DNA Storage Toolkit: A Modular End-to-End DNA Data Storage Codec and Simulator

Puru Sharma*, Gary Goh Yipeng, Bin Gao, Longshen Ou, Dehui Lin, Deepak Sharma, Djordje Jevdjic
Department of Computer Science, National University of Singapore
Email: {puru*, e0148664, bingao, oulongshen, e0203126}@u.nus.edu, dr.dpk@nus.edu.sg, djolent@gmail.com
*Corresponding author

***Abstract*—With the amount of data being generated every year increasing exponentially, figuring out where and how to store it efficiently and inexpensively is becoming a larger problem every day. The rapid improvement in performance and cost of DNA synthesis and sequencing methods has led to an increased interest in the use of DNA as a durable and compact medium for data storage. Today, we have a large spectrum of available chemical tools that enable efficient data access and manipulation of in-DNA data. While several DNA storage architectures have been proposed, there is no open-source codec or simulator that implements all of the required components of the DNA-based data storage pipeline for research and development.**

We present an open-source end-to-end DNA data storage toolkit that can take an input file through the entire DNA storage pipeline. Our work contains implementations of the state-of-the-art techniques for each step of the pipeline, including our own algorithms for each step. These steps include encoding data into DNA strands, simulating the wetlab processes of synthesis, storage and sequencing of those DNA strands, clustering of the sequenced results, reconstruction of DNA strands from noisy clusters, and decoding the initially encoded file with support for error-correction mechanisms. Each module can be used individually or combined to form an entire pipeline. We hope that our toolkit will be useful to researchers and developers who seek to experiment with the new and promising storage technology.

***Index Terms*—DNA Storage, Data Storage, Noise Simulation, Clustering, Trace Reconstruction**

I. Introduction

The exponential increase in the amount of data being generated every day has spurred significant interest in storage technologies beyond the ones that are commonplace today. The rapid improvements in DNA synthesis and sequencing technology have led to DNA being proposed as a sustainable storage medium [6], due to its outstanding density and durability. Many DNA storage architectures have been proposed, which leverage a large spectrum of available chemical tools to enable efficient and long-lasting data storage [5], [10], [12], [25], [39], [40].

Unfortunately, there are a lot of circumstances that make it difficult to do research in DNA-based data storage systems. First and foremost, there exists no publicly available end-to-end DNA storage pipeline that can serve as a baseline for research. While in recent years, we have seen a lot of research trying to tackle different parts of the DNA storage pipeline, these works tend to be standalone implementations of small sections of the pipeline, where the impact of any change on prior and later modules in the pipeline are not studied [18], [30], [31]. The strong inter-module dependencies and the lack of the end-to-end framework makes it difficult to judge the actual performance of any proposed improvements, as the overall impact of the change on the whole DNA storage system cannot be measured.

Secondly, the steps in the DNA storage pipeline that are performed in the wetlab, i.e., the synthesis, storage and sequencing of DNA molecules, can be very expensive, both in terms of cost and time, and require specialized equipment and expertise. Only a handful of companies can reliably synthesize DNA strands, and their services are very expensive, currently costing at least 1000 dollars per MB of information encoded in DNA [6], and oftentimes orders of magnitude more. Beyond synthesis, most computing labs do not have access to sequencing machines, or the required tools, chemicals, and expertise needed to manipulate DNA and prepare it for sequencing, and hence need to send their DNA strands to other commercial or research wetlabs, adding days, if not weeks to their experiments. To perform experiments with DNA storage systems in a timely and resource-efficient manner, we need to be able to accurately simulate various wetlab steps in the pipeline. While a few different simulation approaches have been proposed, they assume fairly simplistic error models that do not capture the complex nature of errors that can occur when storing data in DNA [18] and retrieving it back, resulting in unrealistic performance and behavior of individual pipeline modules, and the unrealistic overall end-to-end performance. Researchers who develop algorithms for various software tasks in the pipeline, such as clustering or trace reconstruction, therefore cannot properly evaluate their algorithms using simple synthetic datasets.

In this work, we present our end-to-end DNA-based data storage toolkit, including a simulator for the wetlab processes. Our toolkit can be used to encode a digital file into DNA strands, that can then be taken through the entire DNA storage pipeline, to be decoded and recovered at the end. Our toolkit includes an accurate simulator for the wetlab steps of the pipeline, which successfully captures the error profiles we observe when using different DNA synthesis and sequencing technologies. Importantly, our implementation of the DNA storage pipeline is modular, allowing users to easily swap in or out, their own implementation of any module of the pipeline, to evaluate the changes in the performance of the entire storage system. The toolkit contains implementations of

multiple algorithms for each step of the DNA storage pipeline, including the current state-of-the-art. Additionally, there are a number of novel algorithms implemented in the toolkit, each providing their own advantages to the storage system.

The rest of the paper is organized as follows. Section II covers the background information about DNA storage needed to understand our storage pipeline. Section III introduces our DNA storage pipeline and the individual steps it comprises. Section IV describes a flexible encoding module of our pipeline and the multiple functionalities it supports. Section V covers the challenges of simulating the wetlab steps and our wetlab simulator in detail. Sections VI and VII cover our baseline implementation and alternative algorithms for two most computationally intensive parts of the decoding pipeline, clustering and trace reconstruction, respectively. We discuss the pipeline as a whole with some further overall evaluations in Section IX, while other open-source implementations of individual modules and the related work are reviewed in section X.

II. Background

A. Motivation for DNA Storage

DNA storage is an emerging storage technology that offers many unique advantages. One of the most important is its incredible physical density, which is several orders of magnitude ahead of any alternative storage technologies [5], [6], [23], [25]. Furthermore, DNA as a storage medium offers extreme durability, measured in thousands or even millions of years, depending on the specific preservation methods used [13]. This is in stark contrast to conventional storage mediums that can retain data only for a few years before requiring costly acquisition of new storage and tedious data copying to the new medium.

Being a chemical form of data storage, many important data operations can be performed as chemical reactions. This brings significant advantages, such as convenient and low-cost data copying via simple sampling and the use of various polymerase chain reactions (PCR). Importantly, data stored in DNA can be accessed at random using PCR in largely constant time, regardless of the size of the data being searched [5], [6], [25]. Finally, the read-write interfaces for DNA storage will never become obsolete and will only improve in cost, latency and throughput over time due to the shared goals with human medicine, whereas all other storage technologies and their interfaces will eventually become obsolete [5], [6].

B. Challenges in DNA Storage

While the DNA storage technology is evolving rapidly, it has its own set of unique challenges that need to be addressed before it can become a viable alternative to traditional storage media. The primary obstacle to widespread adoption of DNA storage is the prohibitive cost of reads and writes [23], which impedes not only the commercial deployment of DNA storage, but also the research efforts. The latency of DNA read and write operations is also substantially higher compared to traditional storage media.

