GENERATIVE ADVERSARIAL NETWORKS FOR DNA STORAGE CHANNEL SIMULATOR

A Seminar Report

Submitted to the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the degree

Bachelor of Technology

in

Computer Science and Engineering

by

ABIN SKARIA

CHN20CS006

Department of Computer Engineering

College of Engineering Chengannur Kerala - 689121

Phone: (0479) 24541125 Fax: (0479) 2451424

Website: www.ceconline.edu

OCTOBER 2023

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ENGINEERING CHENGANNUR

2023 - 24

CERTIFICATE

This is to certify that the report **GENERATIVE ADVERSARIAL NETWORKS FOR DNA STORAGE CHANNEL SIMULATOR** submitted by **ABIN SKARIA** (CHN20CS006), to the APJ Abdul Kalam Technological University in partial fulfillment of the B.Tech. degree in Computer Science and Engineering is a bonafide record of the seminar work carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Smt. Sushitha Susan Joseph
(Seminar Guide)
Assistant Professor,
Dept. of Computer Engineering
College of Engineering,
Chengannur

Smt. Neethu Treesa Jacob
(Seminar Coordinator)
Assistant Professor,
Dept. of Computer Engineering
College of Engineering,
Chengannur

Dr. Manju S Nair
Head of the Department,
Dept. of Computer Engineering
College of Engineering,
Chengannur

DECLARATION

I, ABIN SKARIA hereby declare that the seminar report **GENERATIVE ADVERSARIAL NETWORKS FOR DNA STORAGE CHANNEL SIMULATOR**, submitted for partial fulfillment of the requirements for the award of the degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Smt. Sushitha Susan Joseph, Assistant Professor, Department of Computer Engineering, College of Engineering, Chengannur .

This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources.

I also declare that I have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and / or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Chengannur
04-10-2023

ABIN SKARIA

Abstract

DNA data storage systems have rapidly developed with novel error-correcting techniques, random access algorithms, and query systems. However, designing an algorithm for DNA storage systems is challenging, mainly due to the unpredictable nature of errors and the extremely high price of experiments. Thus, a simulator is of interest that can imitate the error statistics of a DNA storage system and replace the experiments in developing processes. We introduce novel generative adversarial networks that learn DNA storage channel statistics. Our simulator takes oligos (DNA sequences to write) as an input and generates a FASTQ file that includes output DNA reads and quality scores as if the oligos are synthesized and sequenced. We trained the proposed simulator with data from a single experiment consisting of 14,400 input oligo strands and 12,108,573 output reads. The error statistics between the input and the output of the trained generator match the actual error statistics, including the error rate at each position, the number of errors for each nucleotide, and high-order statistics.

Acknowledgement

I take this opportunity to express my deepest sense of gratitude and sincere thanks to everyone who helped me to complete this work successfully. I express my sincere thanks to **Dr. Manju S Nair**, Head of Department, Department of Computer Engineering, College of Engineering Chengannur for providing me with all the necessary facilities and support

I would like to express my sincere gratitude to **Smt. Neethu Treesa Jacob**, Assistant Professor, and **Smt. Nasseena N**, Assistant Professor, Department of Computer Engineering, College of Engineering, Chengannur for their support and co-operation.

I would like to place on record my sincere gratitude to my seminar guide **Smt. Sushitha Susan Joseph**, Assistant Professor, Department of Computer Engineering, College of Engineering, Chengannur for the guidance and mentorship throughout the course.

I would like to give proper credit to the authors of , which was the main reference material I used for this seminar. Most images in this seminar report are from the aforementioned research article.

ABIN SKARIA

Contents

List of Figures

1. Introduction

The burgeoning field of data storage is witnessing a paradigm shift, with DNA storage emerging as a promising next-generation system [1], [2]. Unlike traditional digital storage, which encodes data in a binary format using 0s and 1s, DNA storage leverages a quaternary format utilizing the four nucleotide bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). This innovative approach has the potential to revolutionize data storage by significantly enhancing data density, reaching an impressive 215 petabytes per gram [4]. Moreover, the longevity of DNA storage is remarkable, with fragments of DNA encapsulated in silica capable of preservation for thousands of years at a minimal maintenance cost [5].

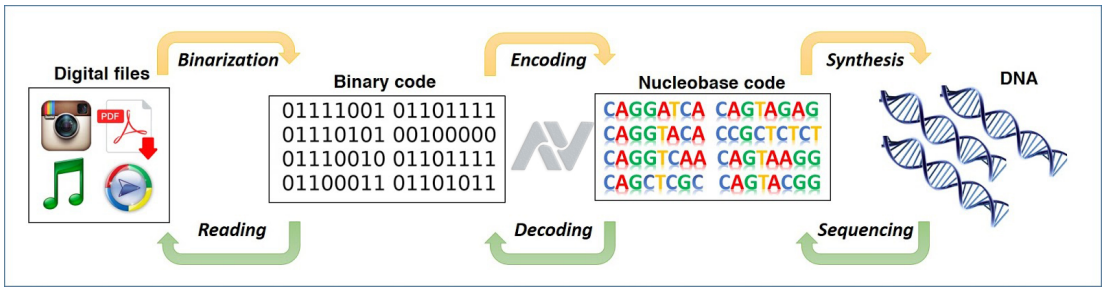


Figure 1.1: DNA STORAGE

However, the efficacy of DNA storage is challenged by the inherent error-proneness of synthesis and sequencing technologies [6]. Accurate and reliable data recovery necessitates the implementation of robust error-correcting codes (ECCs) [4], [7], [8], [9], especially crucial given the asynchronous errors, including insertions and deletions, prevalent in the DNA storage channel. Moreover, the error rates are influenced by higher-order statistics, such as an increase in error rate within consecutive bases,

making precise characterization of the channel statistics and designing effective ECCs a daunting task.

Several researchers have dedicated efforts to devise ECCs suitable for mitigating errors in the DNA storage channel. Blawat et al. [10] proposed a forward error correction scheme, while Erlich and Zielinski [4] combined Reed-Solomon (RS) code and fountain code. Jeong et al. [7] also applied RS codes with an improved decoding technique. Press et al. [8] employed hashing and greedy search to correct asynchronous errors, and Chandak et al. [9] utilized low-density parity-check (LDPC) and BCH codes. Hulett et al. [11] introduced a specific coding scheme tailored for Nanopore sequencing, and Chandak et al. [12] used convolutional codes to correct numerous insertions and deletions.

One major bottleneck hindering the development of ECCs for DNA storage channels is the high cost of synthesis [14], despite the decreasing cost of sequencing [13]. To alleviate this constraint, predictive simulation before conducting real experiments is a valuable approach, proven effective in diverse domains such as wireless communications [15], [16]. Simulations significantly reduce the cost and burden of designing and testing ECCs. However, to date, there is a notable gap in publicly available DNA storage channel simulators, with existing simulators like Antonio et al. [25] predominantly focusing on Nanopore sequencers.Various simulation tools are available for DNA sequencing, simulating different sequencing technologies and error profiles. For instance, ART [26] emulates next-generation sequencers, Flux [27] simulates RNA-seq, pRIS [28], and simLoRD [29] simulate Illumina and PacBio reads, respectively. However, these simulators primarily concentrate on 1st order statistics, such as error probabilities at each location, and are designed for specific sequencing technologies. The rapidly evolving landscape of sequencing and synthesis technologies necessitates versatile simulators that can adapt to diverse experimental setups.

In response to this need, this paper proposes a novel deep learning-based DNA storage channel simulator. This model is based on a generative adversarial network (GAN) [38], enabling the capture of high-order statistics of data in a universal manner,

independent of specific sequencing and synthesis technologies. Notably, the model's training process does not require extensive parameter engineering and depends solely on data statistics. The development of a GAN-based simulator tailored to mimic a noisy channel introduces unique challenges, given the need to match the joint distribution of both input and output. We present a novel GAN structure where the discriminator assesses both input and output to verify input-output statistics, ensuring generated sequences exhibit the necessary randomness to mimic real-world experimental data.

To validate the efficacy of our proposed simulator, we utilized data from a single experiment for model training, demonstrating that extensive data acquisition experiments are unnecessary. Our simulator successfully mirrors insertion, substitution, and deletion error rates, as well as higher-order statistics like the probability of consecutive deletions (2, 3, 4-deletions). Furthermore, the simulator introduces output randomness through multiple weights obtained during training. Applying an ECC technique proposed by Jeong et al. [7] to simulated reads, we observe comparable error-correcting performance to that of real experimental data.

2. Methodology

Generative Adversarial Networks (GANs)-based DNA Storage Channel Simulator comprises a series of strategic steps to effectively model the DNA storage channel and generate simulated sequences that closely resemble experimental data. Initially, a comprehensive dataset is collated, including both input sequences and the corresponding output sequences obtained from experimental studies. Proper data preprocessing and representation are then conducted to ensure the DNA sequences are appropriately formatted for utilization in the GAN framework.

