IL-GAN: Rare Sample Generation via Incremental Learning in GANs

Jón R. Baldvinsson*†, Milad Ganjalizadeh*†, Abdulrahman AlAbbasi*, Mårten Björkman†, Amir H. Payberah†

*Ericsson Research, Stockholm, Sweden
†KTH Royal Institute of Technology, Stockholm, Sweden
Email: jrba@kth.se, {milad.ganjalizadeh, abdulrahman.alabbasi}@ericsson.com, {celle, payberah}@kth.se

***Abstract*—Industry 4.0 imposes strict requirements on the fifth generation of wireless systems (5G), such as high reliability, high availability, and low latency. Guaranteeing such requirements implies that system failures should occur with an extremely low probability. However, some applications (e.g., training a reinforcement learning algorithm to operate in highly reliable systems or rare event simulations) require access to a broad range of observed failures and extreme values, preferably in a short time. In this paper, we propose IL-GAN, an alternative training framework for generative adversarial networks (GANs), which leverages incremental learning (IL) to enable the generation to learn the tail behavior of the distribution using only a few samples. We validate the proposed IL-GAN with data from 5G simulations on a factory automation scenario and real measurements gathered from various video streaming platforms. Our evaluations show that, compared to the state-of-the-art, our solution can significantly improve the learning and generation performance, not only for the tail distribution but also for the rest of the distribution.**

***Index Terms*—generative adversarial networks, rare-event simulations, incremental learning, tail statistics, URLLC.**

I. Introduction

The Fourth Industrial Revolution (i.e., Industry 4.0) is the ongoing automation of manufacturing and industrial practices using modern technology, such as cyber-physical systems, the Internet of things, and cloud computing. Autonomous mobile robots and smart factories are examples of Industry 4.0 applications. Such applications typically require high reliability (e.g., 10 years without failure), high availability (e.g., 99.9999%), and low latency (e.g., <1 ms) [1]. However, fulfilling such stringent requirements is envisioned with the fifth generation of mobile networks, 5G, and beyond 5G via ultra-reliable low latency communication (URLLC) service [2].

A failure in Industry 4.0 applications is typically defined as an event where either a packet cannot be decoded at the receiving entity or is received after its corresponding delay bound [3]. Since we expect extreme system performance in industry 4.0 applications, the tail distribution of metrics resulting in failure events is often small. However, when such rare samples are required, this distribution behavior can cause a problem, for instance, during the training of a deep reinforcement learning (DRL) agent, which acts as an orchestrator for any radio access network functionality. In this example, infrequent encounters with rare events during training may lead to improper decision-making when attempting to avoid failure events in deployment [4].

We define a *rare event environment* as an environment where the distribution of a selected metric is characterized by a few head classes occupying most of the instances and a long-tail having very few instances. Significant research has been done to speed up simulators in rare event environments and increase the amount of simulated rare events. For example, the authors in [5] proposed importance sampling (IS), as an alternative method to monte carlo (MC) sampling, when MC sampling is infeasible or inefficient (e.g., in extreme performance environments). IS modifies the sampling procedure to increase the probability of sampling from a specific area of the distribution and to provide an unbiased result.

At least one rare event (ALOE) [6], is an example of IS, repeatedly chooses rare events at random and samples the system conditionally on the chosen error taking place. ALOE has higher accuracy on rare events than other sampling methods in high-dimensional environments. Additionally, ALOE offers speedup of orders of magnitude over MC methods in rare event wireless networking environments [7]. However, ALOE has worse accuracy than other sampling procedures on the events that do not consider rare events.

Another alternative to governing the occurrence of rare events is to use generative adversarial networks (GANs). GANs are generative models that utilize an adversarial learning process using two separate networks: (i) a *generator* network and (ii) a *discriminator* network. The generator network generates a data sample, and the discriminator network receives the generated sample and a data sample from a real dataset. The discriminator's objective is to detect which data samples are real, and which are fake. The generator's objective is to generate realistic data such that the discriminator is not able to differentiate between generated data and real data [8].

Conditional GAN is a modification of the original GAN architecture that uses additional and conditional information as input to the generator and the discriminator. This modification allows for control over the generated data [9].

