CMOSS: A Reliable, Motif-based Columnar Molecular Storage System

Eugenio Marinelli*
EURECOM
Biot, France
eugenio.marinelli@eurecom.fr

Yiqing Yan*
EURECOM
Biot, France
yiqing.yan@eurecom.fr

Lorenzo Tattini
EURECOM
Biot, France
lorenzo.tattini@eurecom.fr

Virginie Magnone
IPMC
Nice, France
magnone@ipmc.cnrs.fr

Pascal Barbry
IPMC
Nice, France
barbry@ipmc.cnrs.fr

Raja Appuswamy
EURECOM
Biot, France
raja.appuswamy@eurecom.fr

ABSTRACT

The surge in demand for cost-effective long-term archival media, coupled with density limitations of contemporary magnetic media, has resulted in synthetic DNA emerging as a promising new alternative. Despite its benefits, storing data in DNA poses several challenges as the technology used for reading/writing data on DNA are highly error prone. Thus, it is important to design pipelines that can efficiently use redundancy to mask errors without amplifying read/write cost. In this work, we present Columnar MOlecular Storage System (CMOSS), a novel, end-to-end DNA storage pipeline that can provide error-tolerant data storage at low read/write costs. CMOSS differs from state-of-the-art (SOTA) on three fronts (i) a motif-based, vertical layout in contrast to nucleotide-based horizontal layout, (ii) integrated consensus calling and decoding enabled by the vertical layout, and (iii) a flexible, block-based data organization for random access over DNA storage in contrast to object-based organization. Using an in-depth evaluation with several simulated and real wet lab experiments, we demonstrate the benefits of CMOSS design.

CCS CONCEPTS

• **Information systems** → *Information storage technologies*; *Storage architectures*; • **Hardware** → *Memory and dense storage*; • **Computer systems organization** → **Redundancy**; **Reliability**.

*Both authors contributed equally to this research.

SYSTOR '24, September 23–24, 2024, Virtual, Israel

ACM ISBN 979-8-4007-1181-7/24/09.
https://doi.org/10.1145/3688351.3689162

KEYWORDS

molecular storage, DNA storage, PCR, random access, long-term archival, error control coding, read consensus

1 INTRODUCTION

The global datasphere is expected to reach 125ZB by 2025 [16]. However, over 80% of data generated is "cold", and corresponds to data that needs to be archived to meet safety and compliance requirements [25]. Archival data is the fastest growing segment with over 60% cumulative annual growth rate [46]. Thus, as enterprises continue migrate to the cloud, cloud vendors are in need of storage technologies that can provide durable, low-cost storage of archival data for decades. One such novel storage medium that has received a lot of attention recently is synthetic DNA. DNA can store up to 1EB of data in a cubic millimeter [13], making it seven orders of magnitude denser than tape [15]. It can last several millennia when stored under proper conditions. DNA is read by a process called sequencing, and the sequencing technology is decoupled from DNA, the storage medium, itself. Thus, DNA will not suffer from media obsolescence, as we will always be able to read back data stored in DNA. Finally, using common, well-established biochemical techniques, it is very easy to replicate DNA rapidly. Thus, data stored in DNA can be easily copied. Given these benefits, several researchers have demonstrated the feasibility of using DNA as a long-term archival storage medium [6, 9, 13, 19, 21, 23, 30, 41].

The biochemical processes used for writing (*synthesis*) and reading (*sequencing*) DNA are not precise and introduce several errors. In order to provide reliable data storage despite such errors, it is necessary to use redundancy in both writing (using error control coding) and reading pipelines (in the form of high sequencing coverage). The added redundancy has the undesirable side effect of amplifying the read/write cost. Thus, efficient handling of errors is crucial to reducing overall cost. Similarly, given the high density of DNA, an

archive stored in DNA can contain millions of files. However, real-world scenarios often demand access to only a fraction of the information, like a single table from a database, or a specific document from an archive. Thus, the implementation of reliable random access is crucial in making large-scale, cost-efficient DNA-based storage feasible [18, 38]. In this work, we make three contributions.