In DNA storage, data is read through the process of DNA sequencing, for which there are many available technologies. Data is written in the form of DNA molecules using a chemical process known as *artificial DNA synthesis*, which is readily available as a commercial service. This procedure can generate an arbitrary sequence of {A, C, G, T} characters (also known as *bases*, *nucleotides*), regardless of whether they have any biological meaning. While it is now technically feasible to synthesize DNA molecules exceeding 1000 bases in length [39], [40], the synthesis yield drops sharply beyond a few hundred bases and significant errors start to accumulate. This is why, in contrast to the vast natural DNA molecules found within living cells, the synthetic molecules produced by current synthesis technologies are typically limited to a few hundred bases in length. As a result, large data must be broken into pieces that fit into smaller DNA molecules, which can be synthesized economically [5], [10].

Due to its high density, durability, and latency, as well as its infancy, DNA storage is expected to sit at the bottom of the storage hierarchy, with the initial use case projected to be cold storage for archival data [5], where latency is not a critical factor. The read latency is primarily dominated by the DNA sequencing process, which can be several hours for high-throughput Next-Generation Sequencing (e.g., Illumina sequencing). While Nanopore sequencing offers real-time capabilities, its throughput is more restricted and the process introduces more errors. The write latency is primarily dominated by DNA synthesis and scales with the length of DNA molecules. The throughput of both reads and writes can be scaled out infinitely, but the question of cost becomes important.

C. Indexing

Since DNA has 4 bases {A, C, G, T}, we can encode up to 2 bits of information per character. As a result, a 200-base DNA molecule can store 50 bytes of information at the upper limit. The storage of substantial data thus necessitates the fragmentation of this data into smaller segments that can fit into shorter molecules. In contrast to conventional memory and storage technologies, where each byte has its physical address, we cannot establish a physical order between molecules in a test tube. Therefore, some form of internal address referred to as an *index* must be incorporated into each data fragment. The indices preserve the order of all molecules within the file they constitute and ensure that every molecule contains the necessary information for reconstructing the original data from its constituent parts [5], [15], [23], [25].

D. Data Encoding

To convert binary data into a DNA format for storage, it is essential to employ an adequate encoding scheme that translates the binary data into a sequence of {A, T, C, G} nucleotides. Various encoding schemes exist, each with its own trade-offs. Some schemes prioritize adhering to rules that facilitate the chemical processes involved in the pipeline, even if it means sacrificing coding efficiency. For instance, they may aim to

prevent the occurrence of extended runs of *homopolymers* (repeating bases like AAAA) to promote the success of certain types of DNA sequencing [25]. Others focus on balancing the *GC-content*, which is the ratio between the total number of G and C characters and the total length of a DNA molecule, in order to improve the DNA synthesis success [39], [40]. This type of coding is referred to as *constrained coding*, and most of the early work on DNA storage employs some form of it [2], [4], [5], [11], [16], [24]–[26], [29], [39], [40]. However, constrained coding leads to significant losses in information density [37]. In contrast, *unconstrained coding* does not seek to account for any particular error types. Instead, it employs simple data randomization to ensure that long homopolymers occur with low probability and the average GC-content is balanced [23], [34], [37]. Unconstrained coding achieves the maximum coding density of two bits per nucleotide, while relying on conventional error-correcting codes to handle all types of errors, achieving the higher overall coding efficiency and better resilience to any types of errors. In this work we employ unconstrained coding, assuming a simple mapping of two bits per nucleotide, while all error types are efficiently handled using outer Reed-Solomon ECC codes [23], [25], [34]. This approach has been shown to lead to much higher information density for all practical ranges of error rates [37]. It is possible, however, to substitute the encoding module of our pipeline with an arbitrary one.

Once a file has been divided into segments and encoded as DNA strings, we append a pair of special file-identifying sequences known as *primers* to the beginning and end of each string, as illustrated in Figure 2(a). These primers serve as a chemical tag, logically grouping related molecules together, and allowing us to fetch the entire group at once, thereby enabling random access. These tagged sequences are then sent to a commercial synthesis service, where millions of physical copies of each DNA string are synthesized collectively and stored in the same test tube, often referred to as a *DNA pool*.

E. Random Access and Decoding

In order to access a file stored within a DNA pool, a process called *DNA sequencing* is employed. First, to chemically isolate only the molecules containing fragments of the target file, the Polymerase Chain Reaction (PCR) is utilized. This reaction exponentially replicates a selected subset of the molecules (the selective exponential replication is also known as *amplification*). PCR is a parametrizable reaction, the parameters being two short (typically 20-nucleotide long) DNA sequences called *PCR primers*. After PCR, all molecules that begin with the first primer and end with the second primer will be amplified. PCR therefore essentially provides an addressing mechanism for random data access. Following the amplification of the target molecules, they are subjected to sequencing using one of the several DNA sequencing technologies available. The sequencing procedure produces multiple noisy DNA strings, commonly known as *reads*.

Throughout the synthesis, storage, wetlab manipulation, and sequencing stages, errors can occur, leading to discrepancies between the initially encoded DNA molecules and the final reads obtained. The average number of sequenced reads per originally synthesized molecule is termed *sequencing coverage* or *sequencing depth*. A higher coverage for a molecule simplifies its reconstruction from these reads but also increases the sequencing cost linearly.

All reads that possess the correct pair of primers are subsequently clustered based on their similarity, with the goal of ideally grouping together all reads originating from the same encoded DNA molecule. Typically, the Levenshtein (edit) distance [23], [25], [31] serves as the similarity metric for clustering. This distance is defined as the minimum number of insertion, deletion, or substitution operations needed to transform one string into another.

Each cluster contains multiple noisy copies of the same original DNA string. Subsequently, one of several *consensus-finding* algorithms [3], [23], [25], [33], [35], [39] is employed to produce the best estimate of the original DNA string from each cluster of related reads. In the field of information theory, this consensus-finding step is more formally referred to as *trace reconstruction*; we use the two terms interchangeably.

Following the process of clustering and reconstructing the most likely original strand from each cluster, the reconstructed strands undergo decoding to revert to binary data. The original file is recovered by rearranging its constituent pieces with the aid of the internal address (i.e., index) information embedded in each strand. Any remaining errors from previous steps in the pipeline are rectified using error-correction codes, typically employing an outer Reed-Solomon or LDPC code [13], [25]. These ECC schemes group a larger number of molecules, often tens of thousands [25], into encoding units organized as a matrix. This grouping facilitates erasure coding in the event of the losses of entire molecules and amortizes the cost of error correction across a more extensive dataset. For instance, in the state-of-the-art architecture [25], all reconstructed DNA molecules are treated as columns in a matrix, where separate DNA molecules are then generated to serve as an external redundancy, with each codeword representing a row in this matrix, as shown in Figure 2(b).

F. High-level Architecture

PCR provides an addressing mechanism for random data access through a pair of 20-nucleotide long unique sequences called primers. These primers must be designed to be sufficiently different from one another in Hamming distance [25]. As the primers of one file are logically unrelated to the primers of another file, there is no notion of hierarchy or distance between the files [34]. Therefore, the underlying architecture of the system becomes a *key-value store* [5], with a pair of primers constituting the key, and the payloads of all molecules tagged with the same pair of primers constituting the value.