As an example of generating rare events, [4] creates a virtual environment that utilizes a GAN based refiner to pre-train a DRL scheduler, i.e., to provide the agent with more experience before its deployment.

This work was partially supported by Swedish Foundation for Strategic Research (SSF) under Grant iPhD:ID17-0079.

Inspired by learning to segment the tail (LST) approach in [10], in this paper, we propose incremental learning GAN (IL-GAN) through which we enable GANs to learn and govern the generation of rare events. Unlike LST, our solution enables efficient sample generation from all datasets with long-tail distributions; thus, it can benefit machine learning-based solutions for orchestration and management of Industry 4.0 applications, where failures are expected to happen rarely. Our main contributions are as follows:

- We design segmentation techniques to split unbalanced wireless communication datasets into balanced datasets to be suited and utilized for the training of IL-GAN.
- Using balanced replay and layer freezing, we develop IL-GAN, an incremental learning framework on top of the conventional GAN, which augments GANs to uniformly generate samples, regardless of the frequency of their occurrence in the dataset. To the best of our knowledge, this is the first attempt at leveraging incremental learning to enhance generative models.
- We evaluate IL-GAN using data generated by propriety near-product factory automation simulations and real measurements. Our evaluations show that IL-GAN outperforms state-of-the-art GANs in generating URLLC data. In particular, on the extreme tail, our IL-GAN can generate data that its Jensen-Shannon (JS) divergence to real distribution is one-fifteenth of the baseline's JS divergence, with little to no impact on the generation of the samples from the rest of the distribution.

The rest of this paper is organized as follows. We provide required preliminaries in Section II and explain the data collection procedure in Section IV. We describe the proposed IL-GAN in Section III, and present its experimental results in Section V. Finally, conclusions are summarized in Section VI.

II. Preliminaries

Training GANs is often unstable, and it has been shown that leveraging Kulback-Leibler (KL) divergence and Jensen-Shannon (JS) divergence as their training loss functions can lead to training instability, including vanishing gradients issues [11], [12]. Alternatively, Arjovsky et al. [12] introduced Wasserstein GAN in which they used the Earth-Mover distance, formulated as below:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[||x - y||\big], \tag{1}$$

where $\prod(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. In other words, $\gamma(x, y)$ is the amount of mass that needs to be transported from x to y to transform the distribution $\mathbb{P}_r$ into $\mathbb{P}_g$.

Using the Kantorovich-Rubinstein duality to remove the infimum, the loss function becomes

$$\max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}\big[f_w(x)\big] - \mathbb{E}_{z\sim p(z)}\big[f_w(g_\theta(z))\big], \tag{2}$$

where f_w is the set of 1-Lipschitz functions, g_θ is a parametric function and z is a latent variable, sampled from a fixed noise distribution $p(z)$.

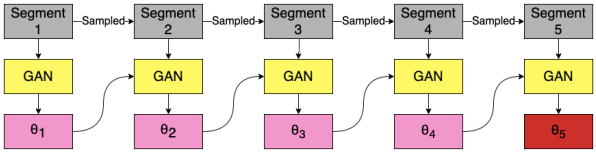


Figure 1: The architecture of incremental learning.

Using neural networks as parametric functions, where f_w is a discriminator network, and g_θ is a generator network, it becomes very similar to the GAN objective. The output is no longer a probability; hence, we do not use a sigmoid function on the output layer of the discriminator. To enforce the Lipschitz constraint on the discriminator, we use gradient penalty on the parameters of the discriminator [13].

Since the Earth-Mover distance is almost continuous and differentiable everywhere, we should train the discriminator to optimality because further discriminator training results in more reliable gradients. Hence, we introduce n_c to limit the discriminator iterations to control the training time. For example, if n_c=5, the discriminator's parameters are updated five times before the generator's parameters are trained once.