The choice of GAN architecture is a critical decision, and modifications may be made to the standard GAN structure to tailor it to the specific features and patterns present in DNA sequences. The architecture typically consists of a generator network tasked with producing synthetic DNA sequences based on random inputs and a discriminator network responsible for distinguishing between the real experimental sequences and the generated ones. The training phase employs a significant portion of the dataset, often around 80percentage, allowing the GAN to learn and replicate the statistical and structural features of the DNA storage channel. A separate validation dataset is used to monitor the model's performance during training, aiding in the adjustment of hyperparameters and fine-tuning the architecture.

Once the GAN is trained, the generator is utilized to produce simulated DNA sequences, emulating the behavior of the DNA storage channel. These simulated sequences serve as valuable tools for various analyses and assessments, including error rate evaluations and the application of error-correction techniques. The simulator's effectiveness is validated by comparing the simulated sequences with real experimental data, ensuring that it successfully captures the essential characteristics of the DNA storage channel.

3. BACKGROUND

3.1 Deep Learning Models for Sequences

In sequential data processing tasks, a gated recurrent unit (GRU) [39] and Long-short term memory (LSTM) [40], variants of RNN [41], are widely used. Specifically, in language representation tasks, which is the most popular task for sequential data, RNN-based models' performance is not satisfactory.

On the other hand, a line of research combined an encoder and a decoder for the language model [42], [43], [44], [45] but had bottleneck problems. Recently, Vaswani et al. [46] proposed a Transformer, an attention-based network, and showed remarkable performance in various language representation tasks. It leads to pre-trained attention-based models such as BERT [47] and GPT [48], [49], [50], which show state-ofthe-art performances. The pre–LN Transformer [51], which places the layer normalization inside the residual block, improves training with well-behaving gradients [52].

3.2 GAN

3.2.1 Overview

A generative adversarial network (GAN) is a machine learning (ML) model in which two neural networks compete with each other by using deep learning methods to become more accurate in their predictions. GANs typically run unsupervised and use a cooperative zero-sum game framework to learn, where one person's gain equals another person's loss.

The two neural networks that make up a GAN are referred to as the generator and the discriminator. The generator is a convolutional neural network and the discriminator is a deconvolutional neural network. The goal of the generator is to artificially manufacture outputs that could easily be mistaken for real data. The goal of the discriminator is to identify which of the outputs it receives have been artificially created.

Essentially, generative models create their own training data. While the generator is trained to produce false data, the discriminator network is taught to distinguish between the generator's manufactured data and true examples. If the discriminator rapidly recognizes the fake data that the generator produces – such as an image that isn't a human face – the generator suffers a penalty. As the feedback loop between the adversarial networks continues, the generator will begin to produce higher-quality and more believable output and the discriminator will become better at flagging data that has been artificially created. For instance, a generative adversarial network can be trained to create realistic-looking images of human faces that don't belong to any real person.

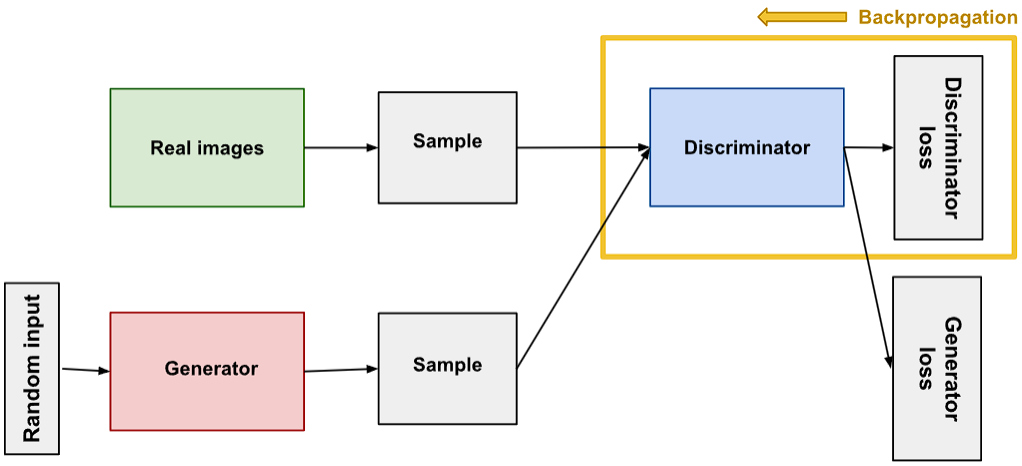


Figure 3.1: GAN Architecture

3.2.2 How GANs work

GANs are typically divided into the following three categories:

- Generative: This describes how data is generated in terms of a probabilistic model.
- Adversarial: A model is trained in an adversarial setting.
- Networks: Deep neural networks can be used as artificial intelligence (AI) algorithms for training purposes.

The first step in establishing a GAN is to identify the desired end output and gather an initial training data set based on those parameters. This data is then randomized and input into the generator until it acquires basic accuracy in producing outputs.

Next, the generated samples or images are fed into the discriminator along with actual data points from the original concept. After the generator and discriminator models have processed the data, optimization with backpropagation starts. The discriminator filters through the information and returns a probability between 0 and 1 to represent each image's authenticity – 1 correlates with real images and 0 correlates with fake. These values are then manually checked for success and repeated until the desired outcome is reached.

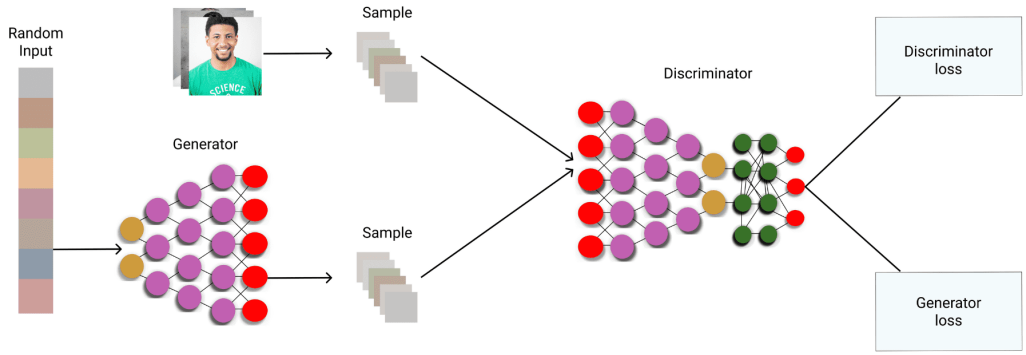


Figure 3.2: Overall working of GAN

A GAN typically takes the following steps:

1. The generator outputs an image after accepting random numbers.
2. The discriminator receives this created image in addition to a stream of photos from the real, ground-truth data set.
3. The discriminator inputs both real and fake images and outputs probabilities – a value between 0 and 1 – where 1 indicates a prediction of authenticity and 0 indicates a fake.

This creates a double feedback loop where the discriminator is in a feedback loop with the ground truth of the images and the generator is in a feedback loop with the discriminator.

4. METHODS

4.1 PROBLEM DESCRIPTION

A DNA storage system stores data in oligonucleotides and reads the stored sequence from synthesized multiple oligos. The goal of the simulator is to imitate the sequential processes of the DNA storage system from the input oligo sequence to the output read sequence.

Let X = {A,C, G, T} be a set of nucleotide bases, and Xe = {A,C, G, T , _ } be a set of extended symbols where '-' is a placeholder that corresponds to inserted or deleted symbols. Further, let $Q \subset [0, 1]$ denote the set of normalized quality values. The input of the simulator O is a nucleotide sequence of length n, which we call the oligo: O = O1 O2...On where $Oi \in X$ for $1 \leq i \leq n$. Then, the simulator's output consists of length M of DNA sequence R, which we call the read, and corresponding quality scores Q: R = R1 R2...RM Q = Q1 Q2...QM where $Rj \in X$and $Qj \in Q$ for $1 \leq j \leq M$. The length of the output read sequence (and quality values) is M, which is a random variable due to asynchronous (insertion and deletion) errors. Let Y = (R, Q) denote the combined output of the simulator, where Yj = $(Rj, Qj) \in X \times$ Q for $1 \leq j \leq M$.

4.2 SIMULATOR

We propose a read generator Gread followed by a quality score (qscore) generator Gqs. Given an input oligo O, the read generator Gread(O) outputs a read R, and Gqs(O, R) generates a quality score Q based on oligo and read sequences. This is based on the nature of quality scores which indicates the confidence level of the read. Roughly

speaking, if there is a mismatch between the bases of the oligo and the read, it is more likely to have a lower quality score.

