	High-training stability	Fast-sampling
GANs	✓	✓	×	✓
VAEs	×	✓	✓	✓
Diffusion Models	✓	✓	✓	×
NFs	✓	✓	✓	×

where the four metrics are

- **High-quality samples**: Generated samples can well represent the features of real samples. (i.e., small feature distance)
- **High-diversity samples**: Generated samples can sufficiently capture the whole real sample distribution, which is illustrated in Fig. 3(a).
- **High-training stability**: Training loss converges after enough training iterations.
- **Fast sampling**: Quick sample generation directly from the generative model, which is illustrated in Fig. 3(b).



(a) High and Low diversity samples difference

(b) Fast and Slow sampling difference

Figure 3: Two Metrics Comparison.

In general, the purpose of applying generative AI models on channel modeling is to produce the synthetic channel data characterized with high-fidelity, high-diversity, and fast acquisition features. Thus GAN can exactly support these all features while other generative AI models do not. It should be noted that though GANs do not characterize the high-training stability feature (i.e., In GANs, it is hard to find the global Nash Equilibrium and the training loss may not converge during the training stage. Thus, there is no specific stopping rule for training.), the suboptimal Nash Equilibriums can be always reached so that both generator and discriminator are good enough to generate/discriminate after enough training iterations. That is why most existing work [15, 16, 7, 9, 5, 10, 12, 17, 6] adopted GANs rather than other approaches on channel modeling.

2.2 Fundamentals of GANs

Suppose we have a Real dataset, the idea is to sample from a simple tractable distribution (say, $z \sim \mathcal{N}(0, I)$) and then learn a complex transformation from noisy data to the real data distribution.

What can we use for such a transformation? A Neural Network. How do we train such a network? By using a two-player game; namely a generator and discriminator.

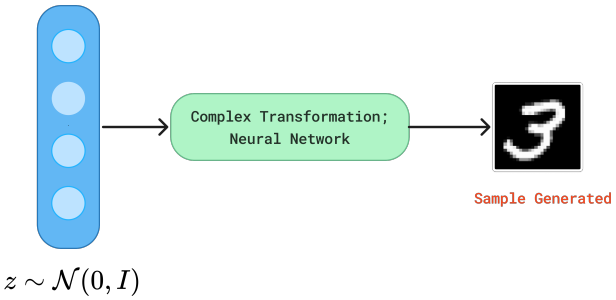
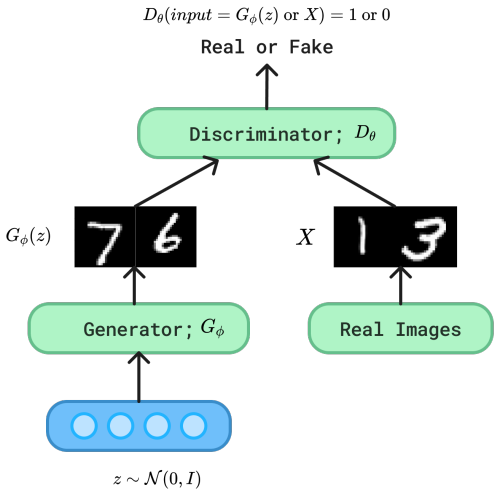


Figure 4: Overall idea for GAN

2.2.1 Components of GAN

- The job of the generator is to produce images/data which look so natural that the discriminator thinks the images came from the real data distribution
- The job of the discriminator is to get better and better at distinguishing between true images/data and generated (fake) images/data.



- Let G_ϕ be the generator and D_θ bet the discriminator where ϕ, θ are the parameters of G and D respectively.
- We have neural network based generator which takes as input; a noise vector $z \sim \mathcal{N}(0, I)$ and produces $G_\phi(z) = X$
- We have a neural network based discriminator which could take as input ($input = X$ or $G_\phi(z)$) and classify the input as real or fake ($D_\theta(input) = 1$ or 0)

Figure 5: Overall idea for GAN

2.2.2 Objective Functions of GAN

Generator:

The **generator** would want to maximize $\log D_\theta(G_\phi(z))$ (log likelihood) or minimize $\log(1 - D_\theta(G_\phi(z))$

Case 1: If z is discrete and drawn from uniform distribution (i.e. $p(z) = \frac{1}{N} \forall z$)

$$\min_\phi \sum_{i=1}^{N} \frac{1}{N} \log(1 - D_\theta(G_\phi(z))$$

Case 2: If z is continuous and not uniform ($z \sim \mathcal{N}(0, I)$)

$$\min_\phi \int p(z) \log(1 - D_\theta(G_\phi(z))$$

$$\min_\phi \mathbb{E}_{z \sim p(z)}[\log(1 - D_\theta(G_\phi(z))]$$

Discriminator:

The discriminator should assign a high score to real images and a low score to fake images.

$$\max_\theta \mathbb{E}_{x \sim p(data)}[\log(D_\theta(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\theta(G_\phi(z))]$$

GAN:

Putting together both the generator's and discriminator's objective function, we get the objective function for the overall GAN or setup.

$$\min_\phi \max_\theta \mathbb{E}_{x \sim p(data)}[\log(D_\theta(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\theta(G_\phi(z))]$$

The first term in objective is only w.r.t the parameters of the discriminator (θ). The second term in the objective is w.r.t the parameters of the generator (ϕ) as well as the discriminator (θ)

The discriminator wants to maximize the second term whereas the generator wants to minimize it (hence, it is a two-player game).

2.3 Proposed GAN-to-POWDER Implementation

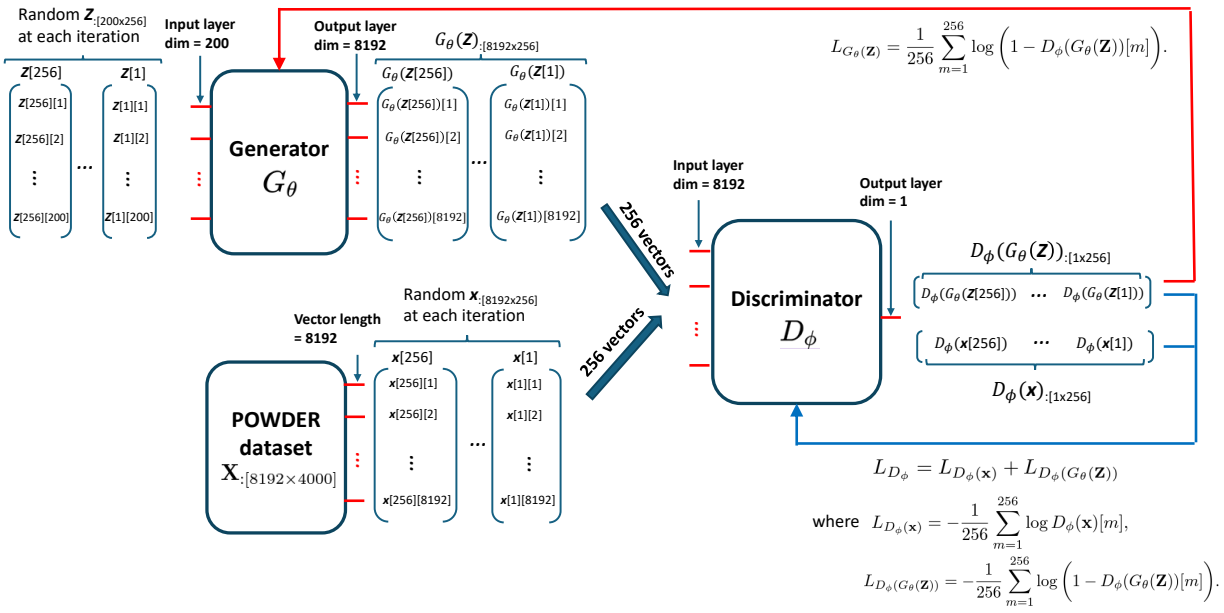


Figure 6: GAN Implementation architecture (at each iteration) for POWDER dataset

where the proposed Neural network structure of the generator and discriminator are shown in the following tables.

Generator Architecture		
Layer #	**Layer Components**	**# of nodes**
Layer 0	Input Layer	200
Hidden Layer 1	Dense : 2048	2048
	Batch Normalization [a] [1]	2048
	ReLU ($\alpha = 0.2$) [2]	2048
Hidden Layer 2	Dense : 4096	4096
	Batch Normalization	4096
	ReLU ($\alpha = 0.2$)	4096
Hidden Layer 3	Dense : 4096	4096
	Batch Normalization	4096
	ReLU ($\alpha = 0.2$)	4096
Hidden Layer 4	Dense : 2048	2048
	Batch Normalization	2048
	ReLU ($\alpha = 0.2$)	2048
Layer 5	Output Layer	8192

Table 2: Generator Architecture

[a]Batch-Normalization is normalization across different vectors in a batch to enforce the data distribution to have 0 mean and 1 variance for faster convergence of training.