III. IL-GAN: Leveraging Incremental Learning in Generative Adversarial Networks

Training on data dominated by a large head distribution and a long-tail distribution is problematic. These infrequent encounters with data from the tail distribution cause the corresponding updates to the model parameter to be overwritten in favor of the more frequent updates corresponding with data from the head distribution. Due to this, the conventional GAN trained on the whole dataset tends to fit well to the head distribution while ignoring the tail of the distribution. In this section, we redesign incremental learning, first introduced in LST [10] for image classification and segmentation tasks, to enhance generative models in learning and generation of the data from the tail of the distribution.

The main idea behind incremental learning is to learn segment by segment and ensure that learning new segments does not result in forgetting the old ones. The abstract algorithm of IL-GAN is as follows: first, we divide the data into segments, then train the model on the data in the first segment, which includes the head distribution. Such training continues until some convergence criterion is met or for a fixed number of epochs. Upon completing the training on the data in segment one, the model is trained using the data from segment two. This procedure continues until the model is trained using the data from all segments. Figure 1 shows the architecture of incremental learning. During incremental learning, the focus is on learning new segments. To combat catastrophic forgetting, we propose balanced replay and freezing for training GANs.

A. Balanced Replay

Balanced replay is a sampling strategy in which the data is sampled from previous segments and added to the training data of the current segment being trained. Hence, it balances previously learned segments and the current segment. Consequently, the model represents all observed segments and the

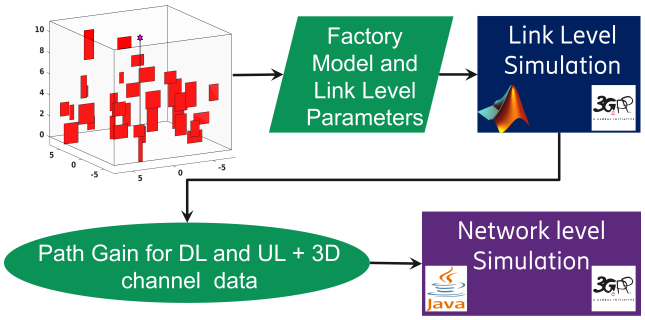


Figure 2: The simulation setup for generating simulated data.

current segment's data. Besides, balanced replay allows the model to modify the weights to remember the old segments and learn the new segments. Since there is a balance between the segments during training, the model can learn to generate data from all segments, including the segments with few samples (rare event samples). Moreover, since there is still a representation of the old segments, there should be no catastrophic forgetting.

B. Freezing

The method of *freezing* nodes in neural networks is a known way to keep a feature representation of a batch of data [14]. Freezing prevents parts of the model from being updated again during backpropagation [15]. The motivation behind freezing is to keep some of the representation learned in each segment, thus helping the model remember old classes despite less representation during training new segments. Different freezing methods exist in the literature, including layer freezing, vertical freezing, or gradually changing vertical freezing [16].

In this paper, we leverage incremental freezing of vertical layers. Initially, all weights are unfrozen, but after training of the first segment, some of the weights and biases between the input layer and the first layer are frozen. After training on the second segment, some of the weights between the first and second layers are frozen. This process continues until the training on the data from the last segment is completed. The motivation behind incremental freezing is to keep a representation of each segment equally in the model. This approach enables uniform data generation from all segments, including the rare samples.

IV. Data Collection

This section describes the data used to train IL-GAN. Since we could not find real measurements on URLLC service, we used two datasets, one from factory automation simulations and one real measurement on the video streaming service. We hypothesize that training a generative model to generate data from these datasets accurately is a step in the right direction to solving the problem using the real URLLC dataset.

A. Data Collection and Processing

Here, we explain the simulated and real datasets.

1) Simulated Data: We perform both link-level and network-level simulations for a factory automation scenario (as shown in Figure 2). In the link-level simulations, we modeled a small factory of size $15\times15\times11\,\mathrm{m}^3$ with stochastic blockers within. We assumed a single gNodeB with a height of 10 m in the middle of the factory. We calculate the path gain and 3D channel data for all possible user locations using ray tracing. Then, we import the channel data to a 3GPP compliant network-level simulator, where we simulate the physical and above layers in a multi-cell multi-user network. Besides, we assume numerology 1 from [17], implying that each slot is 0.5 ms long. The traffic ingress to devices from the control application is periodic, with 2 ms period, 2.5 ms delay bound, and size of 64 bytes. We gather two datasets from the simulated data: (i) channel estimate (CE) data, with 10 000 000 data points, and (ii) signal to interference & noise ratio (SINR) and downlink delay data, with 15 000 000 data points.

