# Millimeter-Wave Statistical Channel Modeling for Indoor Factories Using Generative Nets

Shihao Ju and William Xia
*NYU Tandon School of Engineering, NYU WIRELESS, Brooklyn, NY, 11201*
{shao, william.xia}@nyu.edu

***Abstract*—Deep learning has been revolutionized many classical scientific fields such as image processing and natural language processing. Similarly, more and more researchers in wireless communications are trying to leverage deep learning techniques to improve the wireless system performance. This paper investigates the potential usage of deep learning into one of the most fundamental fields in wireless communications - channel modeling. Generative adversarial networks (GANs) are employed to generate life-like channel samples in a factory scenario. The results show that an auxiliary classifier GAN can faithfully recreate channels with reasonable path loss and azimuth angular spread based on the specified channel condition (line-of-sight (LOS) and non-LOS (NLOS)), but does not perform well from offsets in elevation angular spread and delay spread. This work lays the ground for generative channel modeling framework. More advanced GAN with discrete and continuous conditions will be studied in the future.**

***Index Terms*—Channel modeling; Ray tracing; Indoor factory; Generative adversarial net; Auxiliary classifier GAN; Millimeter wave**

## I. Introduction

### A. Background

The vast amounts of bandwidth (over tens of GHz) offered by the Millimeter-wave (mmWave) and Terahertz (THz) spectrum [1] have the potential to support rapidly increasing mobile traffic demand throughout the world, which is expected to reach 77 exabytes per month by 2022 [2]. Advancements in technologies that operate in these frequencies (e.g. massive multiple input output (MIMO) aims to support such stringent demands, but introduce more difficulties and challenges. Conventional channel modeling approaches may not accurately characterize the wireless channel given the increased number of complex modeling components. Thus, deep learning is considered to be a potential solution to this problem from a data-driven perspective. This project aims to employ a generative neural network to predict statistical channel information by using the simulated channels generated from a ray tracer as the training data.

### B. Conventional Channel Modeling

A received signal is commonly viewed as a superposition of multiple replicas of the transmitted signal with different delays and angles for any wireless propagation channel. Conventional channel modeling approaches can be classified into two types: deterministic approach and statistical approach. Deterministic approach describes the channel by modeling each propagation paths through free space propagation, reflection, diffraction, and scattering. Given a site-specific environment, the dimensions and building materials are required to generate propagation paths. Such deterministic approach is also well known as ray tracing, and high computation complexity and demanding requirements on the details of the environment are major concerns.

Instead, the statistical modeling approach considers the channel as a random variable, which usually cannot accurately recreate the channel conditions in a certain environment, but is capable of realizing the probabilistic distribution of a type of environment. A widely used cluster-based statistical channel model is given by [3]

$$h(t, \overrightarrow{\Theta}, \overrightarrow{\Phi}) = \sum_{n=1}^{N} \sum_{m=1}^{M_n} a_{m,n} e^{j\varphi_{m,n}} \cdot \delta(t - \tau_{m,n}) \cdot \delta(\overrightarrow{\Theta} - \overrightarrow{\Theta_{m,n}}) \cdot \delta(\overrightarrow{\Phi} - \overrightarrow{\Phi_{m,n}}), \tag{1}$$

where $t$ is the absolute propagation time, $\overrightarrow{\Theta} = (\phi_{\text{AOD}}, \theta_{\text{ZOD}})$ is the angle of departure (AOD) vector, and $\overrightarrow{\Phi} = (\phi_{\text{AOA}}, \theta_{\text{ZOA}})$ is the angle of arrival (AOA) vector. $N$ and $M_n$ denote the number of clusters and the number of subpaths within each cluster, respectively. For the $m$th subpath in the $n$th cluster, $a_{m,n}$, $\varphi_{m,n}$, $\tau_{m,n}$, $\overrightarrow{\Theta_{m,n}}$, and $\overrightarrow{\Phi_{m,n}}$ represent the magnitude, phase, absolute time delay, AOD vector and AOA vector, respectively. As shown above, each channel parameter is fitted by some distribution, but the correlations among these channel parameters may not be accurately reserved.

### C. Generative Channel Modeling

As the advent of deep learning, people start thinking about how deep learning techniques can be leveraged in wireless communications. Our idea is to train neural networks so that the neural networks can generate life-like channel information given a set of input parameters such as frequency, environment type, and separation distance.

There are a few attempts to apply generative neural networks to create wireless channels in various settings. A generative adversarial network(GAN)-based wireless channel modeling framework was proposed, where a vanilla GAN was used to generate additive Gaussian white noise (AWGN) channel without any domain-specific knowledge or technical expertise [4]. An experienced deep reinforcement learning

Note: Code for this work can be found at https://github.com/shihao-ju/DLProject.