Discriminator Architecture		
Layer #	**Layers Components**	**# of Nodes**
Layer 0	Input Layer	8192
Hidden Layer 1	Dense : 2048	2048
	ReLU [a] ($\alpha = 0.2$)	2048
Hidden Layer 2	Dense : 4096	4096
	ReLU ($\alpha = 0.2$)	4096
Hidden Layer 3	Dense : 2048	2048
	ReLU ($\alpha = 0.2$)	2048
Hidden Layer 4	Dense: 1	1
Layer 5	Output Layer (Sigmoid)	1

Table 3: Discriminator Architecture

[a]ReLU is an activation function. For a input value x, $\text{ReLU}(\alpha)[x] = \max(\alpha x, x)$.

The training procedure of GAN-to-POWDER is illustrated in Algorithm 1 in detail. In each iteration, the generator and the discriminator are trained simultaneously. The parameters of the generator and the discriminator are updated according to the loss function of equations (1) and (4), respectively.

Algorithm 1 GAN Implementation for POWDER dataset

A POWDER dataset per link $\mathbf{X}_{:[8192\times 4000]}$ contains 4000 OTA transmission frames, where each transmission frame is a vector containing 8192 channel samples.

Initialize (Iteration 0): A Generator G_{θ_0} parameterized by θ_0 and A Discriminator D_{ϕ_0} parameterized by ϕ_0.

Input (Input layer dims = 200) to Generator G_{θ_n}: $\mathbf{Z}_{n:[200\times 256]}$ containing 256 noise vectors, where each vector contains 200 i.i.d. Gaussian random variables following $\mathcal{N}(\mathbf{0}_{200\times 1}, \mathbf{I}_{200\times 200})$, which corresponds to 100 complex Gaussian noises.

Output (Output layer dims = 8192) to Generator G_{θ_n}: $G_{\theta_n}(\mathbf{Z}_n)_{:[8192\times 256]}$ containing 256 generated vectors, where each vector contains 8192 generated channel samples.

Input (Input layer dims = 8192) to Discriminator D_{ϕ_n}: Equal number (batch size $m = 256$) of vectors from the generator G_{θ_n} and the POWDER dataset $\mathbf{X}_{:[8192\times 4000]}$, i.e., $G_{\theta_n}(\mathbf{Z}_n)_{:[8192\times 256]}$ and $\mathbf{x}_{n:[8192\times 256]}$ randomly sampled from $\mathbf{X}_{:[8192\times 4000]}$.

Output (Output layer dims = 1) to Discriminator D_{ϕ_n}: $D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))_{:[1\times 256]}$ and $D_{\phi_n}(\mathbf{x}_n)_{:[1\times 256]}$ include 512 scores $\in (0, 1)$ discriminating where input vectors are from. (Note: If a high score > 0.5, the input vector is from $\mathbf{X}$, otherwise, the input vector is from G_{θ_n}).

for Training iteration $n = 1$ to 100,000; **do**

Sample a batch $\mathbf{Z}_{n:[200\times 256]}$;

Compute generated vectors $G_{\theta_n}(\mathbf{Z}_n)_{:[8192\times 256]}$;

Sample a batch $\mathbf{x}_{n:[8192\times 256]}$ from $\mathbf{X}_{:[8192\times 4000]}$;

Compute $D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))_{:[1\times 256]}$ and $D_{\phi_n}(\mathbf{x}_n)_{:[1\times 256]}$;

Compute Discriminator Loss $L_{D_{\phi_n}}$:

$$L_{D_{\phi_n}} = L_{D_{\phi_n}(\mathbf{x}_n)} + L_{D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))}, \text{where} \tag{1}$$

$$L_{D_{\phi_n}(\mathbf{x}_n)} = -\frac{1}{256}\sum_{m=1}^{256} \log D_{\phi_n}(\mathbf{x}_n)[m], \tag{2}$$

$$L_{D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))} = -\frac{1}{256}\sum_{m=1}^{256} \log\Big(1 - D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m]\Big). \tag{3}$$

Update discriminator parameters D_{ϕ_n} by minimizing $L_{D_{\phi_n}}$ using ADAM optimizer;

Compute Generator Loss $L_{G_{\theta_n}}$:

$$L_{G_{\theta_n}(\mathbf{Z}_n)} = -\frac{1}{256}\sum_{m=1}^{256} \log\Big(D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m]\Big). \tag{4}$$

Update generator parameters G_{θ_n} by minimizing $L_{G_{\theta_n}}$ using ADAM optimizer;

end for

where the key notations in Algorithm 1 are shown in the following table.

Table 4: Glossary for Algorithm 1

Notations	Definition
$\mathbf{X}_{:[8192\times4000]}$	4000 OTA frames containing 8192 IQ samples reshaped into 4000 vectors of 8192 IQ samples
$\mathbf{Z}_{n:[200\times256]}$	256 noise vectors where each sampled following $\mathcal{N}(\mathbf{0}_{1\times200}, \mathbf{I}_{200\times200})$
$\mathbf{x}_{n:[8192\times256]}$	256 vectors where each sampled randomly from $\mathbf{X}_{:[8192\times4000]}$ for training at nth iteration
m	Batch size = 256
G_{θ_0}	Generator G with parameters θ_0 initialized at iteration 0
D_{ϕ_0}	Discriminator D with parameters ϕ_0 initialized at iteration 0
G_{θ_n}	Generator G with parameters θ_n at iteration n
D_{ϕ_n}	Discriminator D with parameters ϕ_n at iteration n
θ_0	Parameters of G at iteration 0
ϕ_0	Parameters of D at iteration 0
θ_n	Parameters of G at iteration n
ϕ_n	Parameters of D at iteration n
n	Training Iteration from $\{0, \ldots, 100000\}$
$G_{\theta_n}(\mathbf{Z}_n)_{:[8192\times256]}$	Generated vectors at iteration n
$D_{\phi_n}(\mathbf{x}_n)_{:[1\times256]}$	Discriminator scores for Real vectors at iteration n
$D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))_{:[1\times256]}$	Discriminator scores for Generated vectors at iteration n
$D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m]$	Discriminator score for the m^{th} generated vector from $G_{\theta_n}(\mathbf{Z}_n)_{:[8192\times256]}$
$D_{\phi_n}(G_{\theta_n}(x_n))[m]$	Discriminator score for the m^{th} vector from $\mathbf{x}_{n:[8192\times256]}$
$L_{D_{\phi_n}}$	Loss function of D with parameters ϕ_n at n^{th} iteration
$L_{G_{\theta_n}}$	Loss function of G with parameters θ_n at n^{th} iteration
$L_{D_{\phi_n}(\mathbf{x}_n)}$	Discriminator loss for vectors from $\mathbf{X}$ at iteration n
$L_{D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))}$	Discriminator loss for generated vectors at iteration n
$L_{G_{\theta_n}(\mathbf{Z}_n)}$	Generator loss for generated vectors at iteration n

2.4 A Recap of ADAM (Adaptive Moment Estimation)

In Algorithm 1, the ADAM optimizer [8] is used to compute the Gradient descent to minimize the loss function of the generator and discriminator. Here, we provide a recap of ADAM fundamentals. The initialization is followed from the ADAM paper as referenced and not much analysis or papers are provide on the impact of Hyper-parameters on GAN or Neural Network Performance or on Hyper-parameter tuning. Initialization of (α) learning rate is referred from [15] as they authors were also using IQ data.

Initialize parameters: $m_0 = 0$, $v_0 = 0$, $n = 0, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \alpha = 0.0002$

For each iteration t:

$$g_{n+1} = \nabla_\theta f(\theta_t)$$
$$m_{n+1} = \beta_1 m_t + (1-\beta_1) g_{n+1}$$
$$v_{n+1} = \beta_2 v_t + (1-\beta_2) g_{n+1}^2$$
$$\hat{m}_{n+1} = \frac{m_{t+1}}{1-\beta_1^{n+1}}$$
$$\hat{v}_{n+1} = \frac{v_{t+1}}{1-\beta_2^{n+1}}$$
$$\theta_{n+1} = \theta_n - \frac{\alpha \hat{m}_{n+1}}{\sqrt{\hat{v}_{n+1}} + \epsilon}$$

- $g_{n+1} = \nabla_\theta f(\theta_n)$: The gradient of the objective function f with respect to the parameters θ at iteration n. This represents the gradient of the loss function at the current iteration.
- In POWDER case, f for the Dicriminator is $L_{D_{\phi_n}}$ and for the Generator is $L_{G_{\theta_n}}$

- m_{n+1}: The first moment estimate, which is an exponentially decaying average of past gradients (similar to momentum). It is calculated as:
$$m_{n+1} = \beta_1 m_n + (1 - \beta_1) g_{n+1}$$
where β_1 is a hyperparameter that controls the decay rate of the moving average of the gradients.