2) The Real Dataset: The real dataset is gathered from Crawdad [18] and consists of wireless networking data gathered from various video streaming platforms, such as YouTube, Skype, and Google Hangouts. We extract and calculate the interarrival time (IT) and packet lengths (PL) from that data. The dataset has 2 494 707 data points.

B. Data segmentation

In this section, we show how the datasets are segmented. In our data, CE, SINR and IT are continuous metrics, while delay and PL are discrete metrics. The continuous variables are normalized to be between 0 and 1, and the discrete variables are categorized using one-hot encoding.

On the simulated data, the whole CE and SINR dataset are normalized to be between 0 and 1. Regarding delay data, since packets can only be transmitted at the beginning of each slot, samples are separated by 0.5 ms. In our simulations, packets that are received after their corresponding delay bound (i.e., 2.5 ms), we logged an identical large delay and did not differentiate them. Hence, the delay data has four possible values, which we categorize into four categories.

On real data, the whole IT dataset is from 0 s to 100 s, and the majority of the samples are below 0.002 s. Normalizing such a dataset would push most values below 0.00002 seconds. Due to their similarity, the model would have difficulty distinguishing between the values in this range. To circumvent this problem, we individually normalize each segment of IT data. Now values of the first segment are between 0 and 1, instead of between 0 and 0.002. Thus, it is easier for the model to differentiate between these low values.

Table I presents the distribution of all the datasets. The table shows a clear head distribution in the CE dataset. The first segment has more than half of the data points in the dataset, and the first three segments include over 99% of the data. The table also shows that SINR and delay have a clear head distribution, where 50% and 99% of the data are in the first and the first three segments, respectively. In the fourth segment, SINR has 0.001% of the dataset, while the delay data has around 1.3%. The IT and PL datasets have the biggest head

Table I: Distribution of dataset segments

Type	Seg. 1	Seg. 2	Seg. 3	Seg. 4	Seg. 5	Total Samples
CE	54%	23%	22%	0.3%	0.0002%	10 000 000
SINR	49%	38%	12%	0.001%	-	15 000 000
Delay	52%	36%	11%	1.3%	-	15 000 000
IT	91%	8%	1%	0.1%	0.03%	2 494 707
PL	99%	0.4%	0.1%	0.16%	0.2%	2 494 707

Table II: Parameters of the GANs

Parameter	Value	Parameter	Value
n_c	5	Dropout	0.33
Gradient Penalty(λ)	10	Layers	5
Latent Space	100	Learning Rate (η)	0.0002
Optimizer	Adam	Nodes	200
IL-GAN		***cWGAN***	
n_p for CE	[1,3,5, 15,2000]	n_p for CE	5
n_p for SINR & Delay	[1,1,1,10]	n_p for SINR & Delay	5
n_p for IT&PL	[3,5,10,25,500]	n_p for IT & PL	10

distribution of the three datasets. The first segment of the IT data has around 91% of the data, and the first two have around 99% of the data. The first segment of the PL data has around 99% of the data.

Since a failure usually occurs when one or several metrics crosses a certain threshold, we also generate metrics jointly (e.g., generating SINR and delay or IT and PL). For such purpose, we propose putting the data point into the segment of both the features. Therefore, if a data point has SINR from segment one and delays from segment three, that data point will be put into segments one and three. This allows us to generate more than one feature using a single model, but it has the disadvantage of possibly creating an imbalance in the segmented datasets. It will also increase the training time since more data is used in each segment.