(DRL) framework based on GANs was proposed to provide model-free resource allocation for ultra reliable low latency communication [5]. A variational GAN was trained to approximate wireless channel responses to more accurately reflect the probability distribution functions (PDFs) of stochastic channel behaviors [6]. In addition, conditional variational autoencoder (VAE) was used as an alternative generative model to generate pathloss and angular information of multipath components (MPCs) [7].

## II. Literature Survey on GANs

GANs were proposed by Goodfellow at 2014 [8] and has been considered one of the most exciting designs in the machine learning field, specifically in the generative model field. In general, many generative models share the principle of maximum likelihood (ML) estimation, even though some of those models do not use ML estimation by default, but can be made to do so. The principle of ML is to choose the parameters for the model that maximize the likelihood of the training data. It is worth noting that minimizing the Kullback-Leilber (KL) divergence is equivalent to maximize the likelihood between the probability distribution of the data and the probability distribution of the model.

Generative models using the ML principle can be categorized into two main classes: one using explicit density function either tractable or approximate densities, and the other one using implicit density function by leveraging samples. GAN lies in the implicit density class and samples directly from the distribution represented by the model. GAN differentiates itself from other generative models by exploiting a neural net as a expert to tell whether the input sample (either generated sample from the generator net or the real sample from real-world data) is fake or real. Such a expert is called discriminator, which needs to be trained along with the generator. Simply put, the GAN consists of two neural nets, one generator and one discriminator, as shown in Fig. 1. By training these two neural nets with two different objective functions, the generator is trained to fool the discriminator by generating realistic samples, and the discriminator is trained to distinguish between the generated fake samples and the real samples. Such design realizes a adversarial setting. Precisely, the generator and the discriminator are expected to reach Nash equilibrium after the training phase.

### A. Vanilla GAN

Assume the real-world data $x$ follows a probabilistic distribution $p_{\text{data}}(x)$, and the input to the generator $z$ is white noise with a prior distribution $p_z(z)$ (e.g., standard normal distribution). The generator maps the input noise vector to the data space $G(z;\theta_g)$, which $\theta_g$ parameterizes the generator neural net. On the other hand, the discriminator maps the input sample (data space) to a scalar $D(x)$ which represents the probability of assigning the correct label. Thus, the objective function of the discriminator is to maximize the probability of assigning the correct label to the real-world samples and to minimize the probability of assigning the correct label to the generated fake samples. On the contrary, the objective function of the generator is to maximize the probability of the discriminator assigning the correct label to its generated samples. Using a standard binary cross-entropy function, the lost function of the discriminator can be written by

$$\begin{aligned}\mathcal{L}^{(D)} = &-\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]- \\ &\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]\end{aligned} \tag{2}$$

The GAN forms a two-player minimax game with value function by setting the loss function of the generator as the inverse of the loss function of the discriminator, which is also called V(D,G) [8]:

$$\begin{aligned}\min_G \max_D V(D,G) =&\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]+ \\ &\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]\end{aligned} \tag{3}$$

Note that the inputs to the function $G$ do not need to correspond to the inputs to the first layer of the deep neural net, which may be provided at any point throughout the network. A popular strategy is to apply the additive or multiplicative noise to hidden layers or concatenate noise to hidden layers of the neural nets. Importantly, the dimension of $z$ must be at least as large as the dimension of $x$ to make $p_{\text{model}}$ have the full support on $x$.

Optimizing the value function in (3) can be proven to be equivalent to optimizing the Jensen-Shannon (JS) divergence, which is a symmetric and finite version of KL divergence:

$$\begin{aligned}D_{JS}(p_{\text{data}}||p_z) = &\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\log\frac{p_{\text{data}}(x)}{\frac{p_{\text{data}}(x)+p_g(x)}{2}} \\ &+\frac{1}{2}\mathbb{E}_{x\sim p_g}\log\frac{p_g(x)}{\frac{p_{\text{data}}(x)+p_g(x)}{2}},\end{aligned} \tag{4}$$

where $p_g$ is the distribution of generator data. Further, the loss function given the optimal discriminator is

$$\mathcal{L}^{(D^*)} = 2\log 2 - 2D_{JS}(p_{\text{data}}||p_g) \tag{5}$$

The optimal generator is to make $p_g$ equal to $p_{\text{data}}$ everywhere.