III. Pipeline Overview

To encourage research on storage architectures and the algorithms for various steps of the DNA storage pipeline discussed in section II, our toolkit is modular, allowing different

modules to be swapped in and out for each step of the pipeline. Our DNA storage pipeline combines five modules, each performing one of the following steps:

- **Encoding**: Converting a file into the encoded strands that need to be synthesized for DNA storage. This includes the redundancy used for error correction, and is covered in Section IV.
- **Simulation**: Simulating the errors introduced by wetlab steps of the pipeline - synthesis, storage and sequencing. We variably replicate the strands and introduce errors into the synthesized strands to obtain noisy *reads* of each encoded strand. Covered in Section V.
- **Clustering**: Clustering the noisy reads such that each cluster contains reads from the same encoded strand. Covered in Section VI.
- **Trace Reconstruction**: Reconstructing the original encoded strand from each cluster of noisy reads. Covered in Section VII.
- **Decoding and Error Correction**: Using error correction codes to correct any error that might have been introduced by the previous steps in the pipeline and recovering the original file. The decoding scheme directly depends on the encoding scheme used and is thus covered along with it in Section IV.

Figure 1 illustrates the full pipeline with a number of available implementations for each step. In the following sections, we will discuss each module and its implementations.

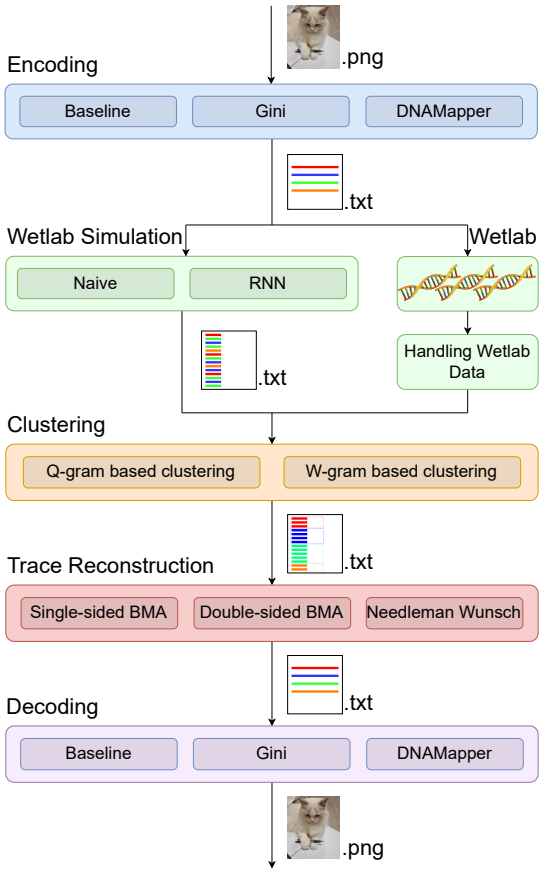

Fig. 1. An overview of the DNA storage pipeline.

IV. Encoding, Decoding and Error Correction

In the encoding step of the pipeline, the digital data is encoded into DNA strands to be synthesized. We implement three different encoding/decoding modules that follow the state-of-the-art encoding schemes and their respective decoding schemes [23], [25], along with Reed-Solomon error-correcting codes.

A. Implementation

Our baseline implementation strictly follows the DNA storage architecture presented by Organick et al. [25], and utilizes Reed-Solomon error-correcting codes (ECC). In this architecture, data molecules and ECC molecules are structured into a matrix, where each DNA molecule corresponds to a column in the matrix, and each row represents a Reed-Solomon codeword. The entire matrix serves as a single encoding/decoding unit. Implementation details can be obtained from the original work [23], [25].

One interesting observation is that in this architecture, errors like insertions and deletions within each molecule (column) are identified as substitution errors within the corresponding codeword (row) and are corrected accordingly. After these corrections are made, the data payload in each codeword is extracted and reassembled into the original data by concatenating them in the order determined by the index.

It should be noted that a big advantage of having the entire DNA storage pipeline is being able to design and improve modules with subsequent modules down the pipeline in mind.

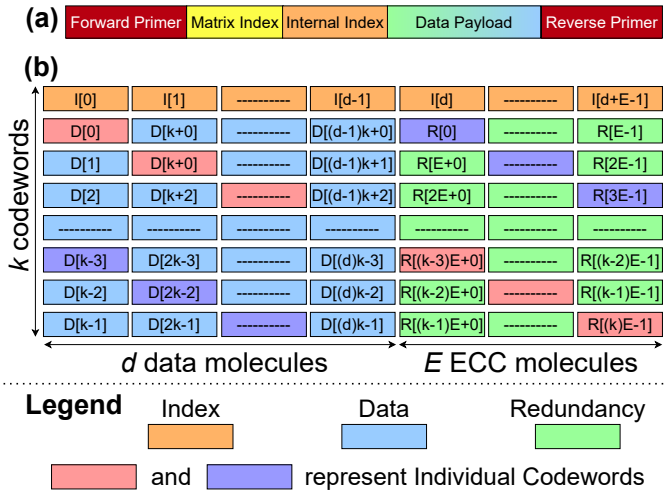

Fig. 2. **(a)** The configuration of a DNA molecule in the context of DNA-based storage as implemented in our pipeline [25]. The payload contains either the encoded data or the error correcting information.
b) The organization of data and ECC molecules within an encoding unit, structured as a matrix, as in the state-of-the-art Gini implementation, which spreads the codewords diagonally to equalize the reliability of different parts of each codeword, following [23]

For example, it has been observed that double-sided BMA used in the trace reconstruction step concentrates the errors onto the middle indexes [23], thus making the middle rows highly error prone and sometimes unrecoverable. This reliability skew is what encouraged Dehui, et al. [23] to develop the following two alternate encoding schemes, which can improve the accuracy of the decoded codewords.

B. Alternative: Gini

Since codewords are more error prone in the middle indexes, Gini [23] simply redistributes the codewords diagonally instead, in effect distributing the error prone middle indexes across all codewords evenly. Gini encoding is illustrated in Figure 2. This removes the positional bias which the original baseline implementation suffers from while still maintaining the ability to correct for erasures as a single molecule is still distributed across all codewords.

Where originally many copies of each molecule might be needed to rectify the most unreliable codewords, redistributing this unreliability evenly ensures that in general, lesser copies are needed to correct the errors for every codeword. Put differently, Gini is more reliably able to correct for errors given the same number of copies per molecule.

C. Alternative: DNAMapper

DNAMapper [23] provides an alternative way to deal with the reliability skew. The intuition is to categorise different bits based on their reliability needs, mapping data that require higher reliability onto more reliable indexes and conversely, mapping data with lower reliability needs onto less reliable indexes. In general, this mapping scheme is suitable for any data that has a concept of quality, such as images or videos. Data is decoded as per normal, just that now, the data from unreliable codewords are from bits that are more corruption-tolerant, such that the overall quality of the retrieved images or videos is maximized.

V. Simulation

The next module of the DNA storage pipeline encompasses a range of processes carried out within a wetlab setting, namely, DNA synthesis, storage, and sequencing. All of these processes introduce errors into the DNA strands. As a result, the DNA sequences obtained at the end through sequencing are noisy versions of the initially encoded strands. To address these errors, subsequent modules in the pipeline, such as clustering and trace reconstruction, are designed with the specific purpose of error correction. They are assessed based on their ability to faithfully restore the original, error-free DNA strand.