- v_{n+1}: The second moment estimate, which is an exponentially decaying average of past squared gradients. It is calculated as:
$$v_{n+1} = \beta_2 v_n + (1 - \beta_2) g_{n+1}^2$$
where β_2 is another hyperparameter that controls the decay rate of the moving average of the squared gradients. This term helps to adaptively scale the learning rate for each parameter.

- $\hat{m}_{n+1}$: The bias-corrected first moment estimate. Since m_n is initialized at zero, it is biased toward zero in the initial steps. This bias is corrected as follows:
$$\hat{m}_{n+1} = \frac{m_{n+1}}{1 - \beta_1^{n+1}}$$

- $\hat{v}_{n+1}$: The bias-corrected second moment estimate. Similar to m_t, v_t is also biased toward zero initially, and its bias is corrected by:
$$\hat{v}_{n+1} = \frac{v_{n+1}}{1 - \beta_2^{n+1}}$$

- θ_{n+1}: The parameter update rule, which adjusts the model parameters θ based on the learning rate α and the bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$:
$$\theta_{n+1} = \theta_n - \frac{\alpha \hat{m}_{n+1}}{\sqrt{\hat{v}_{n+1}} + \epsilon}$$
where α is the learning rate, and ϵ is a small constant added to avoid division by zero.

3 Performance Evaluation

The performance evaluation is conducted on Google Colab [8] with Python 3 Google Compute Engine backend (TPU) comprising 220 compute units. The **Algorithm 1** in Section 2 was implemented in Python using Tensorflow [9]. **We select a POWDER dataset for the link - Location 8 to the 1st antenna of BS "EBC" as the real dataset used for GAN training**, where Location 8 and BS "EBC" are shown in Fig. 1. Note that there are a total of $14 \times 5 \times 2 = 140$ links (14 client locations, 5 BS locations, and 2 antennas per BS) for the collected dataset. We need to train 140 separate GANs for the 140 collected datasets. The proposed GAN algorithm can be applied to all 140 datasets. Here, we evaluate the GAN performance based on one specific link dataset.

3.1 Evaluation Metric 1: Loss Function of Generator and Discriminator

As Fig. 7 shows, we have run the GAN till 100000 iterations, to check how the Loss functions for the Generator and Discriminator behave. We can see that the generator and discriminator loss values follow the general trend that it is decreasing in terms of the larger iterations, however, there are a lot of loss peaks during the training. This is testament to the fact that discriminator and generator plays a game of cat and mouse (overcoming each other in such iterations). Therefore, there is no specific stopping rule during the GAN training as it is hard to find the Nash equilibrium [3] by simultaneously training two neural networks against each other [10].

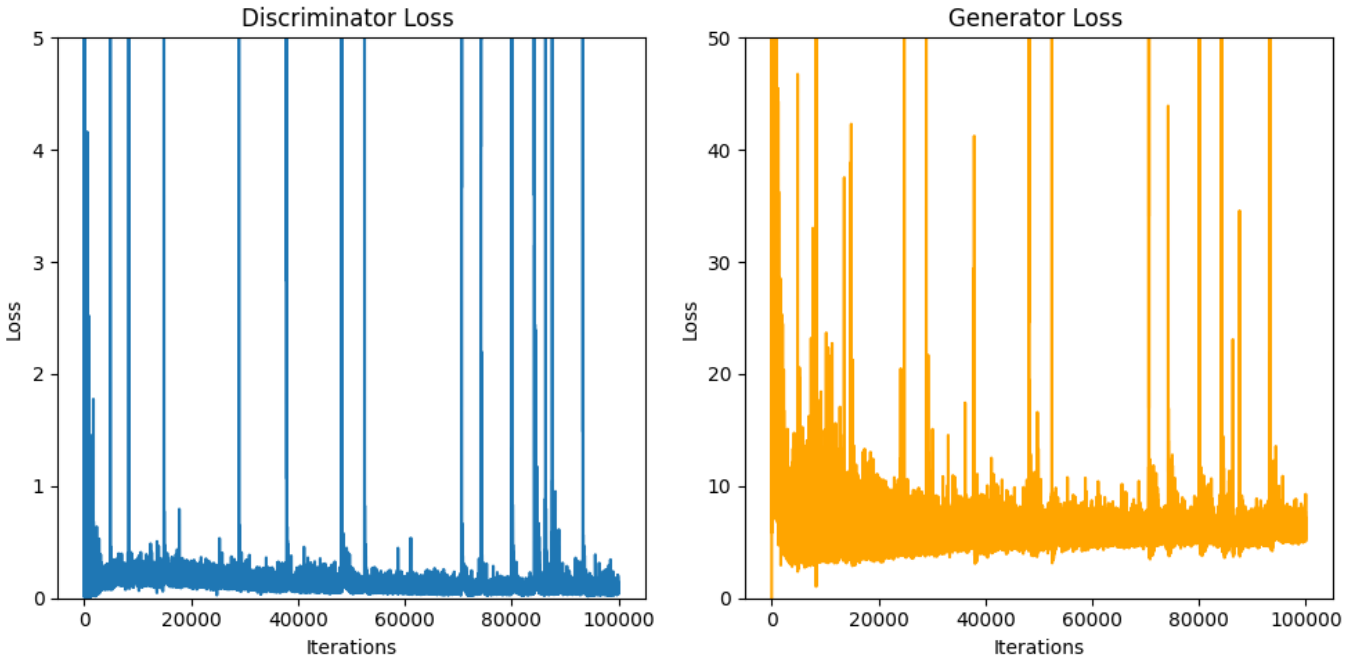


Figure 7: Training Loss values in terms of iterations

3.1.1 Discriminator Training Loss

For the Discriminator, these peaks (sudden increase in Loss values) are attributed to that Discriminator is not able to discriminate the real or generated vectors properly, such as from

[8] Google Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. https://colab.research.google.com/

[9] All code used to generate the results in this report can be found at https://github.com/liucaouw/GAN-to-POWDER

[10] This is different from the one neural network case where only one loss function is minimized so that the Nash equilibrium can be easily found. In GAN, both neural networks try to minimize their loss function, however, while one neural network decreases its loss function, the loss function of the other neural network will increase, which results in an unstable training process, that is, it is hard to find Nash equilibrium during the training stage.

$$L_{D_{\phi_n}} = L_{D_{\phi_n}(\mathbf{x}_n)} + L_{D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))}, \text{where}$$
$$L_{D_{\phi_n}(\mathbf{x}_n)} = -\frac{1}{256}\sum_{m=1}^{256} \log D_{\phi_n}(\mathbf{x}_n)[m],$$
$$L_{D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))} = -\frac{1}{256}\sum_{m=1}^{256} \log\Big(1 - D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m]\Big).$$

In $L_{D_{\phi_n}}$, for real vectors, the discriminator is discriminating or recognizing as Fake which from the fact that $D_{\phi_n} \to 0$, where D_{ϕ_n} is a probability score whether the vector being real. This makes $-\log D_{\phi_n}(\mathbf{x}_n)$ blow up to very large values.

On the other hand, the discriminator is recognizing the generated vectors as Real which makes $\log(1-D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m])$ blow up with $D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n)) \to 0$, suggesting the Generator is has become too strong, which might be what we want, but there is no way of saying here, whether the generator has generated good vectors or the discriminator is bad at discriminating. Due to these factors, $L_{D_{\phi_n}}$ blows up to larger values.

3.1.2 Generator Training Loss

For the Generator, the increase in Loss values, can be due to the fact Generator is making the same mistake in generating its vectors, which makes the Discriminator discriminate the generated vectors as Fake, viz., $D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n)) \to 0$ making $L_{G_{\theta_n}(\mathbf{Z}_n)}$ larger.

$$L_{G_{\theta_n}(\mathbf{Z}_n)} = -\frac{1}{256}\sum_{m=1}^{256} \log\Big(D_{\phi_n}(G_{\theta_n}(\mathbf{Z}_n))[m]\Big).$$

In summary, from the Discriminator and Generator Loss curve, we can see that both losses have significantly decreased after around **30000 iterations**, and around **55000 to 70000 iterations** seems to be our **our region of interest regarding there are no sudden peaks**, however, only relying on that is not sufficient.

