IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 November 2022, accepted 31 December 2022, date of publication 9 January 2023, date of current version 12 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3235201

RESEARCH ARTICLE

Generative Adversarial Networks for DNA Storage Channel Simulator

SANGHOON KANG[1], YUNFEI GAO[1], JAEHO JEONG[2], (Graduate Student Member, IEEE), SEONG-JOON PARK[2], (Graduate Student Member, IEEE), JAE-WON KIM[3], (Member, IEEE), JONG-SEON NO[2], (Fellow, IEEE), HAHYEON JEON[4], JEONG WOOK LEE[5], SUNGHWAN KIM[6], (Member, IEEE), HOSUNG PARK[7,8], (Member, IEEE), AND ALBERT NO[1], (Member, IEEE)

[1]Department of Electronic and Electrical Engineering, Hongik University, Seoul 04066, South Korea
[2]Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[3]Department of Electronic Engineering, Engineering Research Institute, Gyeongsang National University, Jinju 52828, South Korea
[4]Clinomics, Ulsan 44919, South Korea
[5]Department of Chemical Engineering, POSTECH, Pohang 37673, South Korea
[6]School of Electrical Engineering, University of Ulsan, Ulsan 44610, South Korea
[7]Department of Computer Engineering, Chonnam National University, Gwangju 61186, South Korea
[8]Department of ICT Convergence System Engineering, Chonnam National University, Gwangju 61186, South Korea

Corresponding author: Albert No (albertno@hongik.ac.kr)

This work was supported by the Samsung Research Funding and Incubation Center for Future Technology under Grant SRFC-IT1802-09.

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. The code is available at https://github.com/gyfbianhuanyun/DNA_storage_simulator_GAN.

INDEX TERMS Channel simulator, DNA storage, generative adversarial networks, recurrent neural networks, transformer.

I. INTRODUCTION

DNA storage is one of the most promising next-generation storage systems [1], [2]. Contrary to the modern digital system, which stores data in binary format using 0 and 1, the DNA storage system converts data in the quaternary format using four bases (Adenine(A), Cytosine(C), Guanine(G), and Thymine(T)). Then, the system writes data by synthesizing corresponding DNA strands and reads data using the DNA sequencing process [3]. The data density of DNA storage can reach 215 petabytes per gram [4]. At the same time, maintenance costs are low where fragments of DNA encapsulated in silica can be preserved for thousands of years [5]. However, the synthesis and sequencing technologies are error-prone [6], and therefore an appropriate error-correcting code (ECC) is necessary for reliable data recovery [4], [7], [8], [9].

The errors in the DNA storage channel are often asynchronous (insertion and deletion). Moreover, the sequence

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

contains higher-order statistics; For example, the error rate increases in the region of consecutive bases. Thus, it is notoriously hard to characterize the channel statistics perfectly, so designing an ECC is also challenging. A line of research focuses on ECC for a such noisy channel, which we call the DNA storage channel; Blawat et al. [10] proposed a forward error correction scheme, Erlich and Zielinski [4] combined Reed-Solomon (RS) code and fountain code, where Jeong et al. [7] also applied RS codes with an improved decoding technique. Also, Press et al. [8] corrected asynchronous errors using hashing and greedy search, and Chandak et al. [9] used low-density parity-check (LDPC) and BCH codes. In addition, Hulett et al. [11] proposed a specific coding scheme for Nanopore sequencing, where Chandak et al. [12] used convolutional codes to correct many insertions and deletions.

Although the cost of sequencing is rapidly decreasing [13], the high price of synthesis [14] is still the main bottleneck for developing ECCs for DNA storage channels. It is well known that simulation has predictive functions before conducting actual experiments. It is used to predict the performance of new ECC schemes in wireless communications [15], [16]. This approach significantly reduces the cost burden of designing and testing ECC. Likewise, we hope to achieve a similar effect using simulation in the DNA storage channel.

In addition, the simulator is also valuable for the channel-related random access verification [17], [18], analysis of new synthesis techniques [19], [20], and bound analysis of the channel [21], [22], [23], [24]. However, to the best of our knowledge, there is no publicly available DNA storage channel simulator except Antonio et al. [25], which mostly focused on the Nanopore sequencer.

There are various types of simulators related to DNA sequencing at present. ART [26] mimics a next-generation sequencer, whereas Flux [27] simulates RNA-seq. Also, pRIS [28] and simLoRD [29] simulate Illumina and PacBiuo reads, respectively, when the error profiles are given. PBSIM2 [30] simulates long reads of PacBio sequencer with the quality value generated by the trained Hidden Markov Model (HMM). However, the above sequencing simulators focus only on 1st order statistics (e.g., error probabilities at each location). Deep Simulator [31], [32] uses deep learning techniques to mimic the entire pipeline of Nanopore sequencing. NanosigSim [33] improves the signal generator module of the Deep Simulator using a bidirectional gated recurrent unit (GRU). Nanopore SimulatION [34] is a modular software tool capable of simulating the raw current values of Nanopore sequencing reads. Schwarz et al. [35] simulated error sequences with pre-defined error probability, which needs to be specified by the user.