Beyond being error prone, these wetlab processes, particularly DNA sequencing and synthesis, are currently prohibitively expensive. Their high cost poses a significant obstacle to conducting large-scale experiments on DNA storage pipelines using authentic data. A commonly used practical solution to this problem is simulating the introduction of errors during the wetlab processes. Researchers can then effectively evaluate the performance of the subsequent modules in a more cost-efficient manner. In an ideal world, this simulation-based approach enables the assessment of error correction methods without the cost associated with real-world wetlab experimentation. However, accurately simulating these wetlab processes is challenging.

A. Current Limitations

While some sophisticated simulation techniques have been proposed [14], [41], most research in DNA storage uses fairly naïve simulations. We implement a baseline following this generalized data model as described by Rashtchian et al. [31]. In this model, the error profile is modelled using the Levenshtein (edit) distance. At every index of the given input strand, an insertion, deletion or substitution is introduced with probabilities p_I, p_D, p_S specified by the user. Each index of each strand is trialed independently with the same probabilities. This approach does not accurately replicate the errors found in the wetlab experiments. In reality, the sequenced data from wetlab experiments shows that the probability of error is dependent on the index, i.e., that the position of errors follows a certain distribution [18], and the likelihood of insertions, deletions, and substitutions is far from being equal. Furthermore, some of the error types often come in batches, whose length tend to follow a particular distribution [18].

Besides the flawed assumption that every index has identical error rates, any method that models the errors using edit distance inherently assumes that the output strand is always a result of the least number of edits from the original strand, when in reality this ground truth is unknown. In fact, even though errors can be modelled by insertions, deletions and substitutions, these transformations are only a proxy for the actual unknown edits that occurred. Thus, even if a large sample of data is used to estimate the probabilities of insertions, deletions and substitution at the level of every index, we find that it would not be able to accurately simulate real world output from the wetlab module. Prior work [14], [32], [41] also evaluates the accuracy of their simulation by comparing the distributions of different error types in the simulated datasets to the error distributions in real datasets. Such evaluations suffer from the same incorrect assumptions.

However, a key advantage of having the whole pipeline available is the ability to assess the performance of individual modules in relation to the whole pipeline. We can evaluate how realistic the different approaches to simulating the errors of a wetlab are, by comparing the performance of the trace reconstruction module on real data against the simulated data. Thus when designing our simulator, we evaluate the simulator using the following metrics (see table I):

(i) Error rate at different indexes of the reconstructed strands using real data versus simulated data. A better simulation will have highly similar error rates to that of real data after reconstruction. (Figure 3)

(ii) Average value of (i) over all indexes. A better simulation will have a similar average.

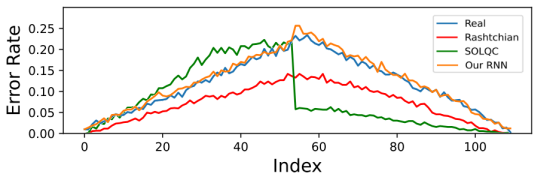

Fig. 3. Comparison of different wetlab simulations against real data from wetlab. The error rate (y-axis) here refers to the proportion of bases wrongly reconstructed from the noisy reads for the specified index.

(iii) Average value of the absolute difference of (i) over all indexes. A better simulation will have an average closer to 0.

(iv) The number of perfectly reconstructed strands when using real data versus simulated data. The closer the numbers, the better the simulation.

In general, the errors introduced in a real wetlab setting are much more complex than the ones introduced by previous simulations. As a result, reconstructing the original strand from the simulated datasets is very easy in comparison to the real datasets. Figure 3 compares the performance of our trace reconstruction module – specifically the double-sided BMA algorithm [23] – on data obtained from our baseline simulation based on Rashtchian et al. [31], SOLQC, a probabilistic model with probabilities conditioned on the nucleotide level for insertion, deletion and substitution errors, as described and implemented in prior work [8], [32], and real sequenced data from a wetlab [35]. It should be noted that the SOLQC model also simulates pre-insertions with a certain probability but not post-insertions, which results in the forward reconstruction being much harder than the reverse reconstruction.

B. Data-Driven Simulator

Modeling errors, as outlined earlier, presents a multi-faceted challenge. These errors can arise from various pipeline stages, spanning synthesis, storage, manipulation, and sequencing. Additionally, there is a clear position-dependence to these errors on the strand's base. To model such layered, position-specific noise, we employ learning-based methods, drawing upon the expansive paired clean-noisy strand dataset from prior work [35]. Two key considerations shape our methodology: (1) Ensuring proper alignment between corresponding nucleotides of input and output strands is crucial, as insertion and deletion errors can cause length discrepancies; (2) Our model needs to efficiently manage long sequences, which proves problematic for simple stacks of traditional recurrent neural networks (RNNs) [1].

Given the analogous nature of sequence-to-sequence transformations in our problem and neural machine translation (NMT), we draw inspiration from established NMT methodologies. Both domains necessitate the generation of new sequences conditioned on provided inputs, and ensuring precise alignment between these sequences is paramount. Consequently, we adopt the attention-based encoder-decoder architecture [1], prevalent in NMT, for our problem. We design a neural network to directly model $\Pr(s_{\text{noisy}}|s_{\text{clean}})$, where s indicates DNA strands. During inference, an auto-regressive decoder generates

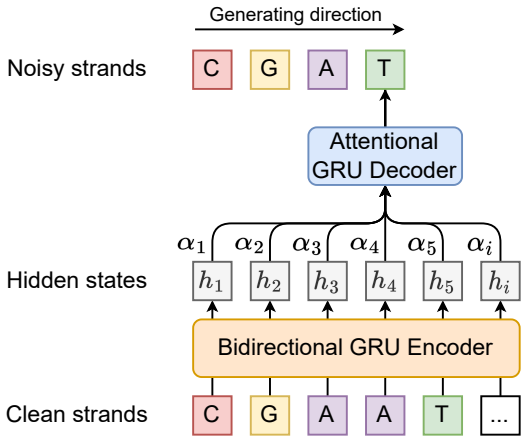

Fig. 4. The sequence-to-sequence RNN structure of our simulator. h_i: hidden states of RNN. α_i: attention weight.

probability distribution predictions of nucleotides from left to right, position-by-position, and a sampling method, e.g., greedy or beam sampling, can be adopted to obtain the sequence output. During training, the model learns the distribution of the subsequent nucleotide in the noisy strand, conditioned on the preceding nucleotides and the provided clean strand in the input.

As depicted in Figure 4, the encoder (orange) consists of a bi-directional recurrent neural network (RNN) designed to transform the raw input sequence into a set of "annotations." Each annotation h_i encompasses information from the entire input sequence, but primarily emphasizing the contextual information surrounding the i-th (nucleotide) in the sequence.