3.2 Evaluation Metric 2: Average power and variance of the vectors

To evaluate the GAN performance, we compare the similarity between the generated vectors and real vectors based on some metrics. One typical metric that evaluates the generative model in PHY layer is the average power of packet (frame). Therefore, we use the same metric to see how good the generated vectors are against the real vectors.

We generate 4000 vectors that align with the POWDER dataset size (4000 vectors) per link. Then we randomly sample 1 vector from the Generated and Real vectors at the initial iteration and converged iteration for comparison. So, for evaluation, we are taking 4000 real and 4000 generated vectors, because 4000 is the # of IQ vectors taken from the hdf5 file for the POWDER data for 1 link. We are calculating **Average Power of the vectors** and **Variance of the Powers of the vectors** present for using the below formula:

$$P(l,v,iq) = I_{iq}^2 + Q_{iq}^2$$
$$P(l,v) = \frac{1}{N_{iq}}\sum_{iq=0}^{N_{iq}=4096} P(l,v,iq)$$
$$P_{avg}(l) = \frac{1}{N_v}\sum_{v=1}^{N_v=4000} P(l,v)$$
$$P_{variance}(l) = \frac{1}{N_v}\sum_{v=1}^{N_v=4000} (P(l,v) - P_{avg}(l))^2$$

where P and P' refer to the power of real and generated vectors, respectively. I_{iq}, Q_{iq} are I and Q values of the iq^{th} sample out of 4096 samples in the frame f for link L. $P(l,v,iq)$ is Power of each iq sample in vector v for Link l.$P(l,v)$ is Power of each vector v in link l. $P_{avg}(l)$ is Average Power of the vectors in Link l. $P_{variance}(l)$ is Variance

of the Power of the frames in Link l. $N_{iq} = 4096$ is Number of iq samples. $N_v = 4000$ is Number of POWDER vectors v.

Fig 8 shows how the Average Powers and Variance of Powers of the Generated vectors vary with iterations (values are plotted every 5000 iterations). and we can find show on increasing iteration we reach convergence for Powers of the Generated vectors with respect to the real Vectors.

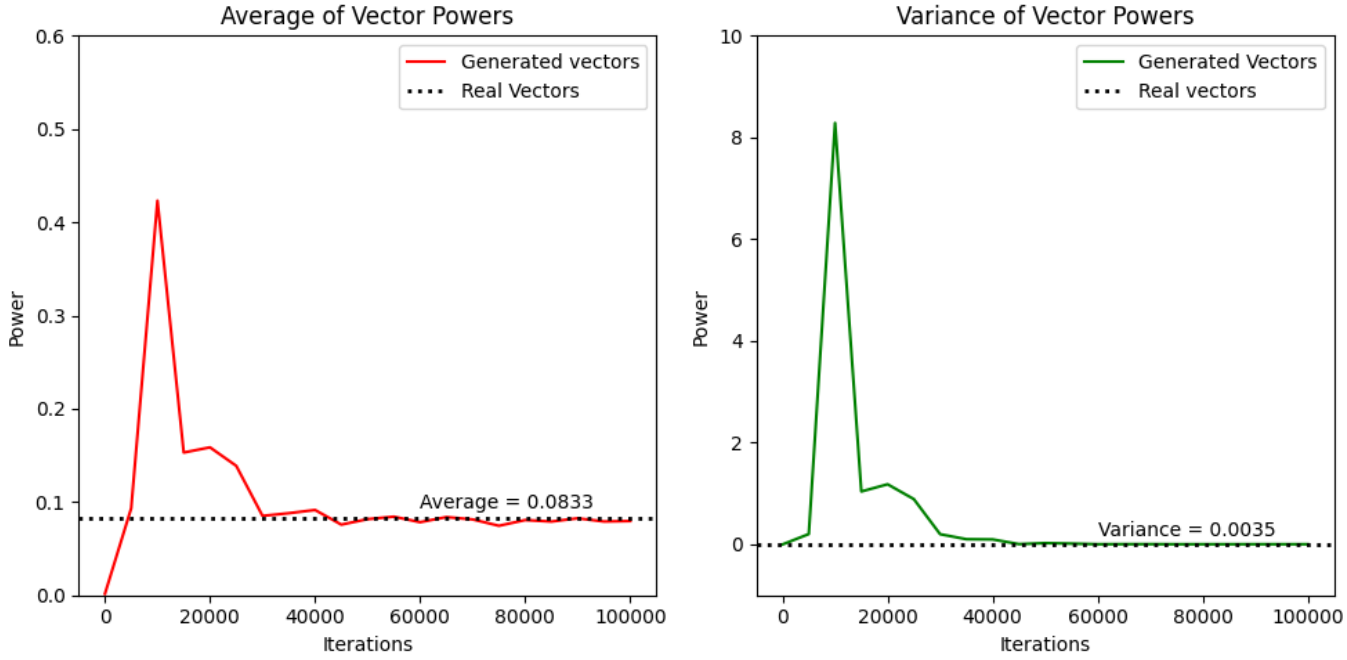


Figure 8: Average and Variance of the Powers of the Vectors

Based on the Average and Variance of Power Figures, we see the Average Power and Variance of Powers of the Generated vectors are **matching with the real vectors at 60000 iteration at the earliest**, matching with the region of interest from the Loss curves. **Hence, we choose 60000 iterations as our converged point or stopping rule.**

Iteration #	**Average Power for Real vectors**: $P_{avg}(l)$	**Average Power for Generated vectors**: $P'_{avg}(l)$
1	0.0833	0.00156
60000 (Convergence)	0.0833	0.0821

Iteration #	**Variance of the Power for Real vectors**: $P_{variance}(l)$	**Variance of the Power for Generated vectors**: $P'_{variance}(l)$
1	0.0035	$2.5248 * 10^{-6}$
60000 (Convergence)	0.0035	0.0040

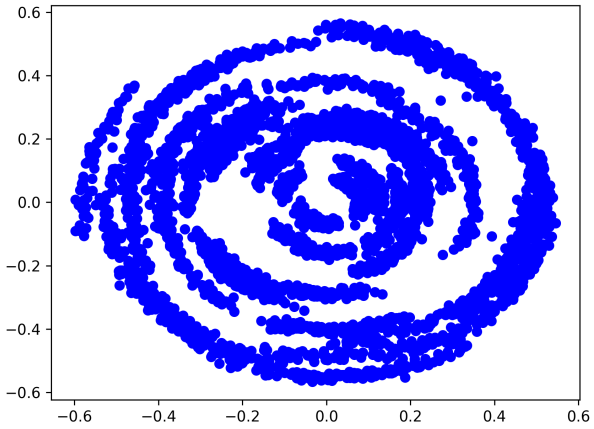
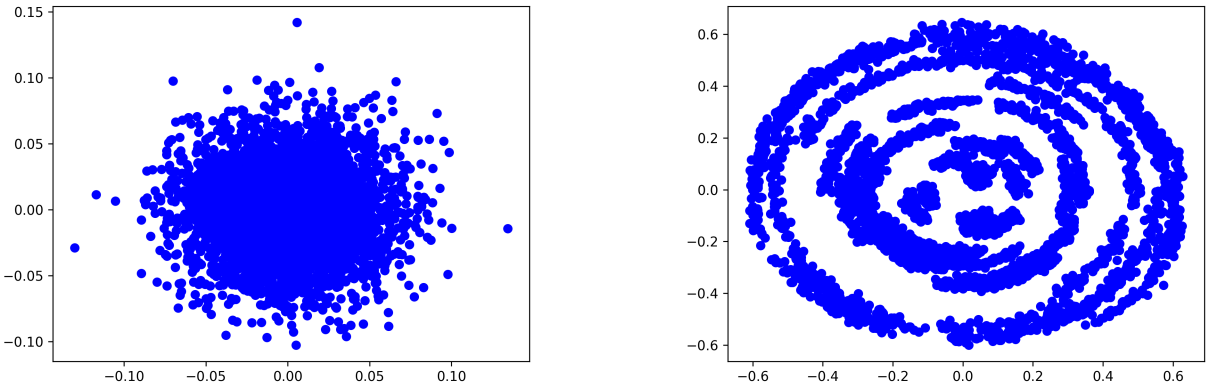


Figure 9: Visualization of a real vector (containing 4096 real IQ samples).