- Using data from real wet lab experiments, we perform a quantification of random-access errors in DNA storage. Prior studies have done quantification of substitution, deletion, and insertion errors introduced by synthesis and sequencing. However, there has been little focus on performing a systematic quantification of errors introduced by Polymerase Chain Reaction (PCR)–the procedure used to achieve random access in DNA storage–over a complex DNA pool storing files of various sizes. We bridge this gap by presenting such an analysis (Section 2).
- We present Columnar MOlecular Storage System (CMOSS), an end-to-end pipeline for DNA storage that provides substantially lower read/write costs than SOTA. The key aspects of CMOSS that distinguish it from SOTA are: (i) a motif-based, vertical layout in contrast to the nucleotide-based, horizontal layout used by SOTA, and (ii) an integrated consensus and decoding technique that exploits the vertical layout to incrementally recover data at low sequencing coverage, and (iii) a fixed-size, block-based random access organization for DNA storage instead of a variable-sized, object-based access used by SOTA.
- We perform several simulated and wet lab experiments to generate complex oligo pools. We use these experiments to (i) validate the CMOSS design by ensuring successful data recovery, (ii) compare CMOSS with SOTA in terms of read and write costs, and (iii) perform a systematic study of the impact of using PCR for randomly accessing fixed-size blocks in contrast to variable-sized objects. In doing so, we show that (i) the motif-based vertical layout and integrated consensus calling and decoding makes CMOSS resilient to errors caused by consensus bias, and (ii) the fixed-size, block-based random access organization of CMOSS makes it resilient to errors caused by PCR bias. We make the CMOSS pipeline [1]publicly available for further research.

2 BACKGROUND

When used as a storage medium, DNA introduces different types errors for providing different functionalities. In this section, we will provide an overview of these errors, while making a distinction between errors that are common to all DNA storage pipelines (Section 2.1), and errors specific to pipelines that support random access (Section 2.2).

[1]https://gitlab.eurecom.fr/marinele/oligoarchive-columnar.git

2.1 Errors due to Consensus Calling

In all SOTA pipelines, binary data is stored in DNA by transforming it into a quaternary sequence of nucleotides (nts) (Adenine, Guanine, Cytosine, Thymine) using an encoder. Subsequently, these sequences are utilized in the fabrication of DNA molecules, commonly referred to as oligonucleotides (or "oligos"), through a chemical process known as synthesis. The retrieval of data stored in DNA is accomplished by first sequencing the oligos to produce reads, which are sequences that correspond to the nt composition of oligos. Both synthesis and sequencing are not precise, and they introduce several types of errors: (i) multiple copies of each oligos are synthesized and library preparation before sequencing often amplifies oligos resulting in duplication, (ii) substitution errors can cause some nts to be substituted with others, (iii) insertion and deletion errors can cause some spurious nts to be inserted or existing nts to be deleted. The rate of each error type varying depends on the synthesis (column vs array, enzymatic vs phosphoramidite) and sequencing (short read vs long read) technologies used. As the outcome of sequencing, we receive *reads*, which are noisy replicas of the original encoded sequences. Thus, in all SOTA pipelines, the first step in the decoding process is clustering so that duplicate reads belonging to the same sequence are groped together. Several clustering algorithms have been proposed for this purpose [2, 3, 17, 20, 26, 27, 33, 34, 40, 43, 55].

After clustering, each cluster of noisy reads is processed independently in order to perform consensus calling to determine the most probable original sequence. Several solutions have been proposed in literature for this trace reconstruction problem [4, 22, 28, 29, 32, 44, 45, 49]. Figure 1 shows an example of such a reconstruction algorithm applied to a cluster of three strings. When the noisy reads contain mostly substitution errors, and coverage (number of reads that correspond to an oligo) is sufficient, we can infer the correct nt at each position through majority voting. For instance, in the example, we can assume that the first nt is A, as both the first and third strings have an A as their first nt. This same procedure applies to the rest of the column (nts).

Handling cases with insertions or deletions is more complex. In Figure 1(b)-(d) we have the same three strings but with insertion and deletion error. When we apply consensus to the first character, we can assume that the first nt inferred is an A as there are no errors (Figure 1(b)). However, at the second position (Figure 1(b)), we see that the three strings differ, as the first and third string have a G, while the second string has a T. At this point we have to make an assumption. One possibility is to assume that the T was an insertion error in the second string. We can correct the insertion by deleting T in the second string which will result in G being identified as the consensus output. However, we could have

Figure 1: Example of consensus algorithm applied to a cluster of three strings in case of substitution errors only (a) and insertion/deletion errors (b)-(d).

also assumed something different, for example a substitution error in the second string, where instead of G we have T. This would lead to a different consensus string.