However, the above value function may be good for theoretical analysis, but does not perform well in practice due to the vanishing gradient. Thus, another heuristic loss function is widely used in the training by flipping the labels for the generator:

$$\mathcal{L}^{(G)} = -\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))], \tag{6}$$

suggesting that for the generator, instead of minimizing the log-probability of the discriminator being correct, maximizes the log-probability of the discriminator being mistaken. Note that the discriminator still uses (3).

### B. f-GAN

f-GAN was proposed by Nowozin, *et.al* in 2016 [10]. f-GAN is a general framework based on f-divergence, which provides a methodology to construct various types of GANs. f-divergence has a form:

$$\mathcal{D}_f(P||Q) = \int q(x) f(\frac{p(x)}{q(x)})dx. \tag{7}$$

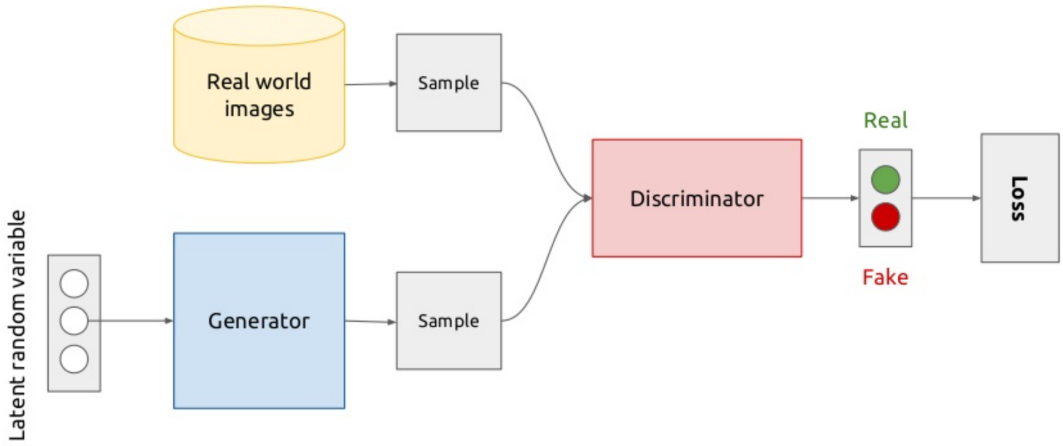



Fig. 1: GAN consists of two neural nets, one generator and one discriminator [9].

The aforementioned KL divergence and JS divergence are special forms of f-divergence. Other examples of f-divergence are total variations, Pearson $\chi^2$, and Squared Hellinger. The key idea is to make a variational estimation of f-divergences in a form of convex conjugate functions. With properly selected activation functions, every f-divergence can be converted to a objective function (minimax game) for GANs. This work provided a uniformed theoretical framework for a type of objective functions for GANs. However, the practical issues for the vanilla GAN will still bother these f-GANs, thus the practical usage and improvement need more investigations.

### C. WGAN

WGAN was first proposed by Arjovsky, *et. al* in 2017 [11], and improved techniques to train WGAN was proposed by Ishaan, *et. al* later in the same year [12]. Unlike f-GANs, WGAN leverages the concept of Wasserstein-1 distance to quantify the difference between the probability distributions:

$$\mathcal{W}(p,q) = \inf_{\gamma \in \Pi(p,q)} \int\int \gamma(x,y) d(x,y) dx dy, \tag{8}$$

which is also known as the Earth-Mover (EM) distance. $\Pi(p,q)$ denotes the set of all joint distributions $\gamma(x,y)$ whose marginals are $p(x)$ and $q(y)$, respectively. Intuitively, $\gamma(x,y)$ represents the amount of "mass" must be transported from $x$ to $y$ in order to transform the distribution $p(x)$ to the distribution $q(y)$, and $d(x,y)$ represents the cost of such a step of movement $\gamma(x,y)$. The EM distance is the minimum cost obtained by the optimal transport plan.

This minimization problem is highly intractable, which can be converted to a maximization problem using Kantorovich-Rubinstein duality:

$$\mathcal{W}(p,q) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim p(x)}[f(x)] - \mathbb{E}_{x\sim q(x)}[f(x)], \tag{9}$$

where $||f||_L \leq 1$ represents the 1-Lipschitz function to constrain the continuity of the cost function. Then, this maximization problem can used as the cost function of the discriminator. The superior property of the EM distance is that it can smoothly represent the distance between two arbitrary distributions. On the contrary, the f-divergence cannot well characterize the distance between two distributions with no overlap.