(a) A generated vector (Output of Generator) at the initial iteration

(b) A generated vector (Output of Generator) at the converged iteration = 60000

Figure 10: Visualization of a generated vector (containing 4096 generated IQ samples)

3.3 Evaluation Metric 3: CDF of IQ Samples Power

As Fig. shows, Here, we use the Two-sample Kolmogorov–Smirnov (KS) test [11] to check the similarity between the two underlying CDF curves. The KS statistic $D_{n,m}$ is

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)| = 0.0216,$$

where $F_{1,n}$ and $F_{2,m}$ are the *empirical cumulative distribution functions* of the real dataset and the generated dataset, respectively. n is the real dataset size and m is the generated dataset size, note that $n = m = 4000$. *sup* is the *supremum function*. The KS statistic can be evaluated against a *critical value* D_α to assess if two empirical CDF curves are significantly different, which is given by:

$$D_\alpha = \sqrt{-\frac{1}{2}\ln\left(\frac{\alpha}{2}\right)} \times \sqrt{\frac{n+m}{n \cdot m}} = 0.0304,$$

[11] The Two-sample Kolmogorov–Smirnov statistic quantifies a distance between the empirical cumulative distribution functions (ECDF) of two dataset samples.

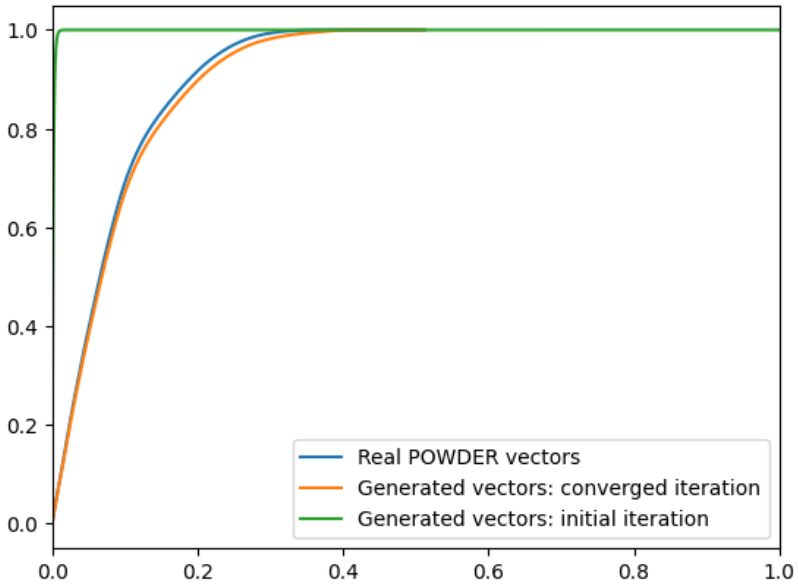


Figure 11: CDF of IQ Samples Power.

where α is the significance level [12], and we set $\alpha = 0.05$.

As $D_{n,m} < D_\alpha$, the null hypothesis that the real dataset and the generated dataset come from the same distribution is true.

3.4 Evaluation Metric 4: Impact of Noise Vector Size

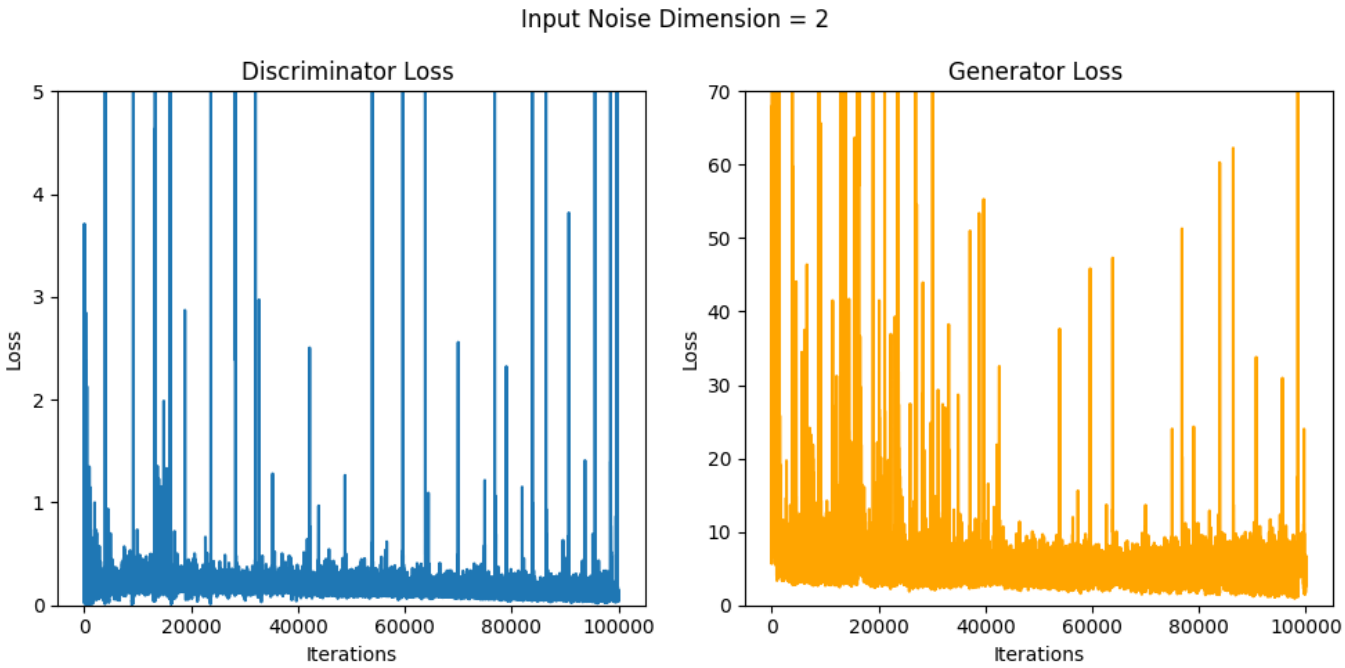


Figure 12: Training Loss values in terms of iterations for Input Noise Dimension = 2

[12] For large samples, the null hypothesis that the two dataset samples come from the same distribution is rejected at level α if $D_{n,m} > D_\alpha$.

From the above image, we see that even though the losses are loosely decreasing in Generator (worse for Discriminator), there are more unwanted Loss value peaks in this case in comparison to Input Noise dimension = 200, and we can barely see a stable convergence **region to be around 45000 - 50000 iterations**. But the Power and Variance images say otherwise. The variance of the Generated vectors are converging as required around the iterations of interests, but the **Average power of the vectors could not converge to required value of 0.0833** even after running for 100000 iterations.

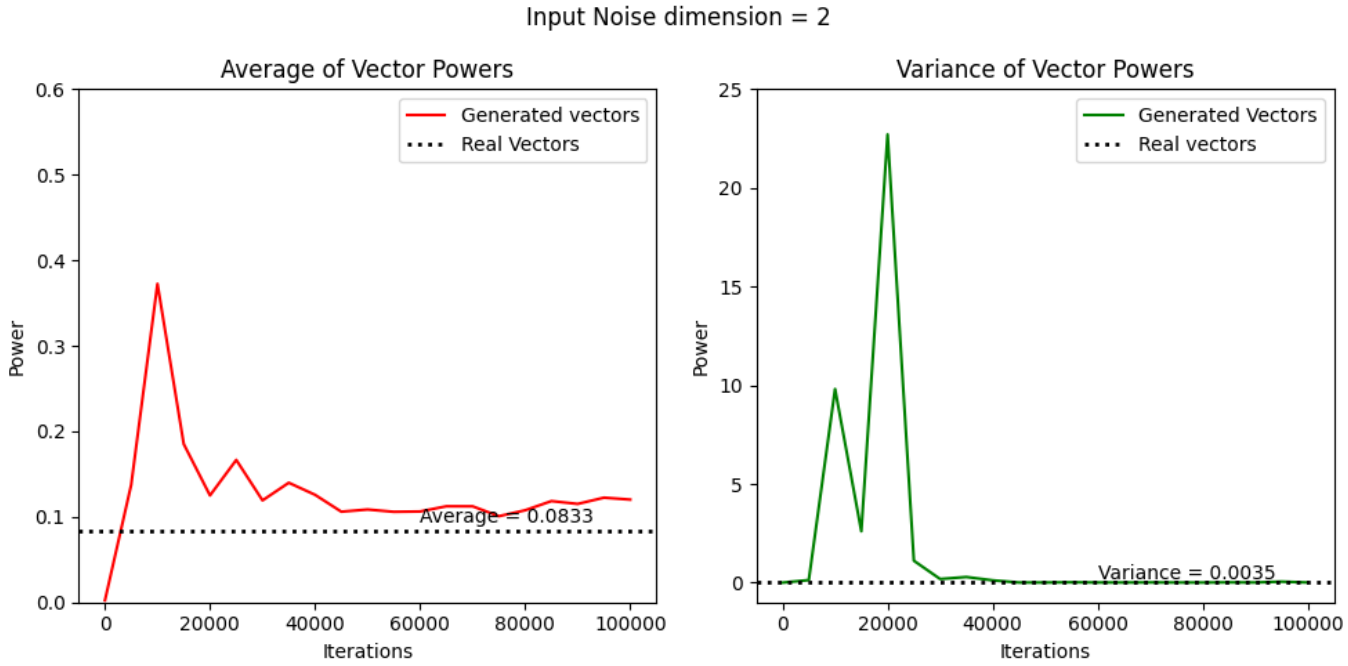


Figure 13: Average and Variance of the Powers of the Vectors for Input Noise Dimension = 1000