Due to the notoriously challenging nature of the asynchronous error and unbalanced error rates between substitution errors and others, we could not train a single generator that handles all three types of errors simultaneously. Instead, we divide the read generation procedure into three steps: insertion generation, substitution generation, and deletion generation. We call each intermediate sequences by Xins, Xsub, and Xdel, i.e., O → Xins → Xsub → Xdel → R (we will specify later why Xdel and R are different). Let L denote the length of the intermediate sequence, which is a random variable and may be distinct from n and M. More precisely, intermediate sequences are given by

$$
\begin{aligned}
X_{\text{ins}} &= X_{\text{ins},1} X_{\text{ins},2} \cdots X_{\text{ins},L} \\
X_{\text{sub}} &= X_{\text{sub},1} X_{\text{sub},2} \cdots X_{\text{sub},L} \\
X_{\text{del}} &= X_{\text{del},1} X_{\text{del},2} \cdots X_{\text{del},L}
\end{aligned}
$$

where Xins,k , Xsub,k , $Xdel, k \in Xe$ for $1 \leq k \leq L$. Due to the insertion, we have $n \leq L$, and similarly we have $M \leq L$ because of deletion errors. Then, the read generator is composed of three generators Gins, Gsub, and Gdel, which introduce insertion, substitution, and deletion errors. Note that the sequence O and Xins may have different lengths due to insertion errors. However, the sequence lengths of Xins, Xsub, and Xdel are always the same since it has a placeholder symbol '-'. Finally, a trimmer removes '-' symbol from Xdel, and we obtain the output read sequence R.

The error-contained input-output pairs are rare compared to error-free pairs in the dataset. In such a case, the generator trained by the GAN [38] framework may encourage the trivial generator, which outputs the input without any errors. In this work, we train the generator only on the error-contained pair to focus on the error statistics. Then, we introduce a profile, a binary vector p = $(pi, ps, pd) \in 0, 1^3$ that indicates whether each type of error occurs or not. More precisely

$$X_{\text{ins}} = \begin{cases} G_{\text{ins}}(O) & \text{if } p_i = 1 \\ O & \text{if } p_i = 0 \end{cases}$$
$$X_{\text{sub}} = \begin{cases} G_{\text{sub}}(X_{\text{ins}}) & \text{if } p_s = 1 \\ X_{\text{ins}} & \text{if } p_s = 0 \end{cases}$$
$$X_{\text{del}} = \begin{cases} G_{\text{del}}(X_{\text{sub}}) & \text{if } p_d = 1 \\ X_{\text{sub}} & \text{if } p_d = 0. \end{cases}$$

While training, we estimate the probability distribution of the profile vector based on occurrences. In the simulation step, for each oligo input, the simulator samples a random profile vector based on the empirical distribution of the profile vector and feeds it to all generators (Gins, Gsub, and Gdel).

There are two main reasons why we use three separate generators instead of a single generator:
1) different nature of synchronous and asynchronous errors, and
2) biased error statistics where substitution errors dominate the other types of errors. Thus, we design generators that can solely focus on each type of error, which is more efficient and robust in training compared to a single generator network. We discuss more on GAN training in the presence of a profile vector

The qscore simulator Gqs takes an oligo and read pair (O, R) as an input. It first aligns the oligo and the read which we call Oaligned and Raligned, respectively. Then, it produces quality scores corresponding to an aligned read. Similar to the read generation, since the error rarely occurs in Gread and the qscore statistics are significantly different when there exists an error, we introduce two generators G (g) qs and G (e) qs . If there is no error between generated read and the input oligo, we apply an error-free generator G (g) qs , and we apply G (e) qs if there is an error. In other words,

$$G_{\text{qs}}(O, R) = \begin{cases} G_{\text{qs}}^{(g)}(O, R) & \text{if } O = R \\ G_{\text{qs}}^{(e)}(O, R) & \text{if } O \neq R. \end{cases}$$

The entire structure of the proposed simulator is described in Figure 4.1

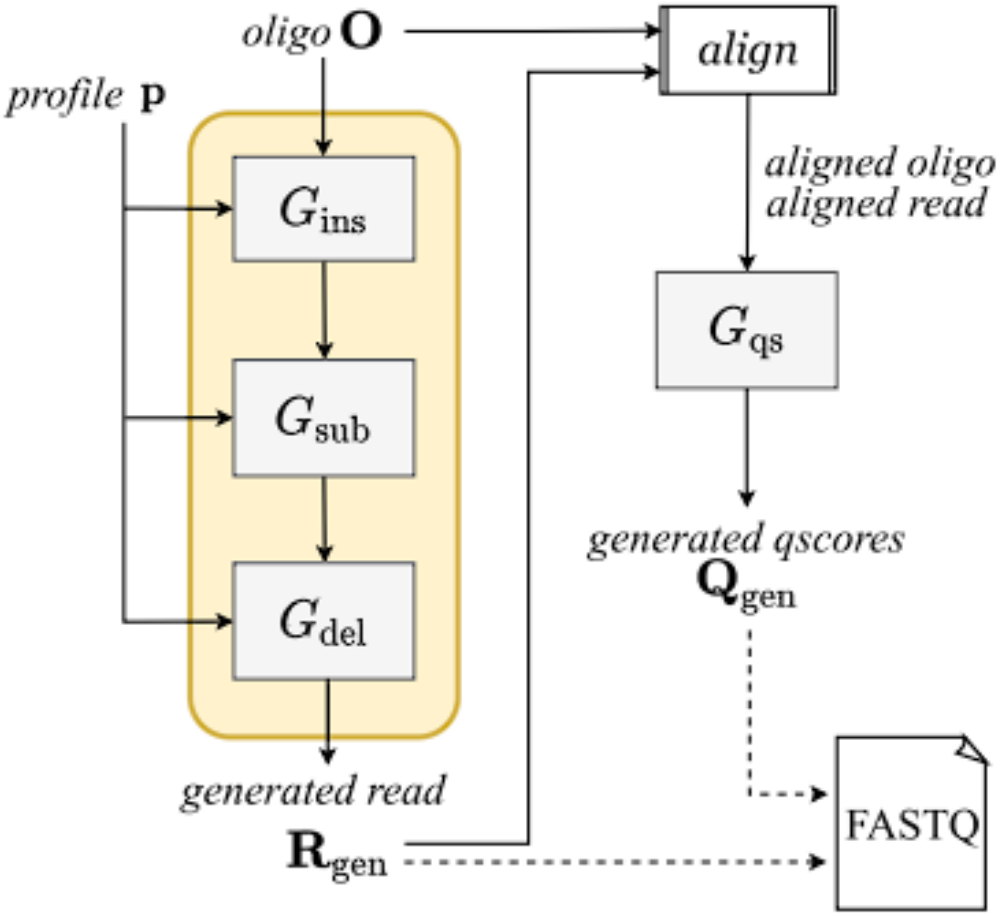


Figure 4.1: Overall structure of the simulator.

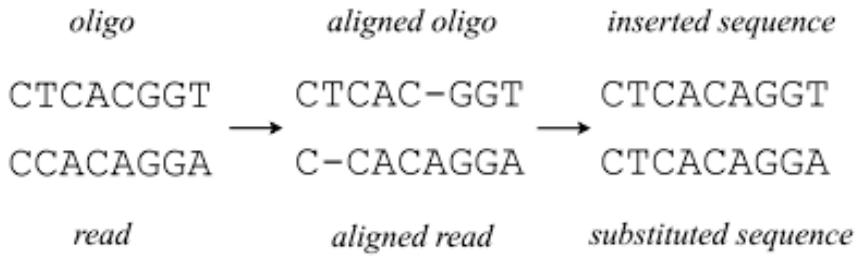
5. TRAINING GENERATORS USING GAN

5.1 GANs FOR READ GENERATOR

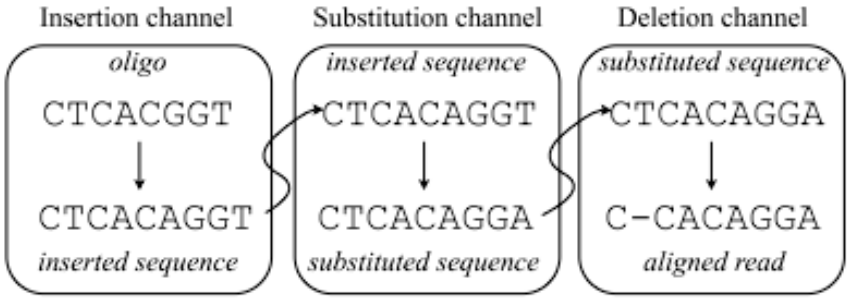
We train Gins, Gsub, and Gdel using three separate GANs (insGAN, subGAN, and delGAN for Gins, Gsub, and Gdel, respectively). However, we need to define each generator's input and output since the intermediate sequences (Xins, Xsub, and Xdel) are not provided. The insGAN requires O and Xins, the subGAN needs Xins and Xsub, and finally, the delGAN is trained on Xsub and Xdel.

To achieve intermediate sequences, we first align the sequences to get Oaligned and Raligned of the same lengths for given oligo O and read R. Then, we obtain an inserted sequence Xins by replacing '-' in Oaligned with the base of the same index of Raligned. Similarly, a substituted sequence Xsub is obtained by replacing '-' in Raligned with the base of the same index of Oaligned. Finally, we have Xdel = Raligned, a deleted version of the substituted sequence. An example of the above data processing is described in Figure 5.1 In this example, oligo O = CTCACGGT and read R = CCACAGGA are given. First, we align two sequences and get aligned oligo Oaligned = CTCAC-GGT and aligned read Raligned = C-CACAGGA. Then, we obtain Xins = CTCACAGGT by replacing '-' in Oaligned with 'A' (the 6th base of Raligned), and obtain Xsub = CTCACAGGA by replacing '-' in Raligned with 'T' (the 2nd base of Oaligned). The deleted sequence Xdel is simply C-CACAGGA which is identical to Raligned.