However, most of the simulators have hard-coded error probabilities or have been designed for a specific sequencing technology. Since it is a rapidly developing research area, there are many different technologies in both sequencing (e.g., Illumina [36] and Oxford Nanopore [37]) and synthesis (e.g., Twist Bioscience). Each technology has unique characteristics; for example, Nanopore technologies can sequence long read (>1000 base pairs) while Illumina sequencing can handle up to a few hundred base pairs. Simulators based on specific experimental setups may not apply to an environment with new sequencing and synthesis technologies. For example, Deep Simulator [31] has pore modeling that requires parameter engineering and may not apply to Illumina sequencing technologies.

In this paper, we propose a deep learning-based DNA storage channel simulator. Our model is based on a generative adversarial network (GAN) [38] which captures high-order statistics of data. Also, the model's training is universal because it does not depend on specific sequencing and synthesis technologies. More precisely, we employ an end-to-end optimized deep learning-based algorithm, not requiring parameter engineering but only depending on data statistics.

Note that designing a GAN-based simulator is non-trivial. Most GAN is optimized to match the output distribution, but the simulator needs to match the joint distribution of input and output. Also, since the goal of the DNA storage simulator is to mimic a noisy channel, we expect the randomness of the generated sequence. In other words, the same input sequence *must* produce various outputs where the distribution should match the actual experimental data. This is not the case in most GAN problems where the randomness is obtained from a random latent input vector, and the generator (simulator) is deterministic. This work proposes a novel GAN structure where the discriminator takes both input and output to verify the input-output statistics to overcome the problems.

We train the proposed model on 10^5 input sequences and 10^9 output sequences were obtained from a single experiment, using a similar procedure described by Jeong et al. [7]. Note that this is enough to train the proposed model, which implies that an extensive data acquisition experiment is unnecessary. The proposed framework allows us to build a simulator with the same statistics based on data from a single experiment.

To verify the effectiveness of the proposed simulator, we measure the error statistics. Our simulator shows similar insertion, substitution, and deletion error rates. It also matches the higher-order statistics, such as the probability of consecutive deletions (2, 3, 4-deletions). Moreover, we achieve the output randomness using multiple weights obtained from the training procedure. Finally, we apply the ECC technique proposed by Jeong et al. [7] to simulated reads. We observe a similar error-correcting performance compared to that of the real experimental data.

II. BACKGROUNDS

A. 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-of-the-art performances. The pre–LN Transformer [51], which places the layer normalization inside the residual block, improves training with well-behaving gradients [52].

B. GENERATIVE ADVERSARIAL NETWORK

A Generative Adversarial Network (GAN) [38] is a framework that adversarially trains two networks, a generator and a discriminator. The role of the generator is to produce fake (in our case, simulated) data that are similar to the given data distribution, while the discriminator tries to distinguish between the real and generated (fake) data. Equivalently, the generator minimizes the minimax loss, while the discriminator maximizes it [53]. With the competing nature of the discriminator and the generator, the generator keeps producing more realistic generated (fake) data while the discriminator detects the generated data more accurately. Finally, the generated data has similar statistics to the actual data and cannot be distinguished by the discriminator.

Wasserstein GAN (WGAN) [54] minimizes the Wasserstein distance between the empirical distribution of actual data and generated data. WGAN is more resistant to the mode collapse problem in which the generator generates only specific data caused by unbalanced training between the generator and discriminator [55], [56]. Berthelot et al. [57] improves WGAN by boundary equilibrium. Gulrajani et al. [58] introduced a gradient penalty that shows better stability in training. Additionally, Mao et al. [59] map the generated data to a latent vector, where Miyato et al. [60] constrained the Lipschitz condition using weight normalization to overcome mode collapse.

Although many generative models focus on mimicking data distribution, there are GANs that considered the input and output correlation [61], [62], [63]. However, the above work proposed deterministic generators. To achieve randomness of the output, many works [64], [65], [66], [67], [68], [69] inject noise into the generator. However, their primary concern is the variability of the output instead of targeting a specific joint distribution of the input and the output.

Most GAN generators (and discriminators) are convolutional neural networks since GAN is originally designed for vision tasks [38]. Recently, some efforts are tried to create RNN-based GAN [70], [71], [72]. Further, Transformer-based GANs [73], [74] were proposed and showed the robustness of training. On the other hand, there are a few works that considered generating sequences using GAN, including the RNN with reinforcement training [75], [76], and the Transformer-based generator for text generation [77], [78], even long sequence generation [79], [80].

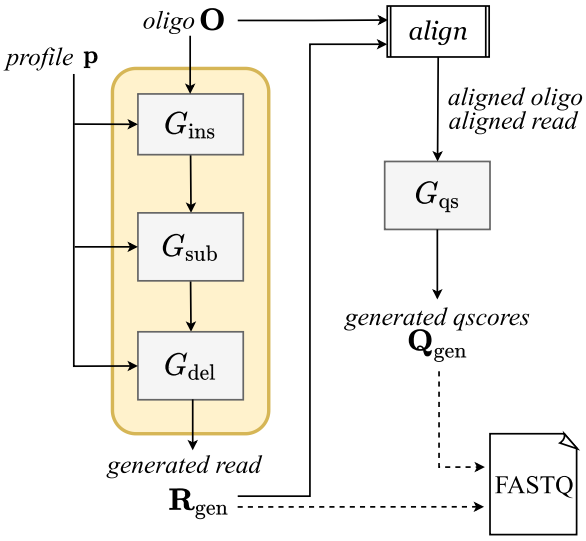


FIGURE 1. Overall structure of the simulator. The read generator, consists of insertion, substitution, and deletion generators, takes an oligo sequence as an input and generates a read sequence. The generated output is aligned with an input oligo sequence, and the quality score generator produces corresponding quality scores.