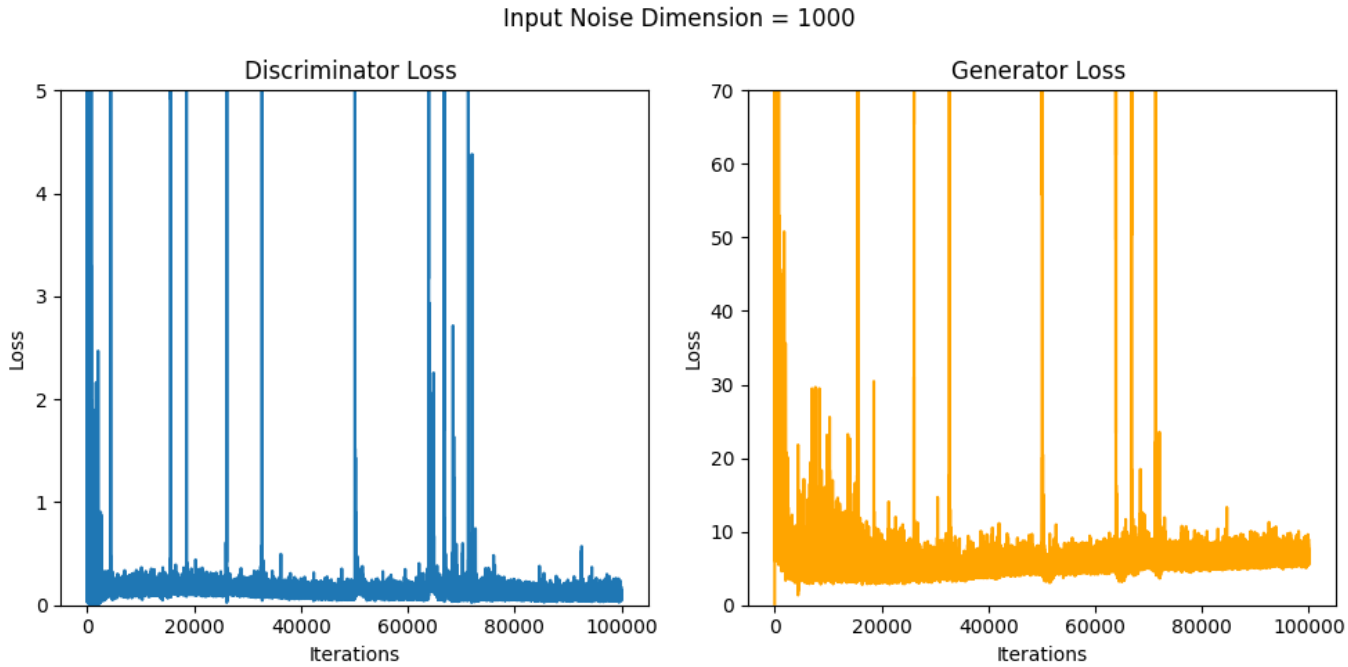


Figure 14: Training Loss values in terms of iterations for Input Noise Dimension = 1000

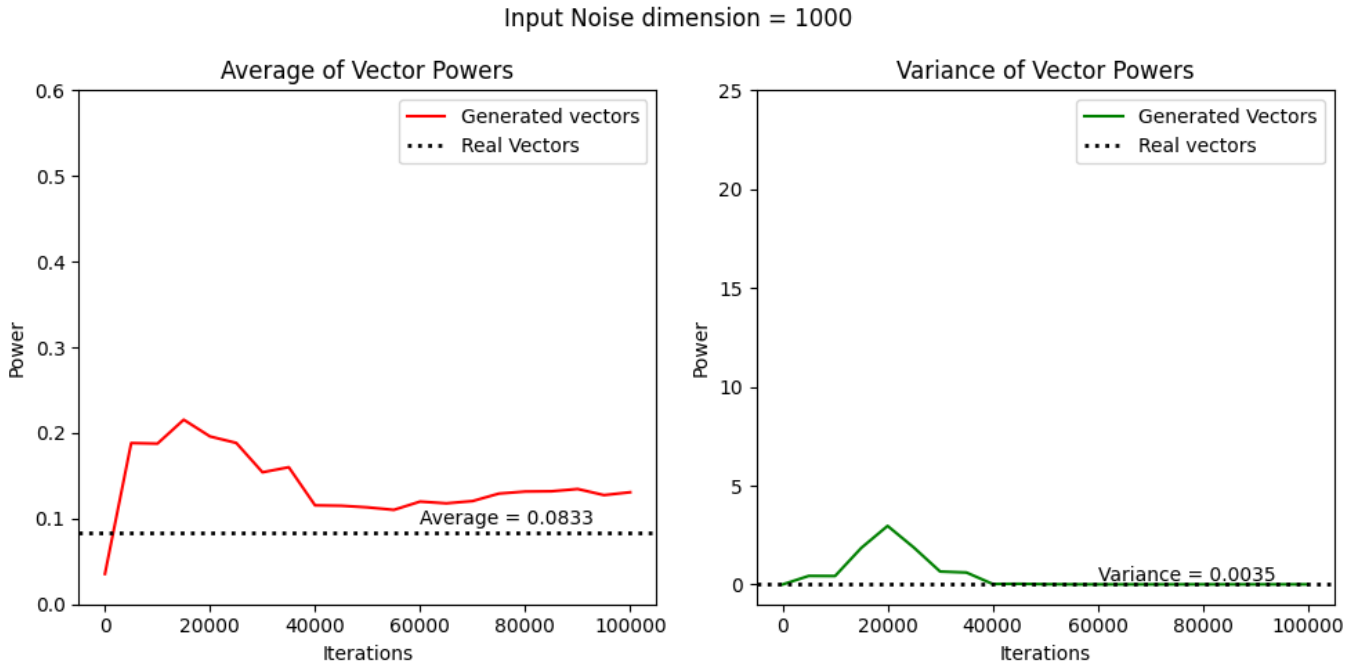
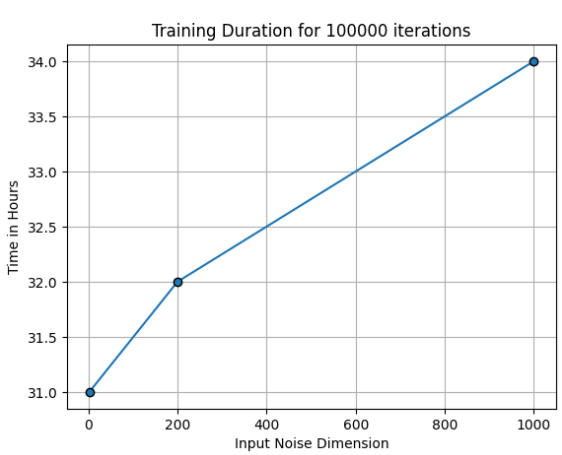


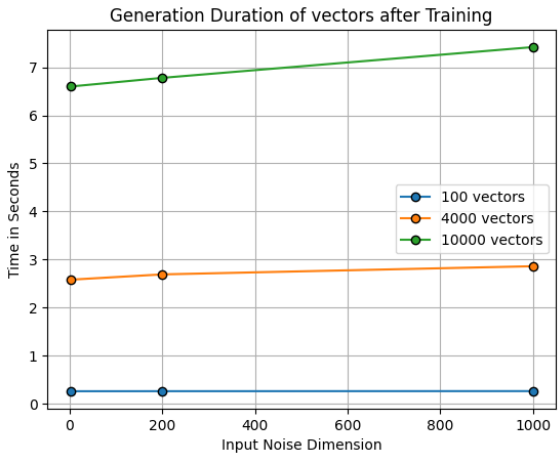
Figure 15: Average and Variance of the Powers of the Vectors for Input Noise Dimension = 1000

Now, when the input noise dimension is 1000, we see that the Loss curve is better when we look at them visually, as there are no various peaks (sudden increases in loss values than when the input noise dimension is 2, 200, but the average Power of the IQ vectors is not converging to the Real vectors' average Power value, even when run for 100000 iterations, stating that taking **input noise dimension as 200, is relatively the best choice** in developing our GAN model

3.5 Training & Generation Run-time



(a) Training Run-time with Different Input Noise size.