The goal of GAN training is to get a generator that produces similar error patterns as if it is obtained from the actual experiments. The training procedure of GANs



(a) Intermediate sequences. We first align the input oligo and the output read. Then, we replace place holders '−' by a corresponding bases.



(b) Insertion, substitution, and deletion channels. We model the DNA storage channel by a composition of these three channels.

Figure 5.1: Dataset generation for insertion, substitution, and deletion channels.

(insGAN, subGAN, and delGAN) are the same; The generator takes an input sequence and produces an output sequence, where the discriminator checks both input and output sequences then determines whether the output sequence is from real experiments or from the generator. We use the framework of WGAN-GP [58] that minimizes the Wasserstein distances between distributions with a gradient penalty that stabilizes the training. This training procedure of GANs (insGAN, subGAN and delGAN) is shown in Figure 5.2

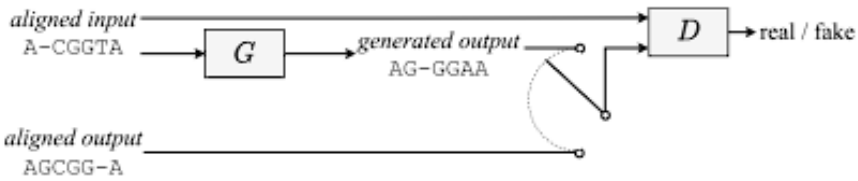


Figure 5.2: Structure of GANs for the read simulator.

5.1.1 insGAN

The critical component of insertion generator Gins is a gated recurrent unit (GRU) [39]. Since GRU takes input sequentially, handling asynchronous errors such as insertion is suitable. Also, GRU has lighter architecture than Transformer, but it can learn short-term dependency effectively and is more appropriate to generate realistic consecutive errors. We stack three bidirectional GRU [87] layers of 64 hidden units, which effectively extract the sequence features. The bidirectional GRU layers are followed by a single fully-connected layer with a softplus activation function. Unlike subGAN or delGAN, the input and output of the insGAN may have different lengths of sequences.

We denote Dins as the discriminator corresponding to the insertion generator Gins. It takes the generator's input and output (O, Xins) as input to check whether the input and output show proper channel statistics. The discriminator Dins is based on a 1-dimensional convolutional neural network (1DCNN) [88], which investigates the joint distribution of input O and output Xins. We use various CNN kernels and multiple layers to capture the high-order dependencies. The discriminator takes stacked input and output sequences as input. Notably, we exclude batch normalization and dropout from the discriminator to stabilize the GAN training. Finally, 1DCNN layers are followed by a fully connected layer without an activation function. The more details of the discriminator are the following. We stacked ten layers of 1D-CNN with eight hidden dimensions. The CNN contains 5×5 kernels and zero padding.

Note that Gins is a generator that introduces the insertion error, and therefore we construct a dataset with erroneous pairs only. I.e., all input and output pairs (O, Xins) in the dataset for insGAN contain insertion errors. This is also the case for subGAN and delGAN where the datasets consist of pairs with substitution errors and deletion errors, respectively

5.1.2 subGAN

The substitution generator Gsub is based on the Transformer model [46]. Since the substitution error is the synchronous error and is the most dominant type of error, the proposed model focuses on higher-order statistics with an attention mechanism.

The generator Gsub takes Xins as an input and outputs Xsub. The proposed model contains a learnable positional encoding and three encoder layers. An encoder layer consists of two multi-head attention layers, followed by a feed-forward layer with 64 hidden dimensions and layer normalization layers. Finally, the generator outputs a sequence processed by a fully connected layer with a soft plus activation function. The discriminator Dsub adopts a similar construction to insGAN, ten layers 1D-CNN with 128 hidden dimensions, 5×5 kernel, and a single zero padding.

5.1.3 delGAN

The deletion generator Gdel and discriminator Ddel are identical to that of insGAN. The generator Gdel has three layers of bidirectional GRU and 64 hidden dimensions, where Ddel has ten layers of 1D-CNN with 8 hidden dimensions, 5×5 kernel, and a single zero padding. However, since the input Xsub and Xdel have the same sequence lengths due to place holder symbol '-', it is much easier to learn the statistics of the deletion channel than the insertion channel.

5.1.4 RANDOMNESS IN READ GENERATOR

A distinctive point of the GAN-based simulators apart from the other GANs is the stochasticity of the generator. To achieve the output variability, we first introduce an additional random number for each input base. We also store ensembles of generators obtained during a single training procedure but in different epoch numbers. Because of instability in the minimax training of GAN [56], [89], the generator after each epoch varies meaningfully. Thus, the simulator randomly selects a generator from the ensemble and then produces the output. We provide experimental results and verify the output variability

5.2 GAN FOR QUALITY SCORE GENERATOR

The quality score generator takes aligned oligo and read pair (Oaligned, Raligned) as an input. We train two quality score generators (G (g) qs , G (e) qs) separately; a generator for error-free sequence pairs and a generator for error-contained sequence

pairs. The training framework is the same for both cases.

Given the dataset consists of oligo, read, and quality scores, we align the oligo and read sequences where the quality scores follow the aligned reads. This provides aligned oligo Oaligned, aligned read Raligned, and aligned quality scores Qaligned. Note that there is no quality score corresponding to the deleted base. As described in the following example,we fill such a gap (2nd base in the example) with the nearest quality value (1st quality score '!' in the example).

$$O_{\text{aligned}}: \quad \texttt{CTCAC-GGT}$$
$$R_{\text{aligned}}: \quad \texttt{C-CACAGGA}$$
$$Q_{\text{aligned}}: \quad \texttt{!!@@*(((}$$

The quality score discriminator takes triplet of aligned oligo, aligned read, and aligned quality scores (Oaligned, Raligned, Qaligned) as an input. Then, it determines whether the quality scores are generated or not. We also follow the framework of WGAN-GP [58] that minimizes the Wasserstein distances with a gradient penalty, which is resistant to the mode collapse problem. The training procedure for quality score GAN is described in Figure 5.3

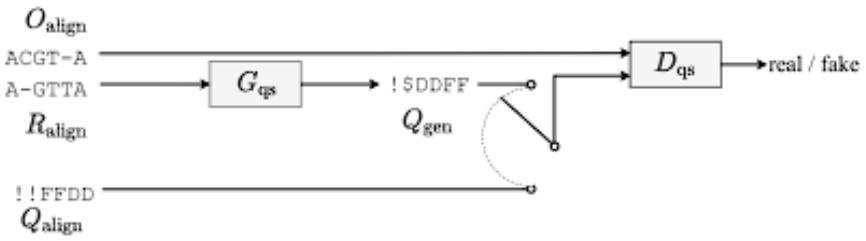


Figure 5.3: Structure of quality score GAN for the quality score simulator.

5.2.1 QSCORE GAN

The quality score generator Gqs has a similar structure with Gins, which consists of two bi-directional GRU layers with 64 hidden dimensions. The GRU layers are followed by the fully connected layer and hyperbolic tangent activation function. Note that the generator produces normalized quality scores between 0 and 1. The quality score discriminator Dqs consists of ten 1D-CNN layers with 8 hidden dimensions, 5 ×

5 kernels, and a single zero padding. A fully connected layer that follows CNN layers generates the scalar value to compute Wasserstein distance.

Similar to generators in Gread, the generator takes additional random numbers as input corresponding to the base. It introduces the randomness of generated quality scores. While training, we added L1-regularization for quality score GAN for stability

6. EXPERIMENT

6.1 DATASET

In the experiment, we use the same oligo design proposed by Jeong et al. [7]. There are 18,000 oligos of length 152 in the dataset, where all oligos have balanced GC contents (base G and C proportion is between 45 to 55 percent) and homopolymer run length is limited by 3. We obtain the corresponding FASTQ file that contains only reverse direction results from the single experiment, consisting of reads and quality scores. We wrote the data using Twist Bioscience synthesis and read it with the Illumina Miseq Reagent v3 kit.

Given the FASTQ file, we match each read to the nearest oligo based on edit distance. After discarding noisy reads (i.e., reads with Ns or distance to nearest oligo is larger than 5), we obtain 15,126,429 reads. Note that a single oligo can be sequenced multiple times (see Figure 6.1).