Since GAN is to mimic the given data distribution, many studies have proposed GAN-based simulators. Peng et al. [81] constructed a GAN medical ultrasound simulator suitable for medical simulation and clinical training. Also, Rahnemoonfar et al. [82] used a similar architecture to CycleGAN [61] to synthesize snow radar images, Kim et al. [83] shows user-parameterized GAN-based simulator of fluid, and Erdmann et al. [84] simulated an electromagnetic calorimeter showers using WGAN. Notably, Yang et al. [85] suggested a GAN-based wireless channel modeling for the continuous additive white Gaussian noise channel, and Orekondy et al. [86] proposed a GAN-based simulator that learns the multi-channel distributions of the multiple-input multiple-output channel.

III. METHODS

A. 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 $\mathcal{X} = \{A, C, G, T\}$ be a set of nucleotide bases, and $\mathcal{X}_e = \{A, C, G, T, -\}$ be a set of extended symbols where '–' is a placeholder that corresponds to inserted or deleted symbols. Further, let $\mathcal{Q} \subset [0, 1]$ 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*:

$$O = O_1 O_2 \cdots O_n$$

where $O_i \in \mathcal{X}$ for $1 \le i \le n$. Then, the simulator's output consists of length M of DNA sequence R, which we call the *read*, and corresponding *quality scores* Q:

$$R = R_1 R_2 \cdots R_M Q = Q_1 Q_2 \cdots Q_M$$

where $R_j \in \mathcal{X}$ and $Q_j \in \mathcal{Q}$ for $1 \le j \le M$. 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 $Y_j = (R_j, Q_j) \in \mathcal{X} \times \mathcal{Q}$ for $1 \le j \le M$.

B. SIMULATOR

We propose a read generator G_{read} followed by a quality score (qscore) generator G_{qs}. Given an input oligo O, the read generator $G_{\text{read}}(O)$ outputs a read R, and $G_{\text{qs}}(O, R)$ 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 $X_{\text{ins}}, X_{\text{sub}}$, and X_{del}, i.e., $O \to X_{\text{ins}} \to X_{\text{sub}} \to X_{\text{del}} \to R$ (we will specify later why X_{del} 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 $X_{\text{ins},k}, X_{\text{sub},k}, X_{\text{del},k} \in \mathcal{X}_e$ for $1 \le k \le L$. Due to the insertion, we have $n \le L$, and similarly we have $M \le L$ because of deletion errors. Then, the read generator is composed of three generators G_{ins}, G_{sub}, and G_{del}, which introduce insertion, substitution, and deletion errors. Note that the sequence O and X_{ins} may have different lengths due to insertion errors. However, the sequence lengths of X_{ins}, X_{sub}, and X_{del} are always the same since it has a placeholder symbol '–'. Finally, a trimmer removes '–' symbol from X_{del}, 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 $p = (p_i, p_s, p_d) \in \{0, 1\}^3$ that indicates whether each type of error occurs or not. More precisely,

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

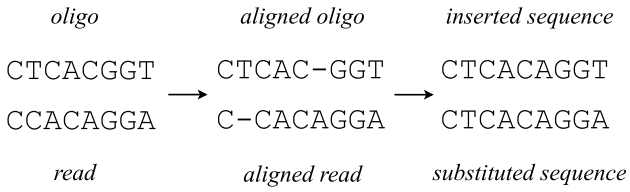
$$X_{\text{sub}} = \begin{cases} G_{\text{sub}}(X_{\text{ins}}) & \text{if } p_s = 1 \\ X_{\text{ins}} & \text{if } p_s = 0 \end{cases}$$

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

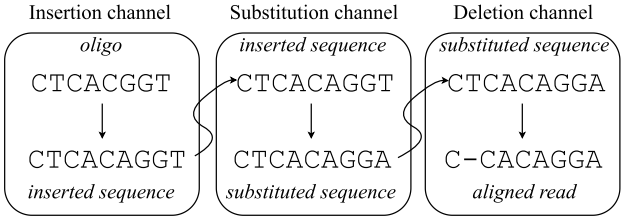
While training, we estimate the probability distribution of the profile vector based on occurrences. In the simulation step, for each oligo input, the simulator samples a random profile vector based on the empirical distribution of the profile vector and feeds it to all generators (G_{ins}, G_{sub}, and G_{del}).

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 in Section IV.