The decoder, represented by the blue box in Figure 4, is a unidirectional RNN that ingests the annotations and previously generated nucleotides to predict the probabilities for the next potential nucleotide. Crucially, before this probability prediction, the decoder employs an attention mechanism—a set of trainable parameters—to determine which encoder annotations are particularly relevant for the current generation step. Instead of simply relaying all encoder annotations directly to the decoder, an attention-weighted average of these annotations is passed. Annotations deemed more relevant are assigned higher weights. In our implementation, both the encoder and decoder RNNs utilize Gated Recurrent Unit (GRU) [9] cells due to their resistance to overfitting compared to traditional Long Short-Term Memory (LSTM) cells. For our optimal model configuration, the hidden layer size is set at 128, and both the encoder and decoder comprise a single GRU layer. For the decoding process, we adopt greedy sampling. In this approach, once the token probabilities for a specific position are determined, immediate sampling ensues.

Figure 3 compares our RNN model against the same real data. While previous simulations produce noisy strands that behave very differently from the real data from a wetlab, we can see that the noisy strands produced by our RNN model closely replicate the difficulty of reconstructing real wetlab data. In table I, for the 3 other metrics from Section V-A, we

TABLE I
(II) RNN MODEL HAS THE CLOSEST AVERAGE ERROR RATE ACROSS ALL INDEXES TO THE REAL DATA. (III) RNN MODEL ERROR RATE DEVIATES THE LEAST FROM REAL DATA. (IV) RNN MODEL HAS THE CLOSEST NUMBER OF PERFECTLY RECONSTRUCTED STRANDS TO REAL DATA.

	Rashtchian	SOLQC	RNN	Real
(ii)	7.21%	8.17%	12.38%	11.808%
(iii)	4.59%	6.20%	0.80%	-
(iv)	546	413	338	332

can see the RNN model also behaves the most similarly to the real wetlab data. In this expriment we use a dataset of $\tilde{2}70$K DNA reads belonging to $\tilde{1}0$K clusters [35]. We have divided the clusters into a test:validation:train split of 7988:998:998.

VI. CLUSTERING

Once we receive the noisy reads from the simulation module, we need to cluster them based on some measure of similarity. This is done in order to correct for errors introduced during the wetlab phase. Ideally each cluster will contain reads of the same originally encoded strand. Once these reads are grouped together into clusters, we can find the consensus encoded strand for each cluster in the next module.

A. Baseline Implementation

Edit distance is the most commonly used similarity metric when clustering DNA strands. However, edit distance computations are very expensive. Thus, designing a fast and efficient algorithm for clustering DNA strands involves using as few edit distance comparisons as possible. Another requirement of the algorithm is to be distributed to efficiently utilize all the resources available to the system. Our baseline clustering algorithm implementation follows the distributed clustering algorithm described by Rashtchian et al. [31]. In this implementation, every strand begins as a singleton cluster and clusters that are highly similar are iteratively merged until sufficiently different clusters remain.

To briefly describe the algorithm, at the beginning of the clustering, each DNA strand forms an individual singleton cluster. In the first step, a random sequence of k bases is designated as an *anchor*. Subsequently, a representative from each cluster is chosen at random, and the clusters are partitioned using the l bases following the first appearance of the anchor in the representative sequence.

After this partitioning, within each partition, a collection of substrings, each with a length of q and referred to as *q-grams*, is randomly selected. From each cluster within the partition, a representative is randomly sampled. For each representative, a binary string known as a q-gram signature is generated, where the presence and absence of q-grams in the representative are denoted by '1' and '0' in the corresponding bit positions.

Now instead of performing edit distance comparisons between the representatives from two different clusters, we can compare two clusters by calculating the Hamming distance between the q-gram signatures of their representatives. Hamming distance calculations significantly outpace edit distance computations, resulting in considerably quicker comparisons.

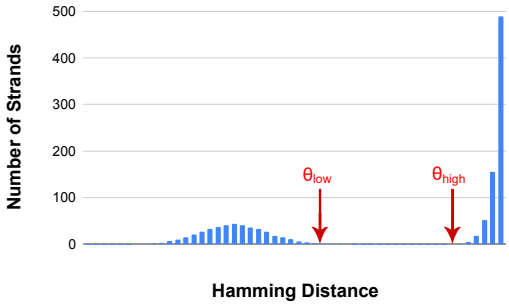

Fig. 5. Automated configuration for clustering: θ_{low} and θ_{high} are determined by plotting this graph using a small sample of strands.

If representatives from two distinct clusters within the same partition are very close in q-gram signatures, the two clusters are merged. If the Hamming distance between them is too high, the clusters are not merged. Thus for this algorithm, a threshold is pre-determined for the Hamming distance between q-gram signatures. Only if the Hamming distance lies between the threshold of too high and too low, an edit distance comparison is made to make the merge decision. Thus, edit distance comparisons are avoided as much as possible. After each round of clustering, this entire process is repeated with a fresh, randomly chosen anchor sequence.

B. Automatic Configuration for Clustering

In prior work [31], the thresholds for Hamming distance between q-gram signatures are tuned by the user to obtain optimal clustering. In our implementation, we have provided a parameter to automate the calculation of threshold values of Hamming distance. This is done by sampling a handful of reads from the entire input and calculating the Hamming distance of q-gram signatures between them and a larger random sample from the remaining reads. This Hamming distance is observed to be plotted as shown in Figure 5. We can use this plot to easily determine the threshold values. In the figure, the lower threshold is indicated by θ_{low} and if the Hamming distance is below θ_{low}, we merge the clusters without making an edit distance comparison. The upper threshold in the figure is marked by θ_{high} and if the Hamming distance is above θ_{high}, the algorithm chooses to simply not merge the clusters without needing to make an edit distance comparison. It is only when the Hamming distance is between θ_{low} and θ_{high} that the algorithm calculates the edit distance between the two cluster representatives.

C. Our Alternative Algorithm: w-grams

In our novel alternative construction, we keep most of the q-gram algorithm as is, but instead of pre-calculating q-gram signatures as in the baseline, we calculate the *w-gram* signatures instead. While a q-gram signature is made up of 0s and 1s to indicate the presence or absence of a set of random substrings in the string, a w-gram signature consists of the position of the first occurrence of the substrings within the string. The Hamming distance then needs to be replaced by the L1 norm for calculating the distance between the w-gram signatures. Outside of these two changes, the algorithm remains the same as the baseline.

TABLE II
COMPARING TWO CLUSTERING ALGORITHMS (AVG OVER 10 RUNS)

Error Rate	Clustering Accuracy		Clustering Time (in seconds)		Signature Calculation Time (in seconds)		Overall Time (in seconds)	
	q-gram	w-gram	q-gram	w-gram	q-gram	w-gram	q-gram	w-gram
0.03	0.9922	0.9923	4	4	1	2	5	6
0.06	0.9919	0.992	9	10	1	2	10	11
0.09	0.9908	0.9913	13	19	1	2	14	21
0.12	0.9883	0.9894	29	35	1	2	30	37
0.15	0.9823	0.9845	56	71	1	2	57	73

These changes make the pre-calculation and storage of signatures more expensive in space and slightly more expensive in time. They however increase the distance between the signatures of clusters significantly, which further decreases the number of edit distance comparisons that the algorithm needs to make. It also decreases the few mistakes being made during clustering because of Hamming distance comparisons. As can be observed in table II, at a coverage of 10, this updated algorithm consistently increased the accuracy of the clustering, at the cost of a low increase in runtime, with the increase in performance improving with the increase in overall error rate.