Thus, every attempt to correct an error in our strings is based on an assumption, which means that there is a possibility of misinterpreting the error type. With such misinterpretation, an error can propagate through the read, as one wrong insertion or deletion call can result in a total mismatch in the remainder of two strings. Lin et al. [30] called this the reliability bias and showed that this is an intrinsic property of the trace reconstruction problem when insertion and deletion errors are present irrespective of the consensus algorithm used.

The reliability bias has significant repercussions for DNA data storage because the probability of errors in a read increases as we move further along the read, and is directly related to the length of oligos. Thus, as new synthesis and sequencing techniques improve and enable the creation and readout of longer oligos, reliability bias becomes more pronounced as errors early on in a read can propagate through its length. This, in turn, necessitates higher sequencing coverage to enable consensus algorithms to successfully infer the original sequences. This increased coverage directly translates to higher sequencing costs. Thus, we need techniques that can allow us to limit the error propagation caused by reliability bias.

2.2 Errors due to Random Access

Having described the reliability bias caused by consensus that affects all DNA storage pipelines, we now focus specifically on pipelines that support random access. A single DNA pool is capable of storing several Petabytes to Exabytes of data. However, it is often necessary to retrieve only a small amount of data. Prior work has achieved this by assuming an object-based get/put interface to DNA storage and relying on the use of PCR for achieving random access of individual objects[38, 51]. The central idea is to associate a distinct pair of short DNA sequences, also called primers, to all oligos belonging to each distinct object. Random access is performed by using PCR to selectively amplify the DNA containing the target primers corresponding to the object that is requested.

Table 1: The number of oligos and corresponding database and table primers in Exp. 1.

Table#	Table primer	Database name/primer		
		SSB/CAATG	TPCH/GATGA	SYN/GTGAG
1	TTAAG	14	6	304
2	GAATT	16	18	312
3	AAGGT	42	18	302
4	ACAGA	2594	10	302
5	AGAGA	34	20	298
6	CAGTT		14	300
7	CATAC		34	298
8	CGATA		16	306

Prior studies have quantified the nature and frequency of substitution, insertion and deletion (indel) errors introduced by different sequencing technologies and used such quantification to configure the amount of redundancy introduced during encoding/decoding[24]. Studies have also looked at oligo drop outs caused by coverage bias[11]. Coverage bias refers to the fact that after sequencing, original sequences are covered at very different rates, with some sequences being covered by multiple sequenced reads and others completely missing. Coverage bias is a well-known issue in DNA storage, with duplication during both synthesis, and library-preparation for PCR, contributing to it. The issue with uneven coverage distribution is the fact that consensus calling might not be able to successfully infer sequences with inadequate coverage due to reliability bias described earlier, leading to data loss. Thus, prior work has modeled coverage bias for specific synthesis and sequencing technologies and proposed methods to optimize redundancy levels to guarantee full recovery.

However, there has been limited work on systematically quantifying coverage bias introduced by PCR-based random access [51] in complex oligo pools containing objects of varying sizes. In order to study this, we conducted a wet lab experiment (**Exp. 1**) where we stored three databases: SSB, TPCH, and SYN, comprising five, eight, and eight tables, respectively. The SSB and TPC-H databases were chosen from the industry-standard TPC-H benchmark, and they represent a size distribution typical in a data warehousing application. The SYN database contains randomly generated records and was configured to have table with fixed sizes. Our intention in using these databases was to isolate and study the sensitivity of PCR to the complexity of the oligo pool created by varying file sizes.

The databases were converted using Goldman et al's[21] rotational encoding approach to generate 5258 sequences of 110nt each. Each sequence consists of a data payload in the middle, flanked by the database primer (DBP), table primer (TBLP), universal forward primer (UFP), and universal reverse primer (URP) on the sides (Figure 2). The UFP and

Figure 2: The oligo structure for object based abstraction. UFP/URP: universal forward/reverse primer, DBP: database primer, TBLP: table primer.