The qscore simulator G_{qs} takes an oligo and read pair (O, R) as an input. It first aligns the oligo and the read which we call O_{aligned} and R_{aligned}, respectively. Then, it produces quality scores corresponding to an aligned read. Similar to the read generation, since the error rarely occurs in G_{read} and the qscore statistics are significantly different when there exists an error, we introduce two generators $G_{\text{qs}}^{(g)}$ and $G_{\text{qs}}^{(e)}$. If there is no error between generated read and the input oligo, we apply an error-free generator $G_{\text{qs}}^{(g)}$, and we apply $G_{\text{qs}}^{(e)}$ if there is an



(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 2. Dataset generation for insertion, substitution, and deletion channels.

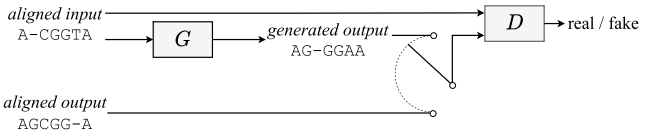


FIGURE 3. Structure of GANs for the read simulator. This includes an insertion GAN, a substitution GAN, and a deletion GAN.

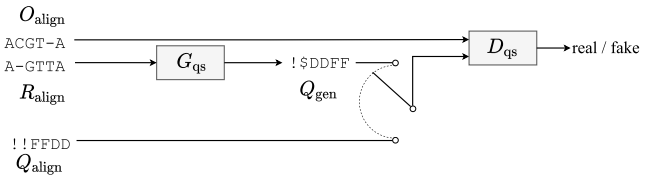


FIGURE 4. Structure of quality score GAN for the quality score simulator.

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 1.

IV. TRAINING GENERATORS USING GAN

A. GANs FOR READ GENERATOR

We train G_{ins}, G_{sub}, and G_{del} using three separate GANs (insGAN, subGAN, and delGAN for G_{ins}, G_{sub}, and G_{del}, respectively). However, we need to define each generator's input and output since the intermediate sequences (X_{ins}, X_{sub}, and X_{del}) are not provided. The insGAN requires O and X_{ins}, the subGAN needs X_{ins} and X_{sub}, and finally, the delGAN is trained on X_{sub} and X_{del}.

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

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 (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 3.

1) insGAN

The critical component of insertion generator G_{ins} 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 D_{ins} as the discriminator corresponding to the insertion generator G_{ins}. It takes the generator's input and output (O, X_{ins}) as input to check whether the input and output show proper channel statistics. The discriminator D_{ins} is based on a 1-dimensional convolutional neural network (1D-CNN) [88], which investigates the joint distribution of input O and output X_{ins}. 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, 1D-CNN 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 G_{ins} 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, X_{ins}) 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.

2) subGAN

The substitution generator G_{sub} 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 G_{sub} takes X_{ins} as an input and outputs X_{sub}. 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

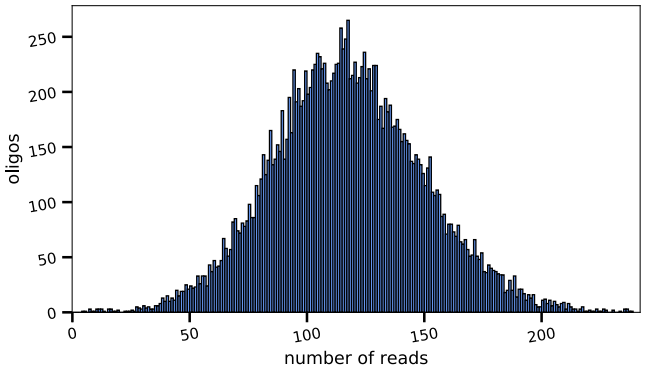


FIGURE 5. Distribution of matched reads per oligos. The most oligos have 100–150 matching reads.

processed by a fully connected layer with a soft plus activation function. The discriminator D_{sub} adopts a similar construction to insGAN, ten layers 1D-CNN with 128 hidden dimensions, 5×5 kernel, and a single zero padding.

3) delGAN

The deletion generator G_{del} and discriminator D_{del} are identical to that of insGAN. The generator G_{del} has three layers of bidirectional GRU and 64 hidden dimensions, where D_{del} has ten layers of 1D-CNN with 8 hidden dimensions, 5×5 kernel, and a single zero padding. However, since the input X_{sub} and X_{del} 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.

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 in Section V-C.

B. GAN FOR QUALITY SCORE GENERATOR

The quality score generator takes aligned oligo and read pair $(O_{\text{aligned}}, R_{\text{aligned}})$ as an input. We train two quality score generators $(G_{\text{qs}}^{(g)}, G_{\text{qs}}^{(e)})$ 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 O_{aligned}, aligned read R_{aligned}, and aligned quality scores Q_{aligned}. 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).

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

The quality score discriminator takes triplet of aligned oligo, aligned read, and aligned quality scores $(O_{\text{aligned}}, R_{\text{aligned}}, Q_{\text{aligned}})$ 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 4.

1) QSCORE GAN

The quality score generator G_{qs} has a similar structure with G_{ins}, 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 D_{qs} 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.