### D. LSGAN

Least square GAN (LSGAN) was first proposed by Mao *et. al* in 2017 [13]. LSGAN employs the least square loss function for the discriminator, in which way the generator is not satisfied with only making fake samples on the correct side of the decision boundary but make the fake distribution as close as the real distribution. Thus, the gradient vanishing problem can be relived by the this setting. The loss function is given by

$$\begin{aligned}\mathcal{L}^{(D)} = -\mathbb{E}_{x\sim p_{\text{data}}(x)}[(D(x)-1)^2] - \\ \mathbb{E}_{z\sim p_z(z)}[D(G(z))^2]\end{aligned} \tag{10}$$

### E. DCGAN

Deep convolutional GAN (DCGAN) was first proposed by Radford, *et. al* in 2015 [14]. The core idea of DCGAN is to use deep convolutional nets (CNN) at both generator and discriminator. Instead of having pooling layers as in conventional CNN, The discriminator takes fake and real images as input and performs strided convolutions to down-sample the images. The generator performs fractional-strided convolutions (also known as transpose convolutions) to map the random noise vectors to the size of images. The original DCGAN uses the same loss functions for the discrminator and the generator as vanilla GAN.

### F. Conditional GAN

Conditional GAN (CGAN) was first proposed by Mirza, *et. al* in 2014 [15]. CGAN provides an additional way to interact with the GANs by making the discriminator and the generator conditioned on some extra information. Take the MNIST data as an example, the generator can be asked to generate hand-written digit images for a certain chosen digit. The labels have been proven to greatly improve the GAN performance

in various tasks. The distinct labels are encoded as one-hot vectors and fed into the GANs as input. Furthermore, the loss function is slightly modified by replacing $D(x)$ and $D(G(z))$ with $D(x|y)$ and $D(G(z|y))$.

### G. ACGAN

Auxiliary classifier GAN (ACGAN) was first proposed by Odena, *et. al* in 2016 [16]. ACGAN is an improved version of CGAN, where each generated sample has a corresponding class label. The generator uses both the noise vector and the class to generator images, and the discriminator outputs the probability of whether the generated sample is real or fake and the probability of whether the generated sample belongs to the given class. ACGAN is considered to perform better than the CGAN since the nets are asked to do some extra multi-class classification work.

### H. InfoGAN

InfoGAN was first proposed by Chen, *et. al* in 2016 [17]. InfoGAN differentiates itself with other GAN settings by the capability of learning disentangled representation in an unsupervised way. InfoGAN provides the control of the nets by inputting a sequence of latent code (compared to the condition used in CGAN and ACGAN), and learns various types of conditions unsupervisely. In other words, the latent code is used to represent the features discovered by the nets, which allow us to change the generated samples by modifying the latent code. To solve the relation between the latent code and the generated output, InfoGAN used the mutual information, a concept from information theory, which measures the mutual dependence between two random variables. Thus, the information-regularized minimax game is given by

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c)), \qquad (11)$$

where $V(D,G)$ is given in 3, and $I(c; G(z,c))$ represents the mutual information between the latent code and the generator distribution.

### I. Summary

There have been extensive research on the development of GANs with respect to the network architecture, loss function, training procedure, practical considerations. However, the reason why GAN performs so well in tasks such as generating realistic high-resolution images is still not fully clear. GAN is one of the most intriguing frameworks among all the generative models, and more research and effort will be invested in the coming decades.

## III. Data Acquisition and Processing

We produce our own data set for this project by using the a commercial raytracer: Wireless Insite Suite from Remcom [18]. The first environment that we attempt to train a model to generate samples for is the 'Factory Warehouse' scenario. This is shown in Fig. 2, where the receivers are denoted with red nodes, and the transmitters are denoted with green nodes. Note that complex environmental factors such as reflection coefficients and material permittivities of various objects like boxes, doors, guard rails, walls, windows, etc. are all considered in the simulator.

The parameters of the simulation performed are given in Table I.