VII. RECONSTRUCTION

In the trace reconstruction module, we recreate the originally encoded DNA strands from each cluster of noisy reads produced in the clustering step. We implement three different algorithms for this module.

A. Baseline Implementation

The baseline implementation follows the BMA-lookahead algorithm for DNA-based data storage as proposed by Organick et al. [25]. For every cluster, a consensus strand is incrementally recreated from left to right by using the noisy strands in the cluster in the following way: Every noisy strand maintains an individual pointer that starts at the beginning of the strand. In each step, a majority vote is taken to determine the most likely base in the consensus strand. Strands that agree with this base have their pointers incremented by one as normal, and this new index is used for the reconstruction of the next base in the consensus strand. However, if a strand does not agree with the majority base, in order to continue using it to find the consensus, the algorithm must first determine the most likely edit that occurred in the strand so that it can properly align it with the rest of the noisy reads. This is achieved by looking ahead at the next few bases to guess whether an insertion, deletion or substitution has occurred, and the pointer of this strand is changed according to this assumption. The algorithm then moves on to reconstruct the next base by taking consensus of the next set of pointers, and continues taking majority vote in a similar manner until the whole consensus strand is reconstructed.

Note that any misalignment of a noisy strand, caused by a wrong assumption of the edit, propagates to later indexes. This reduces the chance of correctly reconstructing the original base for the following indexes. Thus, the longer the strand, the more unreliable the reconstruction of later indexes becomes.

Fig. 6. Double Sided algorithms reduce the peak error rate and concentrates errors in the middle indexes. Needleman Wunsch outperforms prior work.

B. Alternative: Double-sided BMA

A key observation from the BMA-lookahead algorithm is that the propagation of error is symmetric. This means that if the consensus strand was instead being reconstructed from right to left with pointers starting at the end of the noisy reads, it would be the earlier indexes that are more unreliably reconstructed. This observation motivated the strategy of Double-sided BMA [23], which is to first reconstruct the left half of the consensus strand from left to right using the BMA-lookahead algorithm from left to right, and then to reconstruct the right half of the consensus strand by using the BMA-lookahead algorithm from right to left. The two halves are then joined to create the final consensus strand. Since the two reconstructed halves iterate through only half the length of the noisy strands, the errors will only be propagated halfway through, to the middle of the strand where we observe a peak in unreliable reconstruction at the middle indexes. The different error propagation in BMA-lookahead and double-sided BMA algorithms can be observed in Figure 6.

C. Our Alternative Algorithm: Needleman Wunsch

In the BMA reconstruction methods, conflicting strands are realigned by looking ahead to and using the next few bases. However, we can instead first compute the multiple sequence alignment of all noisy strands within a cluster by using the Needleman-Wunsch algorithm [20], [21] to compute the global alignments that minimizes edit distance. Once we have this

alignment, we can reconstruct the consensus strand by merely taking the majority vote at every index.

Sometimes this alignment might exceed the expected length of the original encoded strands. In such cases, we can compute the number of characters exceeded, x, and choose the x most unreliable indexes to omit in the final consensus strand. We do this by counting the number of insertion/deletion alignments ('_' character) at every index for every aligned read, and omitting the x indexes with the most number of indels.

This simple algorithm is our construction and is built on top of the SIMD partial-order alignment [36] implementation of Needleman-Wunsch as described in prior work by Christopher Lee [20]. As can be seen in Figure 6, our Needleman-Wunsch algorithm is much more accurate than prior work. As shown in Table III, it is also significantly faster, with the improvement in speed increasing with increasing coverage.

VIII. Handling Wetlab Data

In an ideal scenario, rather than simulating the wetlab processes, the encoded DNA strands could be synthesized, preserved in a wetlab environment, and subsequently sequenced when we want to retrieve the data. The sequencing process, whether utilizing Illumina or Nanopore sequencing technologies, yields data in the fastq file format. Within our toolkit, we have incorporated a dedicated module designed to manage this sequenced data, effectively allowing it to seamlessly replace the simulation module.

To facilitate the transition of sequenced data from the wetlab to the clustering module, a preprocessing step is required to align its format with that of our simulated data. Initially, the fastq file is converted into a text file format, ensuring compatibility with the wetlab simulation module. Furthermore, note that the sequenced reads obtained from the wetlab exist in both forward and reverse directions, necessitating consideration of DNA strand directionality within our pipeline. To address this, we perform a transformation, converting the 3' to 5' DNA strands into the 5' to 3' orientation, thereby aligning them with the conventions of our simulator. This is done by comparing the primers in the reads with the library of primers used. Before entering the clustering phase, a last critical step is to remove the primers from the sequenced reads. Only the core payload information is retained and passed on to the clustering module for subsequent processing.

IX. Overall Pipeline Evaluation

We evaluate the DNA Storage Toolkit by performing some experiments on the entire pipeline. To properly evaluate its modularity, we isolate and test individual components of the pipeline to ensure they work correctly and independently. We systematically alter one component at a time while keeping the others constant, in order to debug any issues with dependencies or interactions that might exist within the pipeline. The latency breakdown is shown in table III and the evaluations are performed on an Intel Xeon Gold 5118 server with 24 CPU cores, 2 threads per core.

For one image, instead of using the wetlab simulation, we had the encoded strands synthesized into DNA by Twist BioScience. We performed PCR to amplify these strands and then sequenced the file using Nanopore sequencing to obtain the noisy reads of the synthesized strands. These noisy reads were then passed on to the clustering module and the rest of the pipeline was evaluated normally. We were able to successfully retrieve the image as encoded, verifying the end-to-end nature of the pipeline.

When we evaluate all modules of our pipeline in a single thread, we find that the most computationally intensive and thus slowest step by far is clustering. In a working DNA storage system we would have to cluster many billions if not many trillions of reads from every sequencing run. We ensure scalability by harnessing multiple threads and optimized resource distribution for the implementation of the distributed clustering algorithms in our toolkit. We can observe in table III that the difference between the runtime of q-gram and w-gram clustering increases with coverage, making w-gram unsuitable for high coverage settings.

A. Time Complexity

Different modules of the pipeline vary greatly in terms of the computational complexity and amenability to parallelization. In most practical settings clustering runtime tends to be the dominant factor.

Encoding and decoding are dominated by ECC computation, which is in the worst case cubic in the codeword length. Every codeword can be encoded/decoded in parallel.

Clustering is subquadratic in terms of the number of reads. The exact complexity is

$\mathcal{O}(\max(n^{1+\mathcal{O}(p)}, \frac{n^2}{m^{\Omega(1/p)}})) \cdot (1 + \frac{\log(s/\epsilon)}{s})$ [31]

where n is the total number of reads, m is the number of underlying clusters, p is the average probability of errors (insertions, deletions, substitutions) per nucleotide, and accuracy of $1 - \epsilon$. Our w-gram variant further improves the constant factor at lower sequencing coverage. Our clustering implementation supports parallelization in both multi-threaded and distributed settings.

Simulation complexity for both naive and RNN is $\mathcal{O}(n \cdot L)$, where n is the number of reads and L is the strand length. The constant factor is higher for RNN.