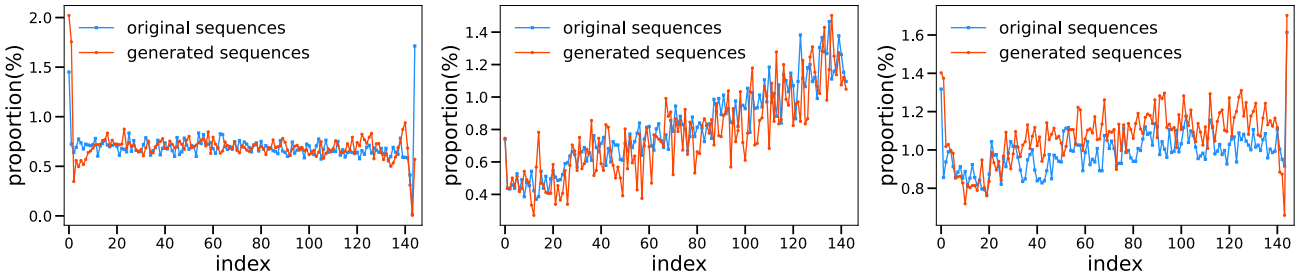
Similar to generators in G_{read}, the generator takes additional random numbers as input corresponding to the base. It introduces the randomness of generated quality scores. While training, we added L1-regularization for quality score GAN for stability.

V. EXPERIMENT

A. DATASET

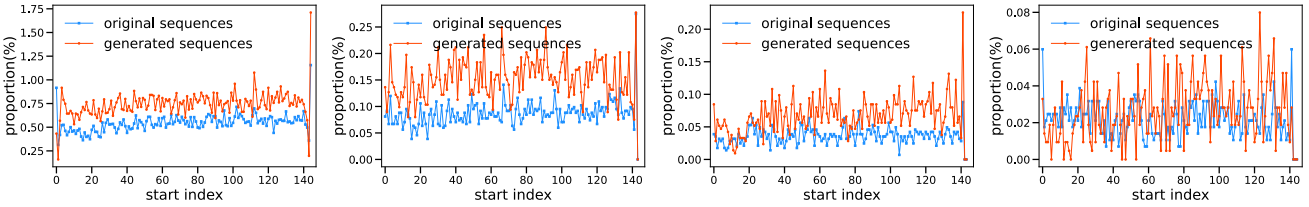
In the experiment, we use the same oligo design proposed by Jeong et al. [7]. There are 18,000 oligos of length 152 in the dataset, where all oligos have balanced GC contents (base G and C proportion is between 45 to 55 percent) and homopolymer run length is limited by 3. We obtain the corresponding 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 5). 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



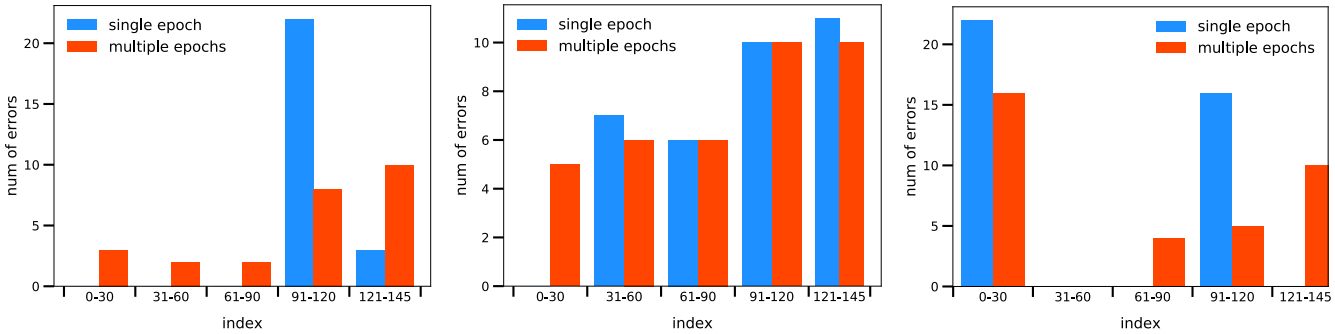
(a) insertion (b) substitution (c) deletion

FIGURE 6. Error rates of insertion, substitution, and deletion channels at each index.



(a) 1-deletion (b) 2-deletion (c) 3-deletion (d) 4-deletion

FIGURE 7. Error rates of consecutive deletion errors at each index. Our simulator shows similar rates, which implies it learns high order dependencies.



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

FIGURE 8. Comparison of error distribution (index of erroneous base) between generated reads using single weight and using multiple weights.

delGAN as described in Figure 2. 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 $G_{\text{qs}}^{(g)}$ and the error-contained dataset for $G_{\text{qs}}^{(e)}$. 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 in Section IV-A4. 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.

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

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

B. TRAINING

We train WGANs (insGAN, subGAN, delGAN, and qscoreGAN) with gradient penalty. The batch size is 1024, and learning rates of the generator and discriminator are 5×10^{-4} for insGAN, 5×10^{-4} for G_{sub}, 1×10^{-4} for D_{sub}, 10^{-3} for

TABLE 2. Comparison of error types (base pair) 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 3. Comparison of error types (base pair) of substitution channels.

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

delGAN, and 10^{-4} for qscoreGAN. 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.