TABLE I: Remcom Simulator configuration

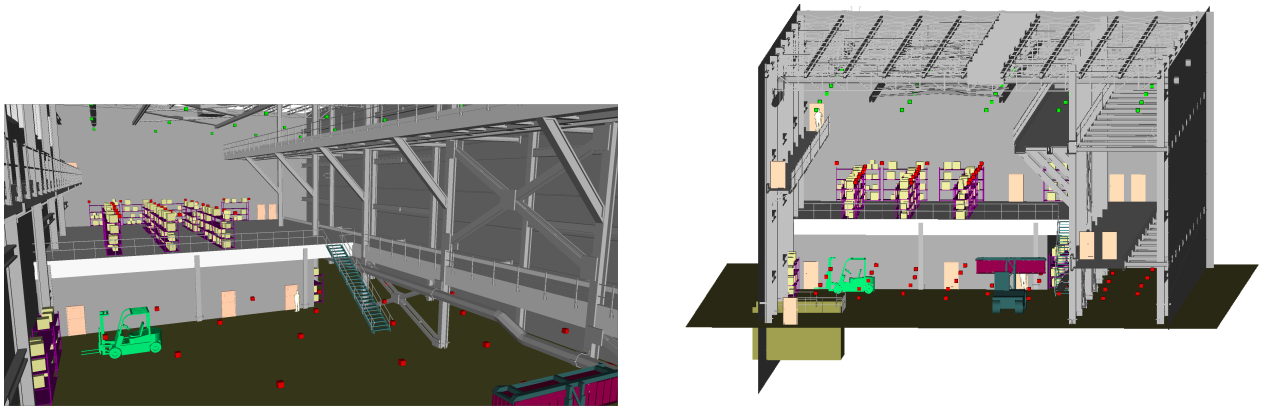
| Parameter | Value |
|---|---|
| Number of TX | 35 |
| Number of RX | 67 |
| Transmitter (gNB) Height | 16 m |
| Receiver (UE) Height (Floor 1) | 2 m |
| Receiver (UE) Height (Floor 2) | 8 m |
| Max Number of Paths | 20 |
| Max Number of Reflections | 6 |
| Max Number of Diffractions | 1 |
| Carrier Frequency | 28 GHz |
| Transmit Power | 16 dBm |

The output of the ray-tracing simulation contains many channel statistics such as the the number of paths per transmission link, as well as the received power ($P_r$), propagation delay ($\tau$), AOA ($\phi_{AOA}, \theta_{ZOA}$), and AOD ($\phi_{AOD}, \theta_{ZOD}$) of each path. In order to process all the output data, a script was written to parse the data using the following format:

- Data structure $x$ is of size $n_{\text{samp}} \times 308$
- Each row of the $x$ represents the statistics of a link between a single TX-RX pair
- The first three columns give the TX location in cartesian coordinates, while the next three similarly give the RX location. This is followed by the number of rays in the link, the link state (LOS or NLOS), and the number of paths.
- The following 300 columns come from the 20 possible paths each of which contain the following 12 parameters: received power, path propagation delay, azimuth & elevation of departure, azimuth & elevation of arrival, cartesian coordinates of the first interaction after the transmitter, and finally the coordinates of the interactions before the receiver.

The data was processed in this way to handle the fact that any link can have a variable number of paths. For example, a link which has only 2 paths will have $8 + 2 \times 12 = 32$ columns of usable data, with all subsequent columns in that row being empty.

The parsed dataset was then further processed and put into a tensorflow dataset. Specifically, the final dataset has 124 columns. The first four columns represent the 3-D distance between the transmitter and receiver position ($d_{\text{3D}}$), the log of the 3-D distance ($10 \log_{10}$ 3D, the vertical distance ($d_z$), and the visibility condition (LOS or NLOS). The following 120 columns are composed by the six attributions of each of the 20 paths (i.e., $P_r, \tau, \phi_{AOA}, \theta_{ZOA}, \phi_{AOD}, \theta_{ZOD}$). These sample links all have these parameters which naturally have



(a) Inside View (b) Outside View

Fig. 2: Factory warehouse environment including transmitters and receivers.

their own respective distributions. Thus the goal of this project is to use this data set to train a generative model that is capable of reproducing link data that follows the same distribution as the actual links that are found in the given environment.

## IV. Network Structure and Implementation

The whole nets are constituted by two nets, one full-connected net for the link state prediction and one ACGAN for the link information generation. The link state prediction network has two hidden layers with 20 and 40 nodes, respectively. The category cross entropy is selected as the loss function to predict one of LOS, NLOS, and outage states. The train accuracy and test accuracy are about 90% and 85%, respectively. More details can be found in the notebook PDF file.

### A. Auxiliary Classifier GAN

As introduced in Section II, both conditional GAN and ACGAN are able to take conditions (discrete labels) into the generator and discriminator networks. Here, we selected AC-GAN since we are interested in finding out the performance of the auxiliary classification function in the discriminator.

*1) Generator:* The generator takes a random vector, whose element is typically uniform random variables between -1 and 1. Additionally, the generator concatenate the specified one-hot label with the random vector as the input. Through two set of dense layer, batch normalization layer, and activation layer, the output vector is further mapped to the size of one channel sample (i.e., 120 elements) with a sigmoid activation function.

*2) Discriminator:* The discriminator takes the real and fake channel samples as input. After two set of dense layer, batch normalization layer, and activation layer, the discriminator output the decisions through the main network and the labels through the auxiliary network. The decision for each channel sample is a scalar, true or false. The label is a probability vector after softmax operation.