The complexity of trace reconstruction for both single- and double-sided BMA is $\mathcal{O}(n \cdot L)$, where n is the size of the cluster and L is the strand length. The complexity of the Needleman-Wunsch approach is $\mathcal{O}(L \cdot N \cdot Np)$ [21] , where N is the number of nodes in the partial order graph and Np is the average number of predecessors per node. In all approaches the clusters are reconstructed in parallel.

X. Related Work

While there is no prior openly available end-to-end DNA data storage pipeline, there are many prior works that have made available individual modules or a combination of modules from the pipeline. By combining these modules with the modules in our toolkits, different complete pipelines can be created.

TABLE III
LATENCY OF THE MODULES IN OUR DNA STORAGE TOOLKIT IN SECONDS. (SETTING: BASELINE ENCODING, PAYLOAD LENGTH = 120, ERROR RATE = 6%, AVG OVER 10 RUNS.)

Pipeline	Encoding	Clustering	Recon	Decoding	Total
Coverage = 10					
q-gram + BMA	108	22	13	2	145
q-gram + DBMA	108	22	25	2	157
q-gram + NWA	108	22	11	2	143
w-gram + BMA	108	23	13	2	146
w-gram + DBMA	108	23	25	2	158
w-gram + NWA	108	23	11	2	144
Coverage = 50					
q-gram + BMA	108	68	167	2	345
q-gram + DBMA	108	68	329	2	507
q-gram + NWA	108	68	51	2	229
w-gram + BMA	108	128	167	2	405
w-gram + DBMA	108	128	329	2	567
w-gram + NWA	108	128	51	2	289

DBMA: Double-sided BMA; NWA: Needleman Wunsch

Recently, Ping et al. [27] presented the Yin–Yang codec, attempting to achieve a physical information density close to the theoretical maximum. Their prior work Chamaeleo [28] provides a toolkit to evaluate codecs for DNA storage and evaluates some popular codec available at the time. It should be noted that these evaluations were made with very simple simulations methodologies which are generally not very representative of real wetlab experiments. In the completely opposite direction, DNA-Aeon [38] presents a codec that prioritizes handling of any issues that might occur during the wetlab activities of the storage pipeline, ranging from introduced errors to very skewed sequencing distributions. Another alternate DNA codec available to us is the ADS Codex [17], which is a DNA storage codec released by the Los Almos National Laboratory, which is focused on offering exceptional high data density while remaining adaptable to various needs related to DNA synthesis and sequencing, for example adhering to restrictions on the longest homopolymers allowed.

While not designed for DNA-based data storage, the DeepSimulator by Li et al. [22] uses a Bi-LSTM to simulate the entire Nanopore sequencing pipeline. In the initial Deep Learning phase, DeepSimulator accepts a sequence of bases as input and produces electrical current levels corresponding to the sequencing process as output. In the second step, these current levels are transmitted to a basecaller, which interprets the current data and translates it into the corresponding bases. Similar to prior work [14], we found that the DeepSimulator is not directly successful in simulating the DNA storage model. Hamoum et al. [14] build a probabilistic model similar to noisy channel models to simulate the errors introduced in Nanopore sequencing.

Yuan et al. [41] have released a probabilistic simulator model for the wetlab steps in DNA storage which incorporates some domain knowledge so that it can be easily trained for different sequenced datasets. Their implementation is quite modular and can be easily slipped in place of our simulation module.

Compared to the clustering algorithm implemented in our toolkit [31], Clover [30] presents a low memory consumption algorithm for clustering DNA reads. The algorithm constructs a multiple tree structure to search for a specified interval of DNA sequences, thus avoiding computation of the Levenshtein distance, which is the most expensive step in most naive clustering algorithms.

Sabary et al. [33] proposed a few different trace reconstruction algorithms with implementations made available in a user-friendly GUI tool. This work was packaged with two other modules for some basic wetlab simulations and clustering, and presented at the Non-Volatile Memories Workshop as DNA-Storalator [8]. The tool contains good implementations of many different trace reconstruction algorithms. However, it is crucial to emphasize that the three modules implemented in their work remain disconnected from each other, with the clustering module solely providing a performance report rather than returning the generated clusters. Because the clustering output is unavailable, the trace reconstruction implementation directly relies on the ideal clusters produced by their simulation module.

Although Reed-Solomon codes serve as the prevailing framework for error management in most proposed DNA storage architectures, some prior research has explored alternative approaches. In contrast to relying on Reed-Solomon codes, Chandak et al. [7] have made available a novel scheme that departs from the conventional separation paradigm. Their alternative scheme adopts a single, extensive block-length LDPC code, effectively addressing both erasure and error correction tasks. In conjunction with this approach, they incorporate specific heuristics designed to effectively manage insertion and deletion errors. The previously discussed channel model [14] also combines non-binary LDPC codes with an implementation of constrained consensus sequence algorithm for DNA storage [19] for its error correction mechanism.

XI. CONCLUSION

While the remarkable advancement in DNA synthesis and sequencing technologies has ignited a surge of interest in leveraging DNA as a compact and durable medium for data storage, a comprehensive toolkit encompassing all essential components of the DNA-based data storage pipeline for research purposes has been notably absent. In response to this critical gap, in this work we introduce a versatile, end-to-end DNA data storage toolkit that guides an input file through the entire DNA storage pipeline. Our contribution encapsulates implementations of state-of-the-art techniques, and at some junctures of the pipeline introduces novel algorithms for the tasks. These key tasks encompass data encoding into DNA strands, simulation of the intricate wetlab processes such as synthesis, storage, and sequencing, clustering of the sequenced outcomes, trace reconstruction of clusters, and the final decoding of the originally encoded file with robust error-correction mechanisms. Each module within our toolkit stands as an independent entity, capable of individual utilization or seamless integration into a holistic pipeline. Our overarching objective is to empower researchers by providing a highly modular toolkit that encapsulates existing DNA storage methodologies

while inviting innovation through easy integration of any novel algorithms. For quick and inexpensive experiments during research, we also offer a robust simulator that faithfully mirrors real-world error profiles encountered when employing diverse DNA synthesis and sequencing technologies.

Availability

Our DNA storage toolkit is available at https://github.com/prongs1996/DNAStorageToolkit.

Acknowledgments

The authors would like to thank Cyrus Rashtchian for his help with the understanding of the trade-offs around the clustering algorithm. This research was supported by the Advanced Research and Technology Innovation Centre (ARTIC) at the National University of Singapore under grant FCT-RP1 A-0008129-00-00, and by the Ministry of Education in Singapore grants A-0008143-00-00 and A-0008024-00-00.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] Krishna Gopal Benerjee and Adrish Banerjee. On homopolymers and secondary structures avoiding, reversible, reversible-complement and gc-balanced dna codes. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 204–209. IEEE, 2022.

[3] Vinnu Bhardwaj, Pavel Arkadevich Pevzner, Cyrus Rashtchian, and Yana Safonova. Trace reconstruction problems in computational biology. In *IEEE Transactions on Information Theory*, 2020.