C. 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.

1) READ GENERATORS: ERROR STATISTICS

To evaluate the performance of the proposed simulator, we first investigate the error statistics of reads. Figure 6-(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 6-(b) and Figure 6-(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).

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 shows the rate of consecutive deletion errors (1-deletion to 4-deletions) starting at each index. Table 1 compares the ratio of 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.

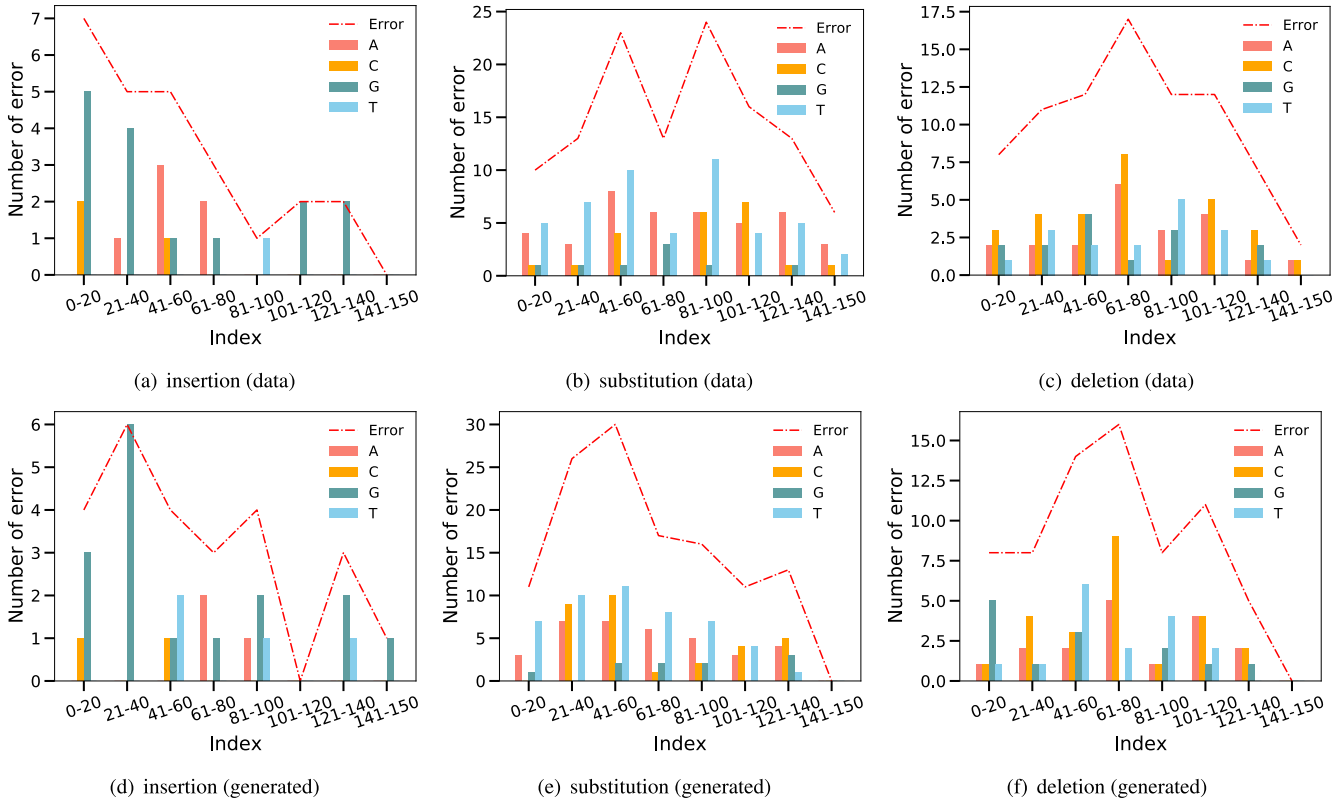
TABLE 4. The number of erroneous bases of 25 generated reads from a single fixed input oligo sequence. Due to unstable nature of GAN training, the error distribution of generated reads using different weights significantly varies.

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

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 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.

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 G_{ins}, we use 11 generators with trained weights at epochs from 2400 to 2410, where we use 11 generators for the deletion generator G_{del} with trained weights at epochs from 1290 to 1300. In the substitution generator G_{sub}, 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 8 show the



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

(d) insertion (generated) (e) substitution (generated) (f) deletion (generated)

FIGURE 9. Randomness of generated reads from a single fixed oligo at insertion, substitution, and deletion channels. The *x*-axis presents the index of the error, while different color indicates the different erroneous bases. The dotted line shows the number of errors of all bases at each index.

analysis of the simulated reads by the insertion generator G_{ins}, the substitution generator G_{sub}, and the deletion generator G_{del}. 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. The comparison is between generated sequences using single weight configuration and sequences using 10 different weights from 10 epochs. Figure 10 implies that the generation with multiple weights has more variability in error patterns. Moreover, Figure 9 shows the error statistics of generated sequences from the fixed oligo, where we generated 25 sequences to test G_{ins}, 100 sequences to test G_{sub}, and 50 sequences to test G_{del}. 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.