*3) Summary:* Detailed network summaries can be found in the notebook output PDF file. The hyperparameter settings of training the generator and discriminator are given in Table. II. The size of the input noise vector (latent variables) is set to 20, and the size of each input sample to the discriminator is 120 (as explained in Section III). During each training iteration, the discriminator is updated based on the gradient of the discriminator loss function calculated by setting the labels of real-world samples as one and the labels of generated samples as zero; the generator is updated based on the gradient of the adversarial loss function. After the GAN is trained, the same number of channel samples as the test dataset are generated from the generator and evaluated in the Section V.

TABLE II: Hyperparameters of the generator and discriminator

| Hyperparameters | Generator | Discriminator |
|---|---|---|
| **Number of inputs** | 20+2 | 120 |
| **Hidden unit** | [32, 64] | [64, 32] |
| **Number of outputs** | 120 | 1+2 |
| **Optimizer** | Adam | Adam |
| **Learning rate** | 1e-4 | 2e-4 |
| **Epochs** | 500 | 500 |
| **Batch size** | 64 | 64 |
| **Loss function** | Binary cross-entropy<br>Categorical cross-entropy | Binary cross-entropy<br>Categorical cross-entropy |

## V. Modeling Results

### A. Path Loss

Path loss is the most important performance metric for wireless communications. A good estimation and prediction of the path loss facilitates the design of cell coverage, handover schemes, and system throughput, etc. We use the close-in free space reference distance (CI) path loss model with 1 m

reference distance [3]. $\mathrm{PL}^{\mathrm{CI}}$ represents the path loss in dB scale, which is a function of distance and frequency:

$$\mathrm{PL}^{\mathrm{CI}}(f,d)[\mathrm{dB}] = \mathrm{FSPL}(f,d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) + \chi_\sigma, \quad \text{for } d \geq d_0, \quad \text{where } d_0 = 1\mathrm{m} \tag{12}$$

where $n$ denotes the path loss exponent (PLE), and $\chi_\sigma$ is the shadow fading (SF) that is commonly modeled as a lognormal random variable with zero mean and $\sigma$ standard deviation in dB. $d$ is the 3-D T-R separation distance. $d_0$ is the reference distance, and $\mathrm{FSPL}(f,d_0) = 20\log_{10}(4\pi d_0 f/c)$. The CI path loss model uses the FSPL at $d_0 = 1$ m as an anchor point and fits the measured path loss data with a straight line controlled by a single parameter $n$ (PLE) obtained via the minimum mean square error (MMSE) method.

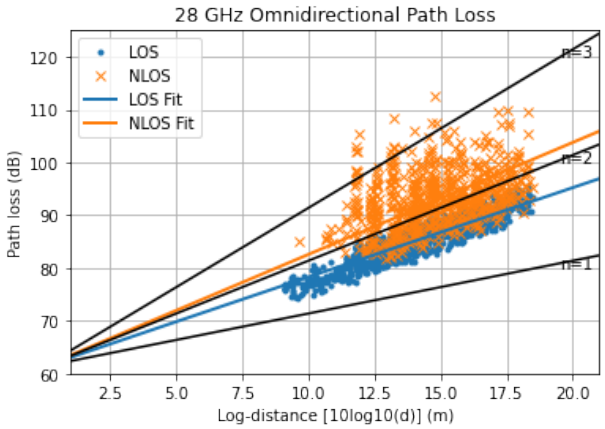



Fig. 3: The omnidirectional path loss from train dataset.

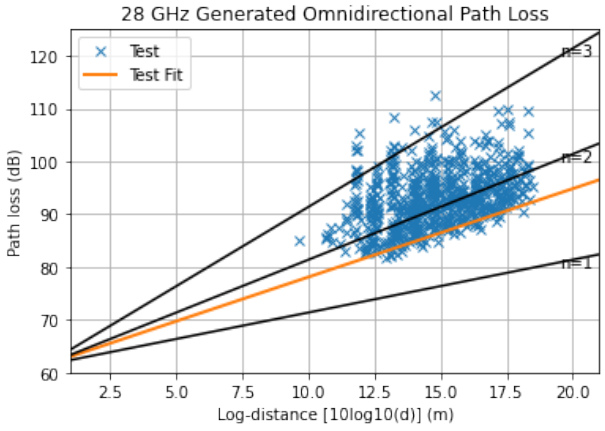



Fig. 4: The omnidirectional path loss created from the generator.