URP are identical across all oligos allowing the selection of the full set (three databases). Table 1 shows the table and database primers used and the number of oligos generated for each table. We utilized primers derived from Illumina adapter sequences, as they inherently adhere to the necessary biological constraints of DNA synthesis and sequencing. The sequences were synthesized by Twist Bioscience and the oligos were sequenced in all experiments described below using Illumina NovaSeq. Error analysis confirmed a high-quality synthesis and sequencing with a low error rate inline with SOTA work on DNA storage (0.0033 for substitutions, 0.0003 for indels), enabling us to reliably quantify coverage bias in the complex pool.

Quantitative definition of coverage bias. We use the term "population fraction" [11] (popfrac) to refer to the proportion of the data that belongs to a specific object (for example, a database or a table) within the entire archival dataset. Given n objects, the popfrac of object i, denoted as p_i, is computed as $p_i = N_i / \sum_{j=1}^{n} N_j$, where N_i represents the number of sequences belonging to the specific object i. We compute two popfracs. First, we compute a popfrac using the number of encoded sequences belonging to each table or database. We refer to this as the *raw popfrac*, donated as p_i^r for object i. Second, we take all the reads obtained from sequencing the synthesized oligos, and align them to the encoded sequences to determine read coverage (number of reads that correspond to each sequence) using Accel-Align [52, 53]. We aggregate per-sequence coverage to compute per-table and per-database coverage. Finally, we use this to compute *post-sequencing popfrac* as the proportion of sequenced reads corresponding to each table/database. To distinguish this from p_i^r, we refer to it as p_i^s for object i. The ratio of the post-sequencing popfrac to the raw popfrac is defined as *popfrac change*. Formally, the popfrac change of object i, denoted as c_i, is computed as $c_i = p_i^s / p_i^r$. In the absence of coverage bias, we expect this ratio to be one. For instance, if 40% of the oligos belong to table i in a database, then after PCR and sequencing, we expect 40% of the reads will still belong to table i, meaning they can be aligned to the oligo of table i. If popfrac change is too high, it indicates that some objects are over represented, and if it is too low, that some objects are under represented.

Coverage bias observation in real wet lab experiment. To investigate coverage bias when the whole dataset is retrieved, we used the UFP (Figure 2) during PCR amplification to sequence all oligos. Using the reads, we determined the post-sequence popfrac for each database as shown in Table 2. Clearly, coverage is uneven and biased, as some databases are over represented, like SYN, while others become under represented, like TPCH, after sequencing.

Table 2: For each database in Exp. 1, the table lists the number of oligos, raw population fraction, number of reads, post-sequencing population fraction, and fraction change.

Database	#Oligos	Raw Pop Frac	#Reads	Pop Frac	Frac Change
SSB	2700	0.514	654335	0.388	**0.76**
TPCH	136	0.026	19576	0.012	**0.45**
SYN	2422	0.461	1013152	0.601	**1.30**

Next, to study coverage bias under random access, we employed the UFP in conjunction with a DB primer (Figure 2)) during PCR to amplify only oligos belonging to one particular database and sequenced the amplified pool. We used the reads to determine post-sequenced popfrac for each table across the three databases as shown in Figure 3.

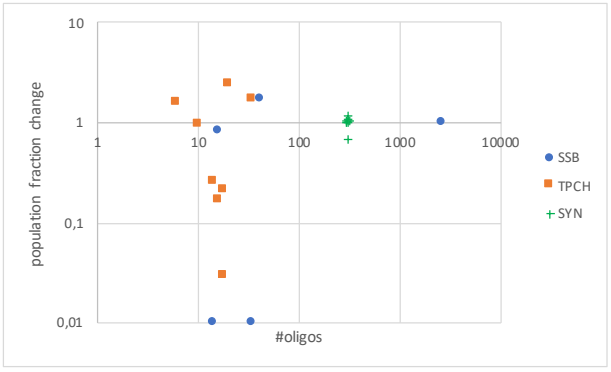

Figure 3: The population fraction change (y-axis) and the number of oligos (x-axis) of each table in Exp. 1.

As can be seen, databases with tables of varying sizes exhibit a huge variation in popfrac change. For instance, for the SSB database, tables vary in size