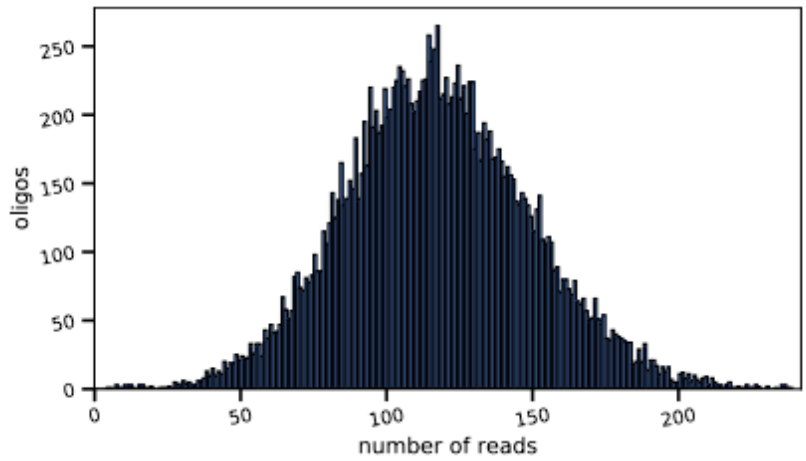


Figure 6.1: Distribution of matched reads per oligos.

Then, the oligos (and the matched reads) are divided into two sets; the training dataset consists of 14,400 oligos and corresponding 12,108,573 reads and the test dataset consists of 3,600 oligos and corresponding 2,999,656 reads. For insGAN, subGAN, and delGAN, we extract the error-contained sequence pair from the training dataset where the edit distance is nonzero and generate training datasets for insGAN, subGAN, and delGAN as described in Figure 5.1 Also, we trimmed aligned sequences at 145 lengths as the later part of the sequence is considered noisy. For qscoreGAN, we generate an error-free dataset for G (g) qs and the error-contained dataset for G (e) qs . We then run the simulator with oligos in the test dataset. The error statistics of simulated data and an actual experiment are compared. To train models with sequential data in batch, we need to match the lengths of sequences. We introduce additional symbols for sequences to match the length; symbols S and E represent the start and the end of the sequence, and symbol P is padded to match the length. Then, we apply the one-hot encoding to sequences, where the one-hot vector is an 8-dimensional vector (ACGT-SEP). For insGAN, subGAN, and delGAN, an additional uniformly distributed random number is added to a one-hot vector to induce randomness of the simulator, as we discussed. For qscoreGAN, the oligo and read sequences are aligned, then covert to one-hot encoded vectors. Then, encoded vectors are stacked, and we also add a random number to form a 17-dimensional vector.

6.2 TRAINING

We train WGANs (insGAN, subGAN, delGAN, and qscoreGAN) with gradient penalty. For better convergence results, we adjust the learning rate in the subGAN according to the number of epochs, i.e. multiply the learning rate by 0.1 every 40,000 batches.

For the randomness of read generator, we obtain the trained weights of 2500 to 2509 epochs for insGAN, 500 to 509 epochs for subGAN, and 1300 to 1309 epochs for delGAN. From the training dataset, the estimated profile vector distribution is the following: the percentage of error-free pairs is 84.253%, the percentage of insertion is 0.945, substitution is 10.646, deletion is 3.572, insertion with substitution is 0.101,

insertion with deletion is 0.039, substitution with deletion is 0.437, insertion with substitution and deletion is 0.007.

7. RESULTS

To the best of our knowledge, our work is the first attempt to design a DNA storage channel simulator, while other simulators focus only on DNA sequencing. Thus, in this section, we mainly provide the error statistics of the proposed simulator and compare them with the error statistics of actual data.

7.1 READ GENERATORS: ERROR STATISTICS

To evaluate the performance of the proposed simulator, we first investigate the error statistics of reads. Figure 7.1-(a) presents the insertion error rates, the number of insertion errors at each index divided by the total number of reads with insertion. Similarly, Figure 7.1-(b) and Figure 7.1-(c) show the substitution and deletion error rates, respectively. Although the error probability is not given explicitly, the generated sequences follow the distribution of positional error (1st order statistics).

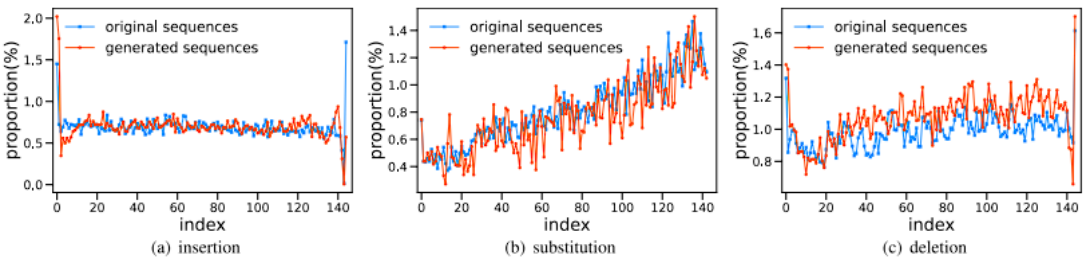


Figure 7.1: Error rates of insertion, substitution, and deletion channels at each index

We also analyze the consecutive insertion and deletion error rates to investigate the high-order statistics of the generator. Since an insertion error is rare, we focus on consecutive deletion errors. Figure 7.2 shows the rate of consecutive deletion errors (1-deletion to 4-deletions) starting at each index. Table 1 compares the ratio of

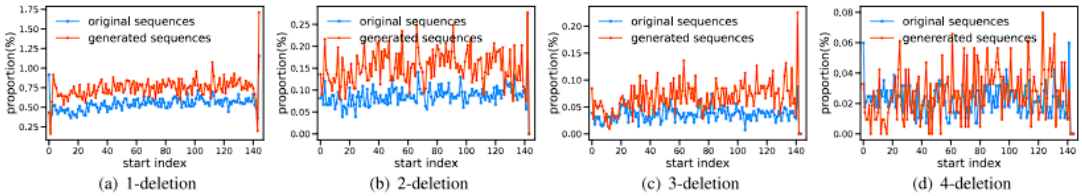


Figure 7.2: Error rates of consecutive deletion errors at each index.

consecutive errors for both insertion and deletion errors, which are averages of the rate of consecutive errors. The results indicate insertion and deletion errors do not occur independently; for example, the rate of 2-deletion is not a square of the rate of 1-deletion. The proposed generator mimics the consecutive error rates of the experiment, which cannot be achieved with predefined error probabilities.

Consecutive Errors	Insertion(%)		Deletion(%)	
	original	generated	original	generated
1	0.68505	0.68481	0.5449	0.6628
2	0.00161	0.00006	0.0858	0.0958
3	0.00014	0.00006	0.0356	0.0542
4	0.00006	0.00002	0.0222	0.0274

Table 1. Consecutive error rates of insertion and deletion channels.

Insertion			Deletion		
Error	Original	Generated	Error	Original	Generated
$- \to A$	7522	6698	$A \to -$	13031	12581
$- \to C$	4238	4864	$C \to -$	13719	13256
$- \to G$	18464	15572	$G \to -$	13607	12468
$- \to T$	3129	5656	$T \to -$	13460	14070

Table 2. Comparison of error types (base pair) of insertion and deletion channels.

We further count the number of errors for each base, illustrated in Table 2 and 3. In Table 2, we count the number of insertion error bases among 33,158 oligo-read pairs from the test dataset, and count the number of deletion error bases among 37,668 oligo-read pairs from the test dataset. In Table 3, we count the number of substitution error bases among 19,170 oligo-read pairs from the test dataset. Errors of three bases (A, C, and T) occur uniformly in the experiment, whereas base G is inserted more often than

Substitution					
Error	Original	Generated	Error	Original	Generated
$A \to C$	752	802	$G \to A$	4862	4422
$A \to G$	669	796	$G \to C$	880	880
$A \to T$	898	1040	$G \to T$	7229	6983
$C \to A$	459	710	$T \to A$	724	838
$C \to G$	145	196	$T \to C$	3433	2237
$C \to T$	2206	2255	$T \to G$	543	751

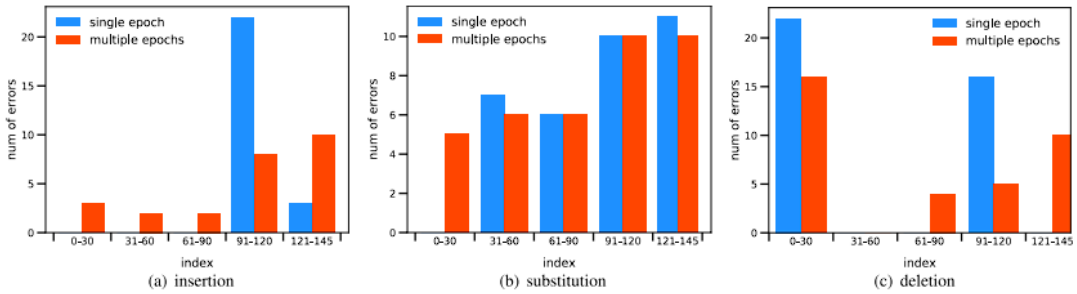
Table 3. Comparison of error types (base pair) of substitution channels.

other bases, and deleted bases have occurred nearly equally. In the case of substitution errors, the frequency of each base error type varies widely. Among them, the number of base G changing to base T is the most, and the number of base C changing to base G is the least. The proposed simulator also captures these per-base error statistics.