V. Evaluations and Experimental Results

A. Performance Metric: Jensen-Shannon Divergence

To evaluate the performance of IL-GAN, we use the JS divergence, a measurement to compare probability distributions. JS divergence enables comparisons of the generated distributions with the true distribution. Although it is based on KL divergence, unlike KL divergence, it is symmetric and bounded to be between $[0, 1]$. The JS divergence, j, is calculated as

$$j(P||Q)=\frac{D_{\mathrm{KL}}(P||M)}{2}+\frac{D_{\mathrm{KL}}(Q||M)}{2}, \tag{3}$$

where $D_{\mathrm{KL}}(\cdot||\cdot)$, P and Q denote KL divergence, the true distribution, and the generated distribution, respectively. Additionally, M=0.5(P+Q). In other words, $j(\cdot||\cdot)$=0 means that the distributions are the same, and $j(\cdot||\cdot)$=1 means that there are no common elements in the distributions.

B. GAN Neural Network Model Aspects

For performance evaluation of our IL-GAN, we consider conditional Wasserstein GAN (cWGAN) from [13] as the baseline for training and generation. For a fair comparison, we trained a cWGAN conditioned on each training segment and leveraged the incremental learning training procedure described in Section III. Since wireless communication data can be both categorical and continuous, the generator needs to be able to generate both types of data. To generate continuous variables, we normalize the data to be between $[0, 1]$ and, consequently, use the Sigmoid activation function in generator's output nodes. We also use the Gumbel-softmax estimator from [19] to generate one-hot encoded categorical variables.

Throughout IL-GAN's training, we randomly sample an equal amount of data from each previous segment as the current segment is trained. Upon completing the training on the data belonging to a segment, we incrementally freeze a portion of the neural network model parameters such that 75% of the nodes in a layer were frozen. The weights from the labels in the conditional GAN are never frozen and are not considered part of the 75% that are frozen.

We use JS divergence as the convergence criterion, both on segment level and IL-GAN level, i.e., we stop the training of either a segment or IL-GAN when the JS divergence does not decrease for a fixed number of iterations, denoted as n_p. Moreover, n_p is set differently for each segment because, as we move forward with the training on different segments, many of the weight becomes frozen, and the number of data points decreases. Therefore, we set n_p to very low values for the first segment (e.g., n_p= 1), while it should be set to higher values for the last segment (e.g., n_p= 2000).

Table II presents our model parameters for both IL-GAN and cWGAN. The n_c and λ parameters are explained in Section II and the values for them are chosen according to [12] and [20] respectively. The learning rate and optimizer are chosen according to [20]. The generator and discriminator neural networks have five hidden layers and 200 nodes in each layer. Dropout is used for regularization to reduce overfitting.

C. Evaluation Methodologies

In this subsection, we compare the generated data from a IL-GAN with the generated data from cWGAN in [13]. We consider two methodologies for evaluating the results:

1) JS method: A similarity measure of the overall generated distribution to the original data, using JS divergence (defined in (3)).
2) Visual method: This visually compares the distribution kernels (e.g., mean) of the last segment distribution which includes the rare samples. For the latter, we generate 1 million samples with IL-GAN and cWGAN.

Besides, we perform three sets of experiments as:

1) CE data generation,
2) Joint SINR and delay generation, and
3) Joint IT and PL generation.