(b) Generation Run-time with Different Input Noise Size & Number of Vectors

The training time for the GAN model is slowly increasing when the Input noise dimension is increasing, this is due to the reason that on increasing the dimension of the input noise, we are increasing the number of nodes in the input layer, which can then increase the number of parameters involved on training the model and that's why as validated by Table 4, we see the Training time is increasing for increasing input noise dimension.

Noise Input Dimension	Time (Hours)
2	31
200	32
1000	34

Table 5: Impact of Run-time on Input Noise Dimension during Training for 100000 iterations

Noise Input Dimension	Time (Seconds)		
	100 vectors	4000 vectors	100000 vectors
2	0.2588	0.2597	0.2611
200	2.58	2.69	2.86
1000	6.60	6.78	7.42

Table 6: Impact of Run-time on POWDER vector generation after Training with respect to sample size (No. of vectors)

For Generating vectors after training, the generation time is also increasing for if we increase the dimension of the input noise, but the increase in time is prevalent when we generate more number of vectors. In our experimentation, when we generated 100 vectors, the time taken by different GAN models of different noise input dimensions are increasing but it seems around the same, which can be seen in Table 5. Then, the increase in Generation time for 4000, 100000 vectors are more prevalent for increasing Input noise dimensions. This is because, we are running the model several times to generate more vectors, this takes longer time and hence the increase in time for generating more vectors.

4 Conclusion & Future Work

In this project, we design a generative AI/ML model that can characterize one specific OTA channel measurement campaign based on the POWDER testbed. As a result, POWDER users do not have to conduct the time-consuming and complicated OTA data collection via POWDER testbed. Our proposed generative AI/ML model for POWDER can achieve fast and high-fidelity OTA data acquisition without reserving and using POWDER testbed resources, which benefits POWDER users.

For future work, we will implement the conditional GAN to reduce the large training runtime issue of the standard GAN in the existing work. We will also compare the goodness performance between the conditional GAN and the standard GAN.

5 Acknowledgment

This effort was funded by NSF CCRI and conducted in collaboration with Kobus Van Der Merwe (The University of Utah), Dustin Maas (The University of Utah), and Rahman Doost-Mohammady (Rice University). The authors acknowledge their guidance and discussion regarding the POWDER OTA experiments and the public POWDER dataset.