Fig. 3 shows the omnidirectional path loss plot for the train dataset, where the PLE for the LOS and NLOS scenarios are 1.69 and 2.12. The PLE ($n$) of free space propagation is 2. The omnidirectional received power in the LOS scenario includes not only the LOS free space propagation path, but many reflected and scattered paths. Thus, the measured PLE of omnidirectional channels is usually less than 2. The omnidirectional PLE for the NLOS scenario usually ranges from 2 to 3. Fig. 4 shows the generated omnidirectional path loss at test locations, where the PLE is about 1.67. The generated path loss values present a good agreement with the measured path loss values. Next we plot the cumulative probability

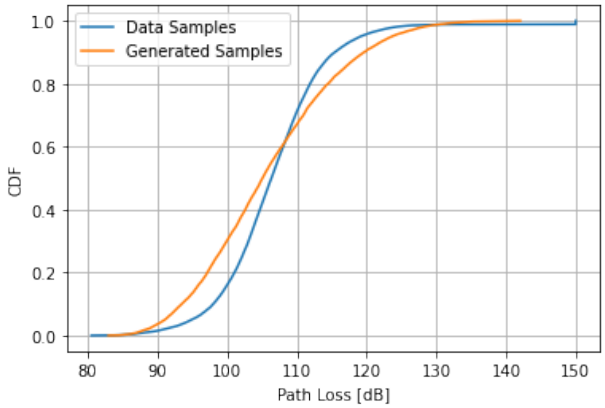



Fig. 5: The cumulative distribution of the path loss for paths across NLOS links

distribution (CDF) of the path losses from the NLOS links of both the test data, and of the link generated from our model. We can see that currently, the model generates links that follow the true distribution for paths that are at approximately 105 dB and below. However, for the remainder of the NLOS links, the GAN is overly pessimistic and generates paths that have path losses that are worse than the original data.

### *B. LOS Probability*

Another metric that we are interested in modeling is the probability of have a LOS link between TX and RX in this environment. Fig. 6 shows the LOS probability for both the

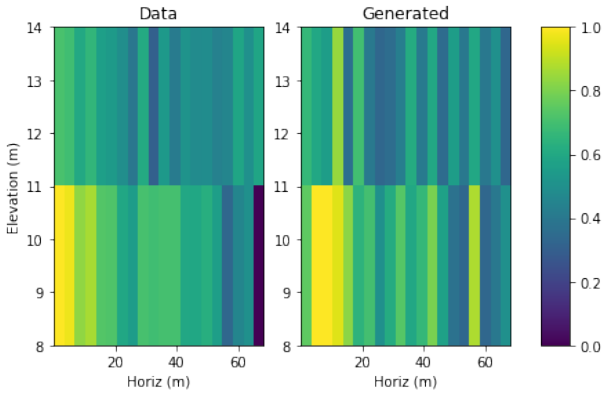



Fig. 6: Probability of LOS parameterized by elevation and horizontal distance.

generated samples and the original ray-traced data. Overall, there is a decreasing probability as the UE moves further away in the horizontal plane as well as vertically upwards in elevation. There are several regions further away from the gNB for which the probability of LOS is quite low, and the neural net seems to be able to accurately capture this trend.

### C. Angular Spread

The angular spread is another interesting metric to study for wireless channels. In our case, we generate samples that include directions of arrival and compare this to the original Remcom data in Fig. 7. What we can see is that our model

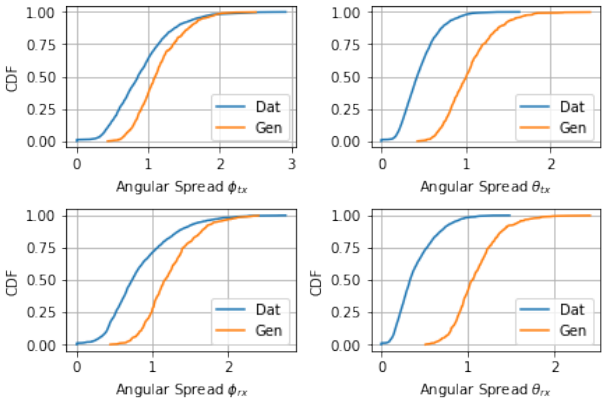



Fig. 7: CDF of the Angular Spread

generates paths that have angular spreads that are wider than the original data. That is, our generated paths seems to be much more erratic, with higher variation in both departure direction and arrival directions. Interestingly, the distributions of the azimuth of departure and azimuth of arrival are both modeled better by the ACGAN than their zenith counterparts. This could be due to lack of data at various elevations or other factors.