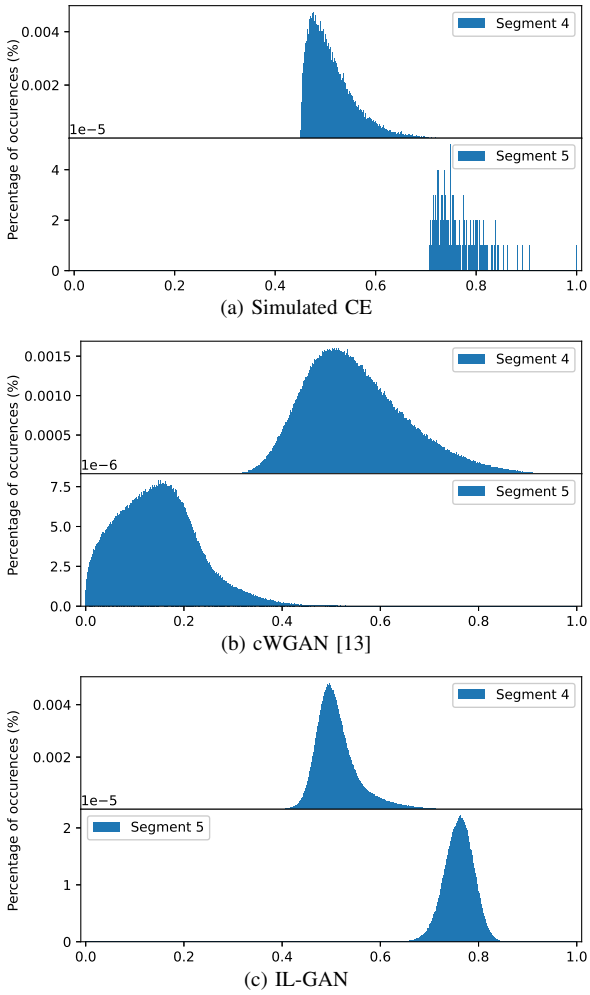
D. Results

1) Channel Estimate: As Table III shows, although cWGAN performs well on the first three segments, its performance dramatically drops on the last two segments. Nevertheless, IL-GAN performs better on all segments, and most notably, compared to cWGAN, it reduces JS divergence significantly on the last two segments.

Figure 3 shows the generated distribution of the last two segments for IL-GAN and cWGAN. On the one hand, the

Table III: JS divergence on the Channel Estimate dataset

Segment	cWGAN	IL-GAN
1	0.0211	0.0193
2	0.0524	0.0107
3	0.0636	0.0184
4	0.27	0.0548
5	1.0	0.264
Average	0.285	0.0777



(a) Simulated CE

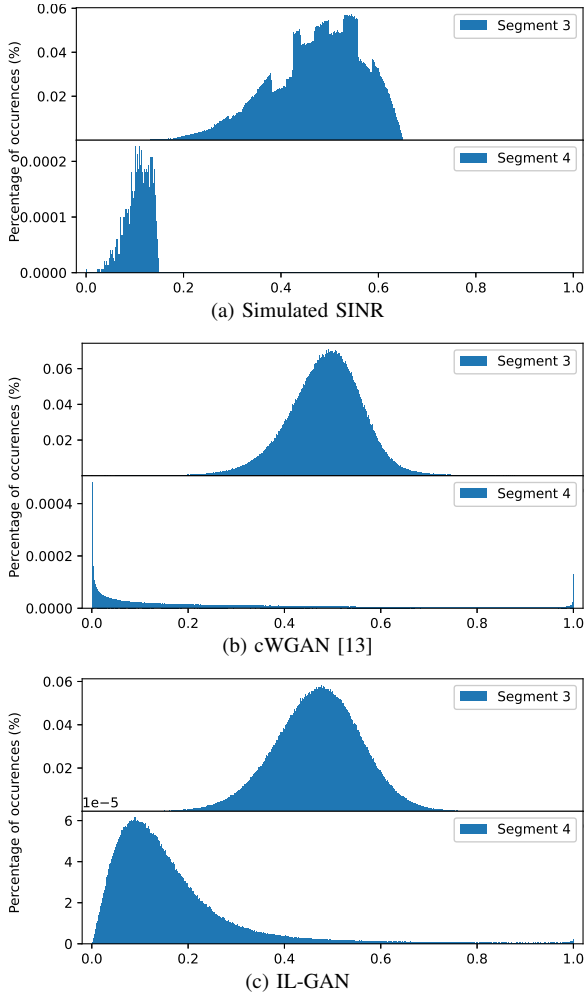
(b) cWGAN [13]

(c) IL-GAN

Figure 3: The distribution of CE rare samples generated by simulations in (a), cWGAN in (b), and IL-GAN in (c).

cWGAN performs poorly on the last two segments, and the distribution of segment 5 is not even in the correct area (This is the reason for JS divergence of 1.0 for cWGAN's segment 5 in Table III). On the other hand, the IL-GAN generated distribution is very comparable to the true distribution of segment 4. For segment 5, although IL-GAN has the distribution in the correct area, it is rough to evaluate the generation of the rare events using solely 159 data points. Nevertheless, IL-GAN predicted the segment 5 distribution reasonably, achieving 74% lower JS divergence over the baseline GAN.