3) QSCORE GENERATOR

Figure 10 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 11 shows

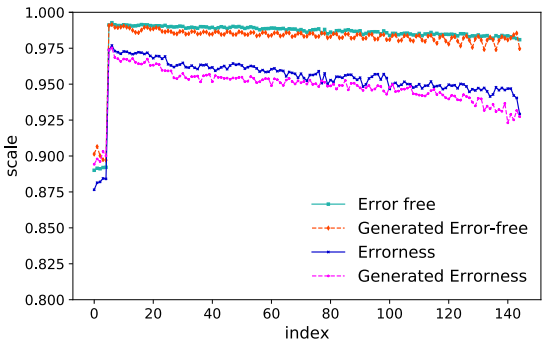
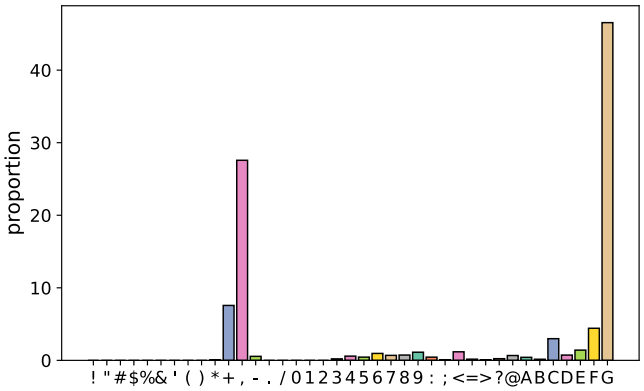
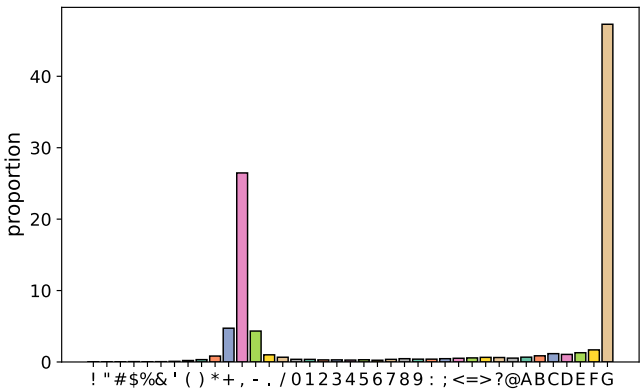


FIGURE 10. Mean of quality scores at each index. We present the mean quality scores of error-free reads, and the mean quality scores of reads with errors.

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 WGAN-GP, 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 11. Distribution of corresponding quality scores when the substitution error occurs.

TABLE 5. Clustering and decoding statistics at the smallest random samples for both simulation (ours) and experiment results [7].

	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%

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-score-based 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 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.

VI. CONCLUSION AND DISCUSSION

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.

ACKNOWLEDGMENT

(Sanghoon Kang and Yunfei Gao contributed equally to this work.)

REFERENCES

[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 error-correcting 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, "Terminator-free 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.

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.

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

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

[52] 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.

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

[54] 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

[55] 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.

[56] 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.

[57] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: Boundary equilibrium generative adversarial networks," 2017, *arXiv:1703.10717*.
[58] 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.
[59] 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.
[60] 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.
[61] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2223–2232.
[62] 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.
[63] 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.
[64] R. Feng, D. Zhao, and Z.-J. Zha, "Understanding noise injection in GANs," in *Proc. 38th ICML*, Jul. 2021, pp. 3284–3293.
[65] 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.
[66] 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.
[67] 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.
[68] 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.
[69] 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.
[70] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional GANs," 2017, *arXiv:1706.02633*.
[71] 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.
[72] 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.
[73] 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.
[74] 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.
[75] 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.
[76] O. Mogren, "C-RNN-GAN: A continuous recurrent neural network with adversarial training," in *Proc. CML Workshop NIPS*, 2016, p. 1.
[77] K.-H. Zeng, M. Shoeybi, and M.-Y. Liu, "Style example-guided text generation using generative adversarial transformers," 2020, *arXiv:2003.00674*.
[78] M. Saha, X. Guo, and A. Sharma, "*TilGAN*: GAN for facilitating tumor-infiltrating lymphocyte pathology image synthesis with improved image classification," *IEEE Access*, vol. 9, pp. 79829–79840, 2021.
[79] X. Li, V. Metsis, H. Wang, and A. H. H. Ngu, "TTS-GAN: A transformer-based time-series generative adversarial network," in *Proc. 20th Int. Conf. Artif. Intell. Med.* Berlin, Germany: Springer-Verlag, 2022, pp. 133–143, doi: 10.1007/978-3-031-09342-5_13.
[80] 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
[81] 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.
[82] 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.
[83] 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.
[84] 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.
[85] Y. Yang, Y. Li, W. Zhang, F. Qin, P. Zhu, and C.-X. Wang, "Generative-adversarial-network-based wireless channel modeling: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 57, no. 3, pp. 22–27, Mar. 2019.
[86] T. Orekondy, A. Behboodi, and J. B. Soriaga, "MIMO-GAN: Generative MIMO channel modeling," 2022, *arXiv:2203.08588*.
[87] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[88] 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.
[89] 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.