7.2 READ GENERATORS: ERROR DISTRIBUTIONS

Since our goal is to simulate the noisy channel, the simulator's output should be random. More precisely, we expect our simulator generates various sequences for the same input, and the distribution should match the experiment. Recall that we added random numbers for each input base and used the ensemble of generators (trained with a different number of epochs) to induce a probabilistic behavior of the simulator. For the insertion generator Gins, we use 11 generators with trained weights at epochs from 2400 to 2410, where we use 11 generators for the deletion generator Gdel with trained weights at epochs from 1290 to 1300. In the substitution generator Gsub, the same 11 generators are used, and the weights are trained at epochs 500 to 510 due to the faster convergence obtained by using the Transformer structure. To verify that the read simulator generates output diversity, we generate 25 reads from the fixed oligo. Table 4 and Figure 7.3 show the analysis of the simulated reads by the insertion generator Gins, the substitution generator Gsub, and the deletion generator Gdel. Table 4 shows that generators with different weights produce significantly various error patterns, especially in insertion.

We also investigate the error occurrences when the input is fixed to a given sequence.



(a) insertion (b) substitution (c) deletion

Figure 7.3: Comparison of error distribution (index of erroneous base) between generated reads using single weight and using multiple weights.

Epoch	Insertion				Epoch	Substitution				Epoch	Deletion			
	A	*C*	*G*	*T*		*A*	*C*	*G*	*T*		*A*	*C*	*G*	*T*
2400	-	1	22	2	500	9	4	16	9	1290	9	4	10	-
2401	-	-	8	17	501	4	3	16	8	1291	1	3	26	17
2402	5	5	-	15	502	5	3	19	3	1292	1	6	20	11
2403	5	5	-	15	503	6	5	24	3	1293	1	3	20	30
2404	-	2	23	25	504	4	7	25	4	1294	8	4	23	24
2405	-	-	25	-	505	4	3	26	6	1295	8	1	28	25
2406	-	-	24	1	506	6	2	22	1	1296	2	9	10	11
2407	-	-	-	25	507	3	6	23	5	1297	-	15	4	7
2408	-	-	-	25	508	4	3	16	3	1298	6	2	21	22
2409	-	-	7	18	509	5	5	18	3	1299	11	4	21	15
2410	-	-	3	22	510	7	1	19	5	1300	5	3	15	19

Table 4.The number of erroneous bases of 25 generated reads from a single fixed input oligo sequence.

The comparison is between generated sequences using single weight configuration and sequences using 10 different weights from 10 epochs. Figure 7.5 implies that the generation with multiple weights has more variability in error patterns. Moreover, Figure 7.4 shows the error statistics of generated sequences from the fixed oligo, where we generated 25 sequences to test Gins, 100 sequences to test Gsub, and 50 sequences to test Gdel. Note that we produce output read sequences from a single input sequence, the exact error location does not perfectly match due to the random nature of the generator. However, it shows that the error statistics of the generated sequence are similar to the true error statistics (from an actual experiment) in terms of both error location and bases.

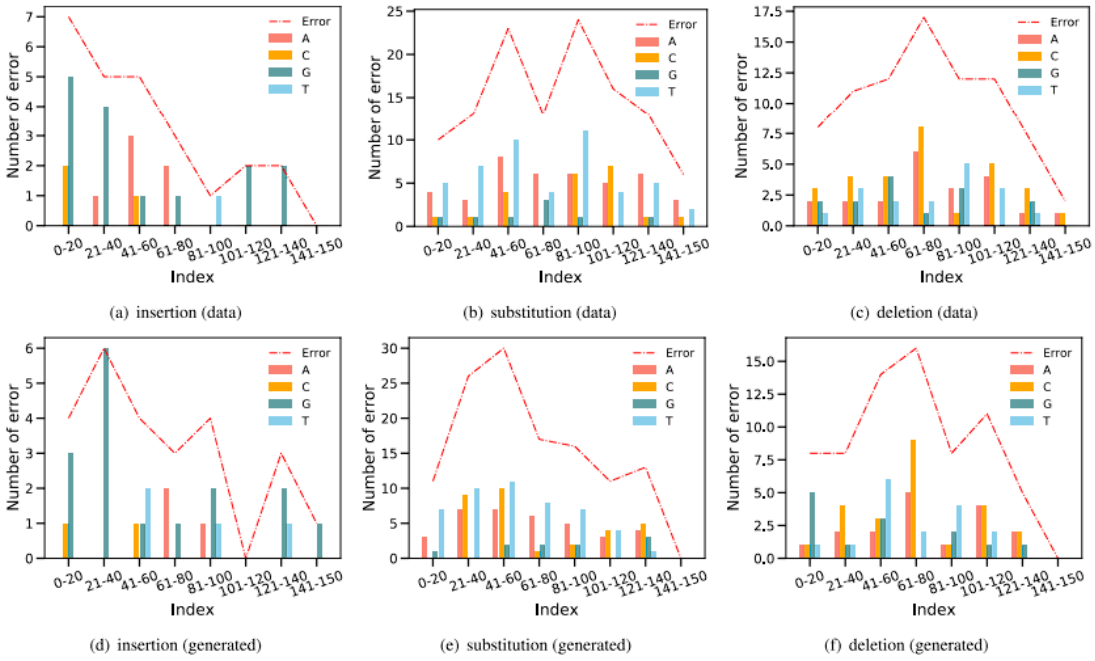


Figure 7.4: Randomness of generated reads from a single fixed oligo at insertion, substitution, and deletion channels.

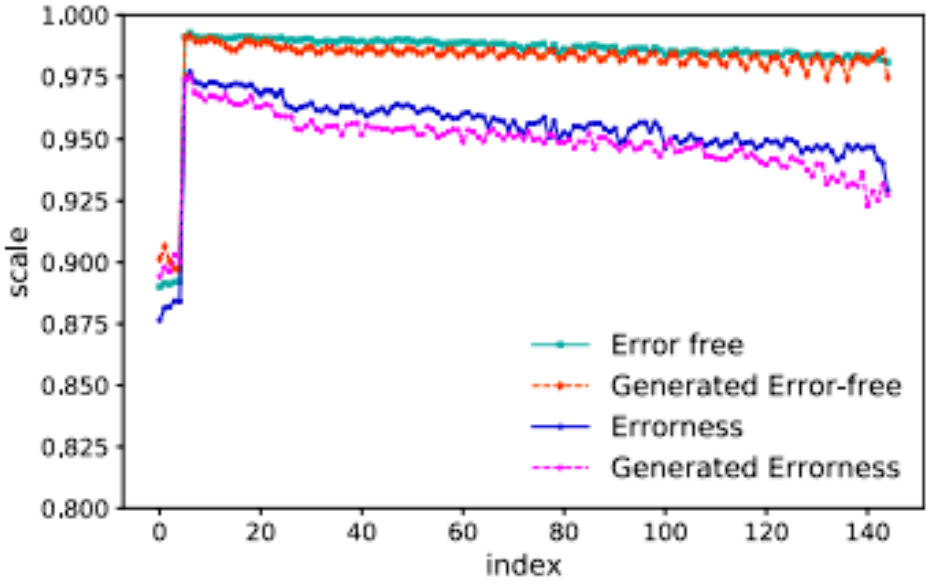
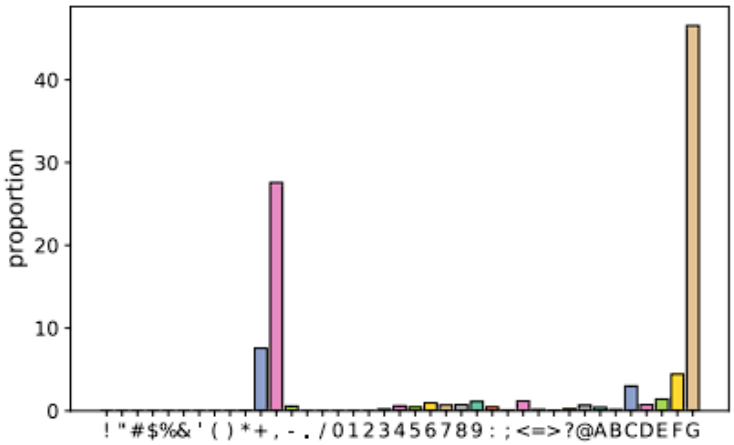


Figure 7.5: Mean of quality scores at each index.