### D. RMS Delay Spread

Finally, the last parameter that we consider is the root mean square (RMS) delay spread. This metric effectively determines how much variation exists in the delay spread across each link. We compute and plot the CDF of the RMS delay spread for both the original data and generated samples across all their respective links in Fig. 8. For the RMS delay spread, we can

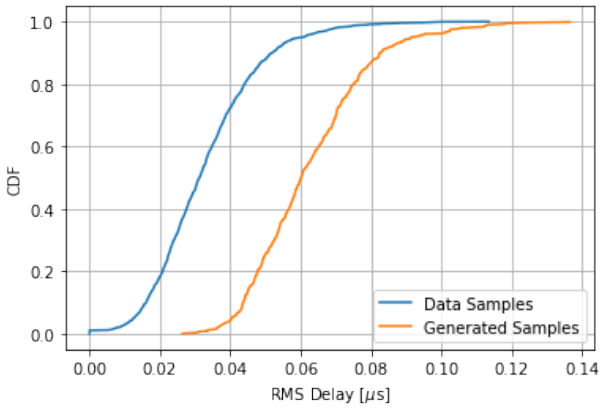



Fig. 8: CDF of the RMS Delay Spread Across all Links

see that the generated samples have a larger delay spread. The generator seems to have modeled paths which an average RMS delay spread of about 0.06 $\mu s$, while the original data actually has an average RMS delay spread of about 0.3 $\mu s$.

## VI. Conclusion

This work implements an ACGAN to learn the distributions of several key multipath component parameters such as path loss, LOS probability, angular spread, and delay spread. We also supplement this with a standard fully connected dense neural network to predict the link state. We successfully implement this architecture which is widely used for images to a potential application in wireless channel modeling, and developed a basic infrastructure to building the models, training them, and then evaluating their performance.

That said, there were many challenges that arose with this work which we suspect to have affected the performance and our ability to assess the models. In [7], the conditional VAE used was able to accept the condition vector into both the encoder and decoder networks as parameters. Note that the condition vector values are continuous valued, and not labels. This became an apparent concern in the design of the GAN in our work, since there is few GAN implementation that feeds the discriminator and generator networks with the condition information. Another challenge that we faced in implementing the ACGAN (although this is also applicable to generative models broadly) for wireless channel models was the fact that the outputs of the model can only be evaluated in terms of a distribution. Unlike the use of generative modeling for an image, where a person can visually determine how realistic a generated image is, there is no readily available equivalent for multipath components to a wireless channel.

Finally, there are several extensions to further this work. Although several forms of preprocessing were employed to aide in the training of the models, different methods can be implemented to try to improve the performance of the generative model. Furthermore, we are looking for approaches to input continuous parameters as conditions such as T-R separation distance, carrier frequency, RF bandwidth, etc.

Through this project, we have read many papers on GANs and tried several different implementations. Deep learning in wireless communications, specifically in channel modeling, is still yet to be exploited. We hope to take this project as the first step for future research.

## References


[1] T. S. Rappaport, Y. Xing, O. Kanhere, S. Ju, A. Madanayake, S. Mandal, A. Alkhateeb, and G. C. Trichopoulos, "Wireless communications and applications above 100 GHz: Opportunities and challenges for 6G and beyond," *IEEE Access*, vol. 7, pp. 78 729–78 757, 2019.

[2] CISCO, "Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022," 2019.

[3] S. Ju, Y. Xing, O. Kanhere, and T. S. Rappaport, "Millimeter wave and sub-terahertz spatial statistical channel model for an indoor office building," *IEEE Journal on Selected Areas in Communications*, pp. 1–15, May 2021.

[4] Y. Yang, Y. Li, W. Zhang, F. Qin, P. Zhu, and C. Wang, "Generative-adversarial-network-based wireless channel modeling: Challenges and opportunities," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 22–27, 2019.

[5] A. T. Z. Kasgari, W. Saad, M. Mozaffari, and H. V. Poor, "Experienced deep reinforcement learning with generative adversarial networks (gans) for model-free ultra reliable low latency communication," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 884–899, 2021.

[6] T. J. O'Shea, T. Roy, and N. 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)*, 2019, pp. 681–686.

[7] W. Xia, S. Rangan, M. Mezzavillla, A. Lozano, G. Geraci, V. Semkin, and G. Loianno, "Generative neural network channel modeling for millimeter-wave uav communication," in *2020 GLOBECOM - IEEE Global Communications Conference Workshop*, 2020, pp. 1–6.
[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.
[9] K. McGuinness. (2017) Generative models and adversarial training (d3l4 2017 upc deep learning for computer vision).
[10] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," *arXiv preprint arXiv:1606.00709*, 2016.
[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.
[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.
[13] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2794–2802.
[14] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.
[15] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.
[16] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*. PMLR, 2017, pp. 2642–2651.
[17] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *arXiv preprint arXiv:1606.03657*, 2016.
[18] "Remcom," available on-line at `https://www.remcom.com/`.