6 Appendices

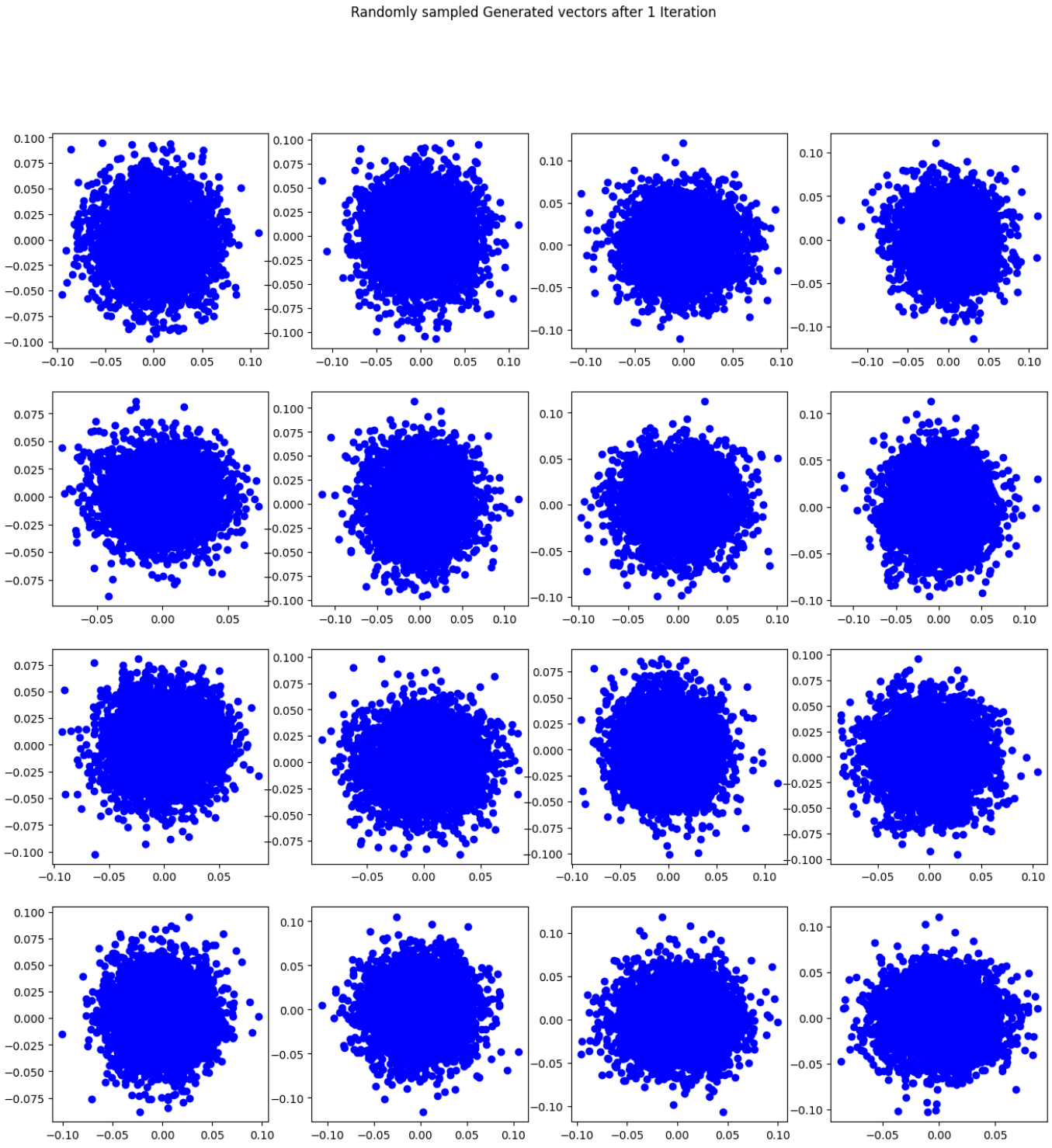


Figure 17: Generated Vectors at Initial Iteration

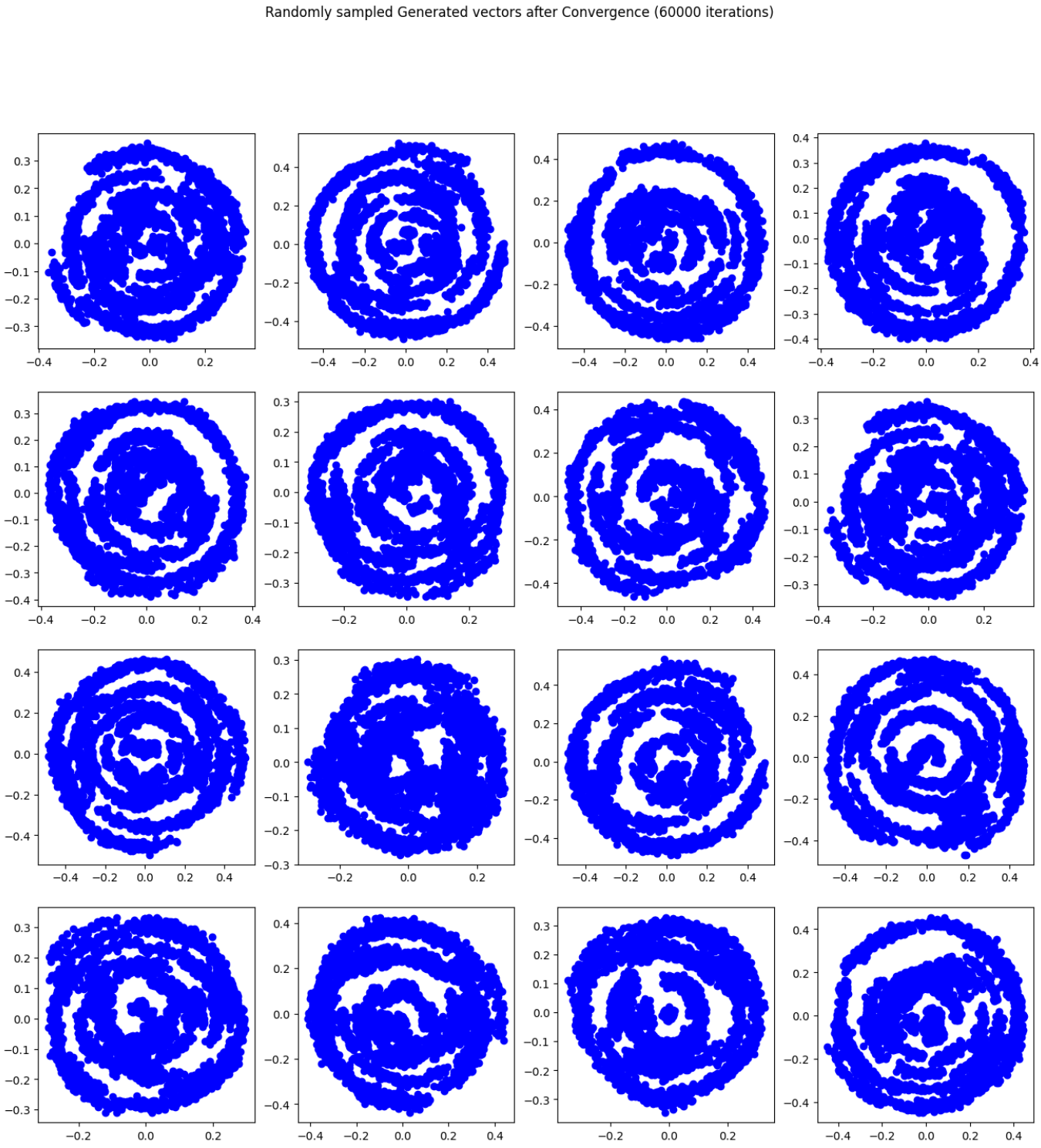


Figure 18: Generated Vectors at the Convergence iteration = 60000

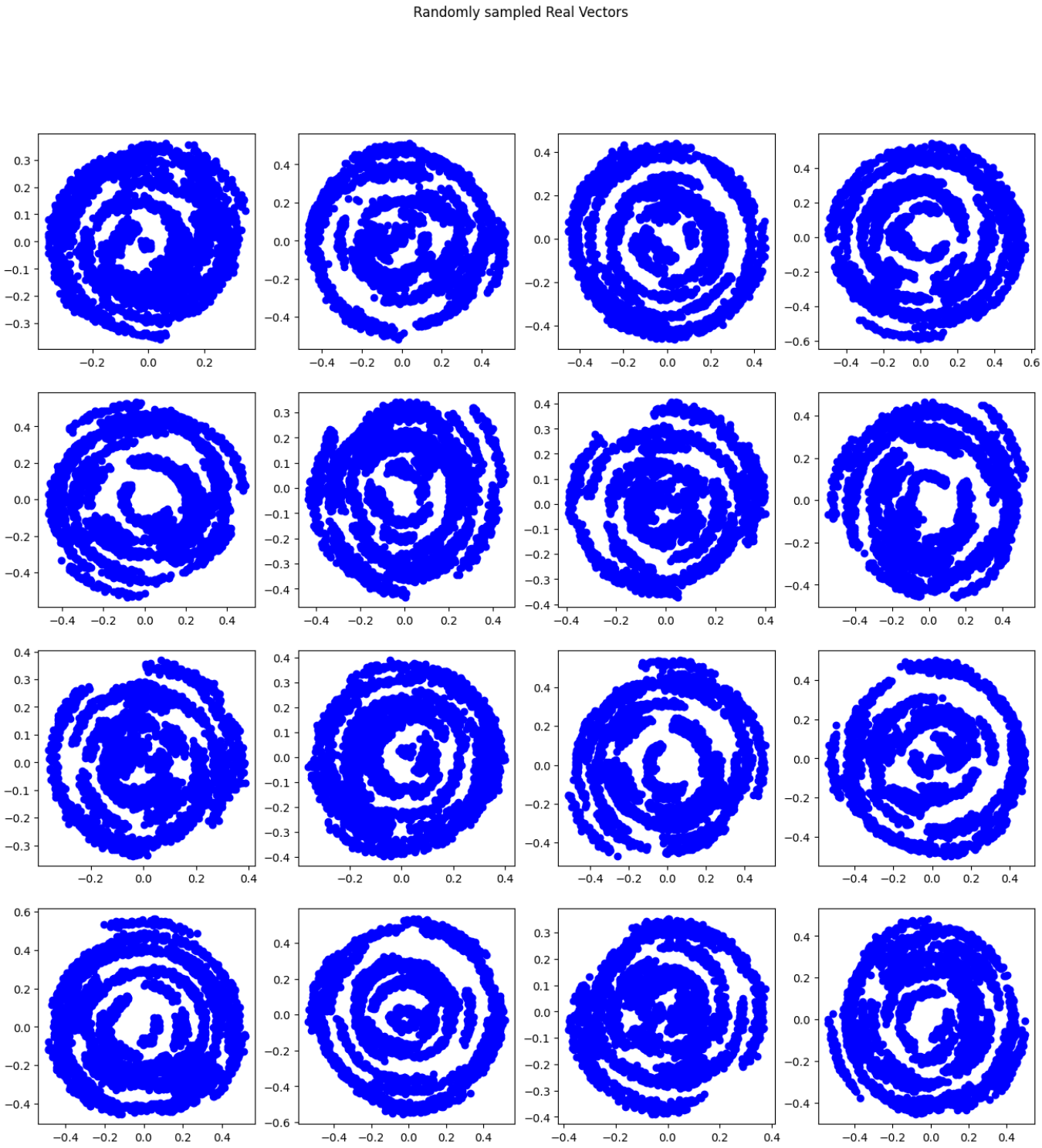


Figure 19: Real Vectors (powder dataset)

References

[1] Oscar Bejarano, Kirk Webb, and Rahman Doost-Mohammady. “Data and analysis script for channel measurement campaign at POWDER-RENEW using Iris SDRs”. In: (2020).

[2] Joe Breen et al. "POWDER: Platform for open wireless data-driven experimental research". In: *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*. 2020, pp. 17–24.

[3] Farzan Farnia and Asuman Ozdaglar. "Do GANs always have Nash equilibria?" In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3029–3039.

[4] Ke He et al. "Learning-based signal detection for MIMO systems with unknown noise statistics". In: *IEEE Transactions on Communications* 69.5 (2021), pp. 3025–3038.

[5] Yaqi Hu et al. "Multi-frequency channel modeling for millimeter wave and thz wireless communication via generative adversarial networks". In: *2022 56th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2022, pp. 670–676.

[6] Zhengdong Hu, Yuanbo Li, and Chong Han. *Transfer Generative Adversarial Networks (T-GAN)-based Terahertz Channel Modeling*. 2023. arXiv: 2301.00981 [eess.SP]. URL: https://arxiv.org/abs/2301.00981.

[7] Joakim Juhava et al. "Wireless channel modeling using generative machine learning models". MA thesis. 2023.

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[9] Timothy J O'Shea, Tamoghna Roy, and Nathan West. "Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks". In: *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2019, pp. 681–686.

[10] Iftikhar Rasheed et al. *LSTM-Based Distributed Conditional Generative Adversarial Network For Data-Driven 5G-Enabled Maritime UAV Communications*. 2022. arXiv: 2205.04196 [eess.SP]. URL: https://arxiv.org/abs/2205.04196.

[11] Ushnish Sengupta et al. "Generative diffusion models for radio wireless channel modelling and sampling". In: *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE. 2023, pp. 4779–4784.

[12] Nguyen Van Huynh et al. "Generative AI for physical layer communications: A survey". In: *IEEE Transactions on Cognitive Communications and Networking* (2024).

[13] Li Wei and Zhaohui Wang. "A variational auto-encoder model for underwater acoustic channels". In: *Proceedings of the 15th International Conference on Underwater Networks & Systems*. 2021, pp. 1–5.

[14] Tong Wu et al. "CDDM: Channel denoising diffusion models for wireless communications". In: *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE. 2023, pp. 7429–7434.

[15] Han Xiao et al. "ChannelGAN: Deep learning-based channel modeling and generating". In: *IEEE Wireless Communications Letters* 11.3 (2022), pp. 650–654.

[16] Hao Ye et al. "Deep learning-based end-to-end wireless communication systems with conditional GANs as unknown channels". In: *IEEE Transactions on Wireless Communications* 19.5 (2020), pp. 3133–3143.

[17] Qianqian Zhang, Aidin Ferdowsi, and Walid Saad. "Distributed Generative Adversarial Networks for mmWave Channel Modeling in Wireless UAV Networks". In: *ICC 2021 - IEEE International Conference on Communications*. 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9501056.

[18] Tianyu Zhao and Feng Li. "Variational-autoencoder signal detection for MIMO-OFDM-IM". In: *Digital Signal Processing* 118 (2021), p. 103230.