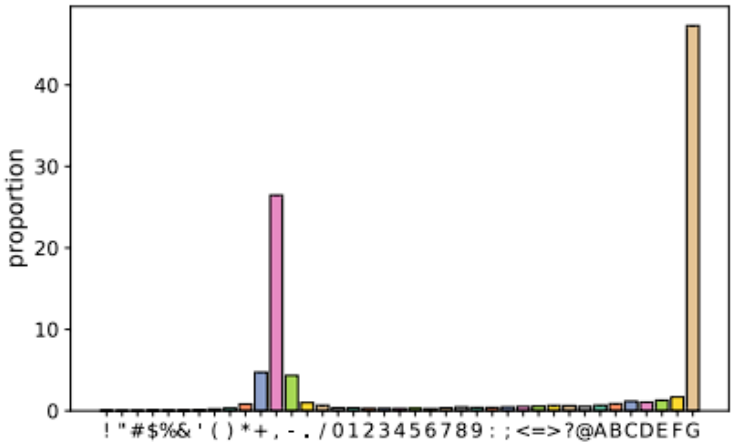
7.3 QSCORE GENERATOR

Figure 7.5 shows the positional mean quality scores from the experiment and the simulation. In the experiment, the quality score decreases except for the first few indices, and the generator also mimics this behavior. Figure 7.6 shows that the output

distributions of quality score generators also match the experimental results. For error-free bases (when oligo and read coincides), the corresponding quality score is mostly G, which is the highest value. However, in the presence of error, a certain portion of lower quality scores (such as ',' and '+') exists. Our quality score simulator captures this characteristic. Note that the original GAN often suffers from mode-collapse where the generator only recovers a single mode of the target distribution. However, thanks to WGANGP, we successfully recovered two modes (one on 'G' and another around ',' and '+').



(a) Quality score corresponds to substituted bases (Actual)



(b) Quality score corresponds to substituted bases (Generated)

Figure 7.6: Distribution of corresponding quality scores when the substitution error occurs.

7.4 APPLICATION OF ECC ALGORITHM

We test the error correcting code (ECC) to simulated reads and the actual experimental reads. More precisely, we apply ECC that proposed by Jeong et al. [7], where the author combined Hamming distance-based clustering, Reed-Solomon (RS) based error correcting techniques, and quality-scorebased decoding. The authors seek a minimum number of read samples where the ECC algorithm perfectly recovers the oligo input. With experimental reads, the minimum number of reads was 86000, where 200 out of 200 decoding trials succeeded. We also applied the same ECC technique to the simulated data from the same oligo inputs. The minimum number of generated reads was 84000, where 10 out of 10 decoding trials succeeded. Furthermore, Table 5 also

	Simulation	Experiment
Number of required reads for 100% decoding success	84000	86000
Number of all clusters	17824.2	17634.7
Number of size-1 clusters	2026.2	2054.2
Ratio of size-1 clusters to all clusters	11.4%	11.6%
Ratio of size-1 clusters to all sequence reads	2.6%	2.8%

Table 5. Clustering and decoding statistics at the smallest random samples for both simulation (ours) and experiment results

shows the other statistics provided by the ECC algorithm, including cluster-related statistics. The ECC algorithm produces about the same number of clusters for both experimental and simulated data. Interestingly, the number of size-1 clusters is also identical, which is a crucial error factor in the cluster-based ECC algorithms. These results imply that the proposed simulator produces similar reads to the experiment.

8. Conclusion

We presented the generative adversarial network framework for the DNA storage channel simulator. Our simulator generated sequences and quality scores with similar error statistics to an actual experiment. The proposed framework is purely data-centric that does not depend on specific sequencing or synthesis technologies.

However, our model does have limitations. First, learning data from Nanopore sequencing may be challenging due to the training complexity of Transformer models. We leave developing a training-efficient network for the longer sequence as crucial future work. Another interesting direction is to manage the GAN training. It is well-known that the GAN is extremely hard to train, and our framework also requires fine-tuned hyperparameters to train the generator successfully. We believe that the well-behaving GAN architecture allows us to apply the technique to a more challenging setting, including long sequence length, biased error probabilities, high-order dependencies, and a massive amount of data.

References

[1] [1] G. M. Church, Y. Gao, and S. Kosuri, “Next-generation digital information storage in DNA,” Science, vol. 337, no. 6102, p. 1628, 2012.

[2] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, “A DNA-based archival storage system,” in Proc. 21st Int. Conf. Architectural Support Program. Lang. Operating Syst., Mar. 2016, pp. 637–649.

[3] L. Ceze, J. Nivala, and K. Strauss, “Molecular digital data storage using DNA,” Nature Rev. Genet., vol. 20, no. 8, pp. 456–466, Aug. 2019.

[4] Y. Erlich and D. Zielinski, “DNA Fountain enables a robust and efficient storage architecture,” Science, vol. 355, no. 6328, pp. 950–954, 2017.

[5] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, “Robust chemical preservation of digital information on DNA in silica with errorcorrecting codes,” Angew. Chem. Int. Ed., vol. 54, no. 8, pp. 2552–2555, 2015.

[6] R. Heckel, G. Mikutis, and R. N. Grass, “A characterization of the DNA data storage channel,” Sci. Rep., vol. 9, no. 1, pp. 1–12, Jul. 2019.

[7] J. Jeong, S.-J. Park, J.-W. Kim, J.-S. No, H. H. Jeon, J. W. Lee, A. No, S. Kim, and H. Park, “Cooperative sequence clustering and decoding for DNA storage system with fountain codes,” Bioinformatics, vol. 37, no. 19, pp. 3136–3143, Oct. 2021.

[8] W. H. Press, J. A. Hawkins, S. K. Jones, J. M. Schaub, and I. J. Finkelstein, “HEDGES error-correcting code for DNA storage corrects indels and allows

sequence constraints," Proc. Nat. Acad. Sci. USA, vol. 117, no. 31, pp. 18489–18496, Aug. 2020.

[9] S. Chandak, H. Ji, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, and T. Weissman, "Improved read/write cost tradeoff in DNA-based data storage using LDPC codes," in Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton), Sep. 2019, pp. 147–156.

[10] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso, B. W. Pruitt, and G. M. Church, "Forward error correction for DNA data storage," Proc. Comput. Sci., vol. 80, pp. 1011–1022, Jan. 2016.

[11] R. Hulett, S. Chandak, and M. Wootters, "On coding for an abstracted nanopore channel for DNA storage," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jul. 2021, pp. 2465–2470.

[12] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, and H. Ji, "Overcoming high nanopore basecaller error rates for DNA storage via basecaller-decoder integration and convolutional codes," in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), May 2020, pp. 8822–8826.

[13] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian, "DNA storage: Research landscape and future prospects," Nat. Sci. Rev., vol. 7, no. 6, pp. 1092–1107, Jun. 2020.

[14] P. L. Antkowiak, J. Lietard, M. Z. Darestani, M. M. Somoza, W. J. Stark, R. Heckel, and R. N. Grass, "Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction," Nature Commun., vol. 11, no. 1, pp. 1–10, Oct. 2020.

[15] K. E. Baddour and N. C. Beaulieu, "Autoregressive modeling for fading channel simulation," IEEE Trans. Wireless Commun., vol. 4, no. 4, pp. 1650–1662, Jul. 2005.

[16] H. Bai and M. Atiquzzaman, "Error modeling schemes for fading channels in wireless communications: A survey," IEEE Commun. Surveys Tuts., vol. 5, no. 2, pp. 2–9, 4th Quart., 2003.

[17] H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," Sci. Rep., vol. 5, pp. 1–10, Sep. 2015.

[18] L. Organick et al., "Random access in large-scale DNA data storage," Nature Biotechnol., vol. 36, pp. 242–248, Mar. 2018.

[19] H. H. Lee, R. Kalhor, N. Goela, J. Bolot, and G. M. Church, "Terminatorfree template-independent enzymatic DNA synthesis for digital information storage," Nature Commun., vol. 10, no. 1, pp. 1–12, Jun. 2019.

[20] L. Anavy, I. Vaknin, O. Atar, R. Amit, and Z. Yakhini, "Data storage in DNA with fewer synthesis cycles using composite DNA letters," Nature Biotechnol., vol. 37, no. 10, pp. 1229–1236, Oct. 2019.

[21] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over sets for DNA storage," IEEE Trans. Inf. Theory, vol. 66, no. 4, pp. 2331–2351, Apr. 2020.

[22] R. Heckel, I. Shomorony, K. Ramchandran, and D. N. C. Tse, "Fundamental limits of DNA storage systems," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2017, pp. 3130–3134.

[23] I. Shomorony and R. Heckel, "Capacity results for the noisy shuffling channel," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jul. 2019, pp. 762–766.

[24] N. Weinberger and N. Merhav, "The DNA storage channel: Capacity and error probability," 2021, arXiv:2109.12549.

[25] E. G. S. Antonio, T. Heinis, L. Carteron, M. Dimopoulou, and M. Antonini, "Nanopore sequencing simulator for DNA data storage," in Proc. Int. Conf. Vis. Commun. Image Process. (VCIP), Dec. 2021, pp. 1–5.

[26] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: A next-generation sequencing read simulator," Bioinformatics, vol. 28, no. 4, pp. 593–594, 2012.

[27] T. Griebel, B. Zacher, P. Ribeca, E. Raineri, V. Lacroix, R. Guigó, and M. Sammeth, "Modelling and simulating generic RNA-seq experiments with the flux simulator," Nucleic Acids Res., vol. 40, no. 20, pp. 10073–10083, Nov. 2012.