[4] Swapnil Bhatia and Kevin Gildea. A combinatorial writing scheme for high throughput dna-based data storage. In *20th USENIX Conference on File and Storage Technologies (FAST)*, 2022.

[5] James Bornholt, Randolph Lopez, Douglas Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[6] Luis Ceze, Jeff Nivala, and Karin Strauss. Molecular digital data storage using dna. *Nature Reviews Genetics*, 20(8):456–466, 2019.

[7] Shubham Chandak, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, Tsachy Weissman, and Hanlee Ji. Improved read/write cost tradeoff in dna-based data storage using ldpc codes. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 147–156. IEEE, 2019.

[8] Gadi Chaykin, Nili Furman, Omer Sabary, Dvir Ben-Shabat, and Eitan Yaakobi. Dna-storalator: End-to-end dna storage simulator, 2022.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[10] George McDonald Church, Yuan Gao, and Sriram Kosur. Next-generation digital information storage in DNA. In *Nature*, 2013.

[11] Danny Dubé, Wentu Song, and Kui Cai. Dna codes with run-length limitation and knuth-like balancing of the gc contents. In *IEEE Symposium on Information Theory and its Applications (SITA), Japan*, 2019.

[12] Yaniv Erlich and Dina Zielinski. Dna fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.

[13] Robert Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin Jan Stark. Robust chemical preservation of digital information on DNA in silica with error-correcting codes. In *Angewandte Chemie International Edition*, 2015.

[14] Belaid Hamoum, Elsa Dupraz, Laura Conde-Canencia, and Dominique Lavenier. Channel model with memory for dna data storage with nanopore sequencing. In *2021 11th International Symposium on Topics in Coding (ISTC)*, pages 1–5. IEEE, 2021.

[15] Reinhard Heckel, Ilan Shomorony, Kannan Ramchandran, and David Tse. Fundamental limits of DNA storage systems. In *International Symposium on Information Theory*, 2017.

[16] Kees Antonie Schouhamer Immink and Kui Cai. Properties and constructions of constrained codes for dna-based data storage. *IEEE Access*, 8:49523–49531, 2020.

[17] Latchesar Ionkov, Bradley Settlemyer, and Dominic Manno. Ads codex: The adaptive codec for organic molecular archives. https://github.com/lanl/adscodex.

[18] Mayank Keoliya, Puru Sharma, and Djordje Jevdjic. Simulating noisy channels in dna storage. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2022.

[19] Dominique Lavenier. Constrained consensus sequence algorithm for dna archiving. *arXiv preprint arXiv:2105.04993*, 2021.

[20] Christopher Lee. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics*, 19(8):999–1008, 2003.

[21] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[22] Yu Li, Renmin Han, Chongwei Bi, Mo Li, Sheng Wang, and Xin Gao. Deepsimulator: a deep simulator for nanopore sequencing. *Bioinformatics*, 34(17):2899–2908, 2018.

[23] Dehui Lin, Yasamin Tabatabaee, Yash Pote, and Djordje Jevdjic. Managing reliability skew in dna storage. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, 2022.

[24] Tuan Thanh Nguyen, Kui Cai, Kees A Schouhamer Immink, and Han Mao Kiah. Capacity-approaching constrained codes with error correction for dna-based data storage. *IEEE Transactions on Information Theory*, 67(8):5602–5613, 2021.

[25] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher N Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze, and Karin Strauss. Random access in large-scale DNA data storage. In *Nature biotechnology*, 2018.

[26] Seong-Joon Park, Yongwoo Lee, and Jong-Seon No. Iterative coding scheme satisfying gc balance and run-length constraints for dna storage with robustness to error propagation. *Journal of Communications and Networks*, 24(3):283–291, 2022.

[27] Zhi Ping, Shihong Chen, Guangyu Zhou, Xiaoluo Huang, Sha Joe Zhu, Haoling Zhang, Henry H Lee, Zhaojun Lan, Jie Cui, Tai Chen, Wenwei Zhang, Huanming Yang, Xun Xu, George McDonald Church, and Yue Shen. Towards practical and robust dna-based data archiving using the yin–yang codec system. *Nature Computational Science*, 2(4):234–242, 2022.

[28] Zhi Ping, Haoling Zhang, Shihong Chen, Qianlong Zhuang, Sha Zhu, and Yue Shen. Chamaeleo: an integrated evaluation platform for dna storage. *Synthetic Biology Journal*, pages 1–15, 2020.

[29] William Henry Press, John A Hawkins, Stephen K Jones Jr, Jeffrey M Schaub, and Ilya J Finkelstein. Hedges error-correcting code for dna storage corrects indels and allows sequence constraints. *Proceedings of the National Academy of Sciences*, 117(31):18489–18496, 2020.

[30] Guanjin Qu, Zihui Yan, and Huaming Wu. Clover: tree structure-based efficient dna clustering for dna-based data storage. *Briefings in Bioinformatics*, 23(5):bbac336, 2022.

[31] Cyrus Rashtchian, Konstantin Makarychev, Miklos Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for DNA data storage. In *Advances in Neural Information Processing Systems*, 2017.

[32] Omer Sabary, Yoav Orlev, Roy Shafir, Leon Anavy, Eitan Yaakobi, and Zohar Yakhini. Solqc: Synthetic oligo library quality control tool. *Bioinformatics*, 37(5):720–722, 2021.

[33] Omer Sabary, Alexander Yucovich, Guy Shapira, and Eitan Yaakobi. Reconstruction algorithms for DNA storage systems. In *International Conference on DNA Computing and Molecular Programming*, 2020.

[34] Puru Sharma, Cheng-Kai Lim, Dehui Lin, Yash Pote, and Djordje Jevdjic. Efficiently enabling block semantics and data updates in dna storage. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 555–568, 2023.

[35] Sundara Rajan Srinivasavaradhan, Sivakanth Gopi, Henry D. Pfister, and Sergey Yekhanin. Trellis bma: Coded trace reconstruction on ids channels

for dna storage. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 2453–2458, 2021.

[36] Robert Vaser. spoa: Simd poa. https://github.com/rvaser/spoa.

[37] Franziska Weindel, Andreas L Gimpel, Robert N Grass, and Reinhard Heckel. Embracing errors is more efficient than avoiding them through constrained coding for dna data storage. *arXiv preprint arXiv:2308.05952*, 2023.

[38] Marius Welzel, Peter Michael Schwarz, Hannah F Löchel, Tolganay Kabdullayeva, Sandra Clemens, Anke Becker, Bernd Freisleben, and Dominik Heider. Dna-aeon provides flexible arithmetic coding for constraint adherence and error correction in dna storage. *Nature Communications*, 14(1):628, 2023.

[39] SM Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and error-free DNA-based data storage. In *Nature Scientific Reports 7*, 2017.

[40] SM Hossein Tabatabaei Yazdi, Yongbo Yuan, Jian Ma, Huimin Zhao, and Olgica Milenkovic. A rewritable, random-access DNA-based storage system. In *Nature Scientific Reports 5*, 2015.

[41] Lekang Yuan, Zhen Xie, Ye Wang, and Xiaowo Wang. Desp: a systematic dna storage error simulation pipeline. *BMC bioinformatics*, 23(1):1–14, 2022.