2) SINR & Delay: Table IV shows that both models perform well on the first three segments. As in the previous experiment, IL-GAN shows significant improvement over the baseline GAN on the last segment. Figure 4 shows the generation of the last two segments of SINR. As this figure illustrates, both



(a) Simulated SINR

(b) cWGAN [13]

(c) IL-GAN

Figure 4: The distribution of SINR rare samples generated by simulations in (a), cWGAN in (b), and IL-GAN in (c).

IL-GAN and cWGAN perform well on segment 3, in which 12% of the data exists. Similar to the previous experiment, the distribution of the generated data from cWGAN on segment 4 is not comparable to the real distribution (i.e., it has not learned the true distribution of segment 4). Although the IL-GAN distribution is closer to the true distribution of segment 4, there are still some problems with the generated distribution. A possible reason is a discrepancy in the number of data points in segment 4. Because of the joint generation strategy, described in Section IV, a majority of the SINR samples in segment 4 training are not actually in SINR segment 4.

Table IV: JS divergence on the SINR & Delay dataset

Segment	SINR		Delay	
	cWGAN	*IL-GAN*	*cWGAN*	*IL-GAN*
1	0.0023	0.0093	0.0005	0.0019
2	0.0356	0.047	0.0003	0.0037
3	0.0391	0.0339	0.0018	0.0005
4	0.799	0.444	0.0069	0.0003
Average	0.219	0.133	0.0024	0.0016

3) Interarrival time and Packet Lenght: Table V confirms that the IL-GAN significantly improves upon cWGAN's JS divergence on real measurements on IT and PL, with the

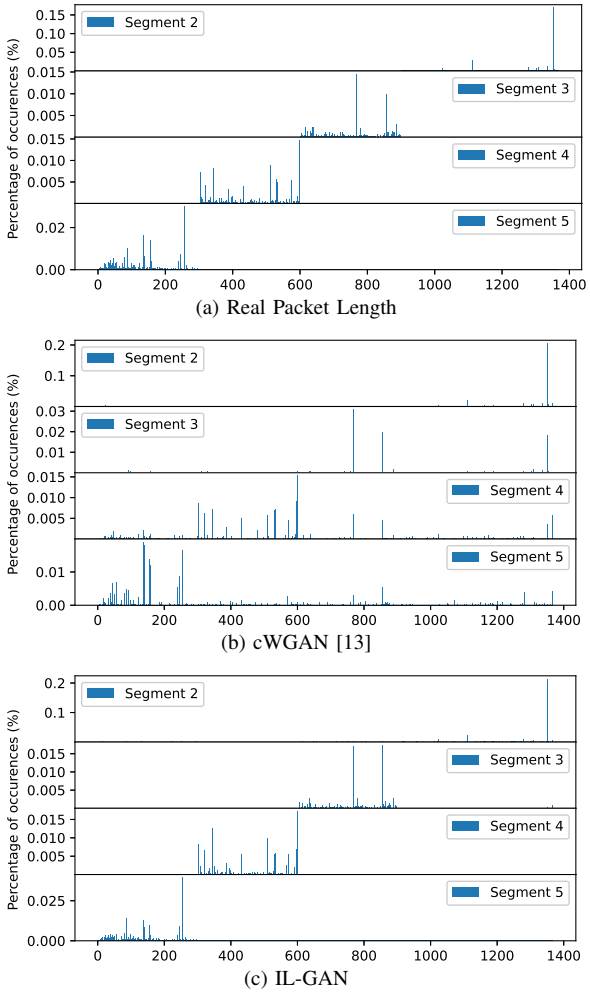


Figure 5: The distribution of PL rare samples from real data in (a), and generated by cWGAN and IL-GAN in (b) and (c), respectively.

maximum observed enhancement on the PL dataset in all of our experiments. As Figure 5 illustrates, for the PL dataset, the cWGAN does not manage to learn the tail distribution in any meaningful sense, while IL-GAN generates data from the correct area in every segment. For the PL dataset, the cWGAN does not manage to learn the tail distribution in any meaningful sense, while IL-GAN generates data from the correct area in every segment.