[28] X. Hu, J. Yuan, Y. Shi, J. Lu, B. Liu, Z. Li, Y. Chen, D. Mu, H. Zhang, N. Li, Z. Yue, F. Bai, H. Li, and W. Fan, "PIRS: Profile-based illumina pair-end reads simulator," Bioinformatics, vol. 28, no. 11, pp. 1533–1535, Jun. 2012, doi: 10.1093/bioinformatics/bts187.

[29] B. K. Stöcker, J. Köster, and S. Rahmann, "SimLoRD: Simulation of long read data," Bioinformatics, vol. 32, no. 17, pp. 2704–2706, Sep. 2016, doi: 10.1093/bioinformatics/btw286.

[30] Y. Ono, K. Asai, and M. Hamada, "PBSIM2: A simulator for long-read sequencers with a novel generative model of quality scores," Bioinformatics, vol. 37, no. 5, pp. 589–595, May 2021.

[31] Y. Li, R. Han, C. Bi, M. Li, S. Wang, and X. Gao, "DeepSimulator: A deep simulator for Nanopore sequencing," Bioinformatics, vol. 34, no. 17, pp. 2899–2908, 2018.

[32] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "DeepSimulator1.5: A more powerful, quicker and lighter simulator for nanopore sequencing," Bioinformatics, vol. 36, no. 8, pp. 2578–2580, Apr. 2020.

[33] W. Chen, P. Zhang, L. Song, J. Yang, and C. Han, "Simulation of nanopore sequencing signals based on BiGRU," Sensors, vol. 20, no. 24, p. 7244, Dec. 2020.

[34] C. Rohrandt, N. Kraft, P. Gieselmann, B. Brandl, B. M. Schuldt, U. Jetzek, and F.-J. Müller, "Nanopore SimulatION—A raw data simulator for nanopore sequencing," in Proc. IEEE Int. Conf. Bioinf. Biomed. (BIBM), Dec. 2018, pp. 1–8.

[35] M. Schwarz, M. Welzel, T. Kabdullayeva, A. Becker, B. Freisleben, and D. Heider, "MESA: Automated assessment of synthetic DNA fragments

and simulation of DNA synthesis, storage, sequencing and PCR errors," Bioinformatics, vol. 36, no. 11, pp. 3322–3326, Jun. 2020.

[36] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, and J. M. Boutell, "Accurate whole human genome sequencing using reversible terminator chemistry," Nature, vol. 456, no. 7218, pp. 53–59, 2008.

[37] D. Branton et al., "The potential and challenges of nanopore sequencing," Nature Biotechnol., vol. 26, pp. 1146–1153, Oct. 2008.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2672–2680.

[39] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in Proc. 8th Workshop Syntax, Semantics Struct. Stat. Transl., 2014, pp. 103–111.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, pp. 533–536, Oct. 1986.

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Proc. Adv. Neural Inf. Process. Syst., vol. 27, 2014, pp. 1–15.

[43] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2014, pp. 1724–1734. [44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in Proc. 3rd ICLR, San Diego, CA, USA, 2015, pp. 1–22.

[44] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning,” in Proc. 34th ICML, 2017, pp. 1243–1252.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. 2019 Conf. North Amer., 2019, pp. 4171–4186.

[47] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” 2015, arXiv:1511.06434.

[48] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” OpenAI Blog, vol. 1, p. 9, Feb. 2019.

[49] T. Brown et al., “Language models are few-shot learners,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, 2020, pp. 1877–1901.

[50] A. Baevski and M. Auli, “Adaptive input representations for neural language modeling,” in Proc. 7th ICLR, New Orleans, LA, USA, 2019, pp. 1–13.

[51] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu, “On layer normalization in the transformer architecture,” in Proc. 37th PMLR, 2020, pp. 10524–10533.

[52] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” 2017, arXiv:1701.04862.

[53] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in Proc. 34th PMLR, Aug. 2017, pp. 214–223. [Online]. Available: https://proceedings.mlr.press/v70/arjovsky17a.html

[54] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, “Mode regularized generative adversarial networks,” in Proc. 5th ICLR, Toulon, France, 2019, pp. 1–22.

[55] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in Proc. Adv. Neural Inf. Process. Syst., vol. 29, 2016, pp. 1–12.

[56] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: Boundary equilibrium generative adversarial networks," 2017, arXiv:1703.10717.

[57] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5769–5779.

[58] Q. Mao, H.-Y. Lee, H.-Y. Tseng, S. Ma, and M.-H. Yang, "Mode seeking generative adversarial networks for diverse image synthesis," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 1429–1437.

[59] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in Proc. 6th ICLR, Vancouver, BC, Canada, 2018, pp. 1–13.

[60] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired imageto-image translation using cycle-consistent adversarial networks," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 2223–2232.

[61] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 1125–1134.

[62] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 8789–8797.

[63] R. Feng, D. Zhao, and Z.-J. Zha, "Understanding noise injection in GANs," in Proc. 38th ICML, Jul. 2021, pp. 3284–3293.

[64] N. C. Rakotonirina and A. Rasoanaivo, "ESRGAN+: Further improving enhanced super-resolution generative adversarial network," in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), May 2020, pp. 3637–3641.

[65] Y. Alharbi and P. Wonka, "Disentangled image generation through structured noise injection," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 5134–5142.

[66] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: Learning a generative model from a single natural image," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 4570–4580.

[67] E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky, "Few-shot adversarial learning of realistic neural talking head models," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9459–9468.

[68] S. Jenni and P. Favaro, "On stabilizing generative adversarial training with noise," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 12145–12153.

[69] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional GANs," 2017, arXiv:1706.02633.

[70] J. Yoon, D. Jarrett, and M. Van der Schaar, "Time-series generative adversarial networks," in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 1–15.

[71] H. Ni, L. Szpruch, M. Sabate-Vidales, B. Xiao, M. Wiese, and S. Liao, "Sig-Wasserstein GANs for time series generation," in Proc. 2nd ACM Int. Conf. AI Finance. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–8, doi: 10.1145/3490354.3494393.

[72] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two pure transformers can make one strong GAN, and that can scale up," in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 1–14.

[73] R. Durall, S. Frolov, J. Hees, F. Raue, F.-J. Pfreundt, A. Dengel, and J. Keuper, "Combining transformer generators with convolutional discriminators," in Proc. KI, 2021, pp. 67–79.

[74] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence generative adversarial nets with policy gradient," in Proc. AAAI, vol. 31, Feb. 2017, pp. 1–7.

[75] O. Mogren, "C-RNN-GAN: A continuous recurrent neural network with adversarial training," in Proc. CML Workshop NIPS, 2016, p. 1.

[76] K.-H. Zeng, M. Shoeybi, and M.-Y. Liu, "Style example-guided text generation using generative adversarial transformers," 2020, arXiv:2003.00674.

[77] M. Saha, X. Guo, and A. Sharma, "TilGAN: GAN for facilitating tumorinfiltrating lymphocyte pathology image synthesis with improved image classification," IEEE Access, vol. 9, pp. 79829–79840, 2021.

[78] X. Li, V. Metsis, H. Wang, and A. H. H. Ngu, "TTS-GAN: A transformerbased time-series generative adversarial network," in Proc. 20th Int. Conf. Artif. Intell. Med. Berlin, Germany: Springer-Verlag, 2022

[79] A. Muhamed, L. Li, X. Shi, S. Yaddanapudi, W. Chi, D. Jackson, R. Suresh, Z. C. Lipton, and A. J. Smola, "Symbolic music generation with transformer-GANs," in Proc. AAAI Conf. Artif. Intell., May 2021, vol. 35, no. 1, pp. 408–417. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16117

[80] B. Peng, X. Huang, S. Wang, and J. Jiang, "A real-time medical ultrasound simulator based on a generative adversarial network model," in Proc. IEEE Int. Conf. Image Process. (ICIP), Sep. 2019, pp. 4629–4633.

[81] M. Rahnemoonfar, M. Yari, and J. Paden, "Radar sensor simulation with generative adversarial network," in Proc. IEEE Int. Geosci. Remote Sens. Symp., Sep. 2020, pp. 7001–7004.

[82] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," Comput. Graph. Forum, vol. 38, no. 2, pp. 59–70, May 2019.

[83] M. Erdmann, J. Glombitza, and T. Quast, “Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network,” Comput. Softw. Big Sci., vol. 3, no. 1, pp. 1–13, Dec. 2019.

[84] Y. Yang, Y. Li, W. Zhang, F. Qin, P. Zhu, and C.-X. Wang, “Generativeadversarial-network-based wireless channel modeling: Challenges and opportunities,” IEEE Commun. Mag., vol. 57, no. 3, pp. 22–27, Mar. 2019.

[85] T. Orekondy, A. Behboodi, and J. B. Soriaga, “MIMO-GAN: Generative MIMO channel modeling,” 2022, arXiv:2203.08588.

[86] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[87] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, “1D convolutional neural networks and applications: A survey,” Mech. Syst. Signal Process., vol. 151, Apr. 2021, Art. no. 107398.

[88] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, “Generalization and equilibrium in generative adversarial nets (GANs),” in Proc. 34th PMLR, 2017, pp. 224–232.