SANGHOON KANG received the B.S. and M.S. degrees in electronic and electrical engineering from Hongik University, Seoul, South Korea, in 2014 and 2018, respectively, where he is currently pursuing the Ph.D. degree in electronic and electrical engineering. His research interests include deep learning, machine learning, and computational biology.

YUNFEI GAO is currently pursuing the Ph.D. degree in electronic and electrical engineering with Hongik University, Seoul, South Korea. His research interests include deep learning, DNA storage, and computational biology.

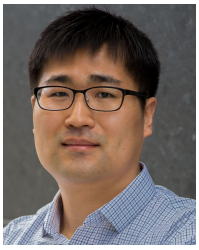
JAEHO JEONG (Graduate Student Member, IEEE) received the B.S. degree in computer science from Yonsei University, Seoul, South Korea, in 2018. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul. His current research interests include error-correcting codes and DNA storage systems.

SEONG-JOON PARK (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from Sogang University, Seoul, South Korea, in 2019. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul. His current research interests include error-correcting codes and DNA storage.

JAE-WON KIM (Member, IEEE) received the B.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2014 and 2020, respectively. From 2020 to 2021, he was a Staff Engineer with Samsung Electronics, South Korea. In 2021, he joined Gyeongsang National University, Jinju, South Korea, where he is currently an Assistant Professor at the Department of Electronic Engineering. His research interests include error-correcting codes, coding theory, index coding, and DNA storage.

JONG-SEON NO (Fellow, IEEE) received the B.S. and M.S.E.E. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1981 and 1984, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1988. He was a Senior MTS with Hughes Network Systems, from 1988 to 1990. He was an Associate Professor with the Department of Electronic Engineering, Konkuk University, Seoul, from 1990 to 1999. In 1999, he joined the Faculty of the Department of Electrical and Computer Engineering, Seoul National University, where he is currently a Professor. His current research interests include error-correcting codes, cryptography, sequences, LDPC codes, interference alignment, and wireless communication systems. He became a fellow of the IEEE through the IEEE Information Theory Society, in 2012. He became a member of the National Academy of Engineering of Korea (NAEK), in 2015, where he served as the Division Chair of electrical, electronic, and information engineering from 2019 to 2020. He was a recipient of the IEEE Information Theory Society Chapter of the Year Award, in 2007. From 1996 to 2008, he served as the Founding Chair for the Seoul Chapter of the IEEE Information Theory Society. He was the General Chair of Sequence and Their Applications 2004 (SETA2004), Seoul. He also served as the General Co-Chair for the International Symposium on Information Theory and its Applications 2006 (ISITA2006) and the International Symposium on Information Theory 2009 (ISIT2009), Seoul. He served as the Co-Editor-in-Chief for the *Journal of Communications and Networks*, from 2012 to 2013.

HAHYEON JEON received the B.S. and M.S. degrees in chemical engineering from the Pohang University of Science and Technology (POSTECH), South Korea, in 2018 and 2020, respectively. He is currently a Researcher at Clinomics, South Korea. His research interests include synthetic biology and bioinformatics.



JEONG WOOK LEE received the B.S. and Ph.D. degrees in chemical and biomolecular engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2004 and 2009, respectively. He was a Postdoctoral Researcher at KAIST, from 2009 to 2011; Boston University, from 2012 to 2014; MIT, from 2014 to 2015; and Harvard University, from 2015 to 2016. He was a Research Scientist at Harvard University, from 2015 to 2016. He has been an Associate Professor at the Department of Chemical Engineering, Pohang University of Science and Technology, Pohang, South Korea, since 2021. His research interests include genetic circuit engineering and its applications in cellular and cell-free environments.

SUNGHWAN KIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Seoul National University, South Korea, in 1999, 2001, and 2005, respectively. He was a Postdoctoral Visitor at the Georgia Institute of Technology (GeorgiaTech), from 2005 to 2007. He was a Senior Engineer at Samsung Electronics, from 2007 to 2011. He is currently a Professor at the Department of Electrical, Electronic and Computer Engineering, University of Ulsan, South Korea. His main research interests include channel coding, modulation, massive MIMO, visible light communication, quantum information, and DNA storage systems.

HOSUNG PARK (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Seoul National University, Seoul, South Korea, in 2007, 2009, and 2013, respectively. He was a Postdoctoral Researcher with the Institute of New Media and Communications, Seoul National University, in 2013, and the Qualcomm Institute, California Institute for Telecommunications and Information Technology, University of California at San Diego, La Jolla, CA, USA, from 2013 to 2015. Since 2015, he has been an Associate Professor with the Department of Computer Engineering, Chonnam National University, Gwangju, South Korea. His research interests include channel codes for communications systems, coding for memory/storage, coding for distributed storage, communication signal processing, compressed sensing, and network information theory.

ALBERT NO (Member, IEEE) received the B.S. degree in electrical engineering and mathematics from Seoul National University, Seoul, South Korea, in 2009, and the M.Sc. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2012 and 2015, respectively. From 2015 to 2017, he was a Data Scientist with Roche. In 2017, he joined Hongik University, Seoul, where he is currently an Assistant Professor of electronic and electrical engineering. His research interests include the learning theory, lossy compression, and bioinformatics.