Table V: JS divergence on the Interarrival time & Packet length dataset

Segment	**IT**		**PL**	
	cWGAN	***IL-GAN***	***cWGAN***	***IL-GAN***
1	0.0279	0.0168	0.0003	0.0003
2	0.0784	0.0492	0.509	0.186
3	0.1457	0.0908	0.707	0.180
4	0.1927	0.1627	0.6403	0.078
5	0.4393	0.2883	0.543	0.036
Average	0.1767	0.1217	0.4797	0.0963

VI. Conclusions

In this paper, we developed IL-GAN framework, in which we leveraged incremental learning on data segments over state-of-the-art GANs to enhance the training and generation of the tail distribution. Our IL-GAN has many applications in wireless communications, for example, (i) in reinforcement learning assisted URLLC service orchestration, where there exist only a few samples on the tail of the distribution to training the agent, or (ii) to serve as a model-free wireless channel transfer function. We evaluated the performance of IL-GAN using real measurements on video streaming over wireless communications and 3GPP compliant simulations on URLLC service for factory automation use case. Our extensive evaluations showed that IL-GAN has superior performance over cWGAN in generating all segments, especially the last one with only a few samples. Although IL-GAN showed significant improvement in the joint generation of metrics over the baseline, our results on joint generation of SINR and delay suggest that there is still room for further developments in such joint generations.

References

[1] *Service requirements for cyber-physical control applications in vertical domains*, 3GPP, TS 22.104 v18.0.0, 2021.

[2] M. Ganjalizadeh *et al.*, "Translating cyber-physical control application requirements to network level parameters," in *IEEE Int. Symp. Pers. Indoor Mob. Radio Commun. (PIMRC)*, 2020.

[3] ——, "An RL-based joint diversity and power control optimization for reliable factory automation," in *IEEE Glob. Commun. Conf. (GLOBECOM)*, 2021.

[4] A. T. Z. Kasgari *et al.*, "Experienced deep reinforcement learning with generative adversarial networks (GANs) for model-free ultra reliable low latency communication," *IEEE Trans. Commun.*, vol. 69, no. 2, pp. 884–899, 2021.

[5] T. Kloek and H. K. Van Dijk, "Bayesian estimates of equation system parameters: an application of integration by Monte Carlo," *Econometrica: J. Econom. Soc.*, 1978.

[6] A. B. Owen *et al.*, "Importance sampling the union of rare events with an application to power systems analysis," *Electron. J. Stat.*, vol. 13, no. 1, pp. 231–254, 2019.

[7] V. Elvira and I. Santamaria, "Multiple importance sampling for efficient symbol error rate estimation," *IEEE Signal Process. Lett.*, vol. 26, no. 3, pp. 420–424, 2019.

[8] I. Goodfellow *et al.*, "Generative adversarial nets," *Adv. Neural Inf. Process. Syst.*, vol. 27, 2014.

[9] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784* [cs.LG].

[10] X. Hu *et al.*, "Learning to segment the tail," in *Proc. IEEE/CVF Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[11] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," 2017 *arXiv:1701.04862* [stat.ML].

[12] M. Arjovsky *et al.*, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017.

[13] Y. Luo and B.-L. Lu, "EEG data augmentation for emotion recognition using a conditional Wasserstein GAN," in *40th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, 2018.

[14] S.-A. Rebuffi *et al.*, "iCaRL: Incremental classifier and representation learning," in *Proc. IEEE/CVF Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

[15] D. E. Rumelhart *et al.*, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: http://www.nature.com/articles/323533a0

[16] L. F. Isikdogan *et al.*, "Semifreddonets: Partially frozen neural networks for efficient computer vision systems," in *Eur. Conf. Comput. Vis. (ECCV)*, 2020.

[17] *NR; Physical channels and modulation*, 3GPP, TS 38.211 v17.1.0, 2022.

[18] S. Sengupta *et al.*, "CRAWDAD dataset iitkgp/apptraffic (v. 2015-11-26)." [Online]. Available: https://crawdad.org/iitkgp/apptraffic/20151126

[19] E. Jang *et al.*, "Categorical reparameterization with Gumbel-Softmax," 2016, *arXiv:1611.01144* [stat.ML].

[20] I. Gulrajani *et al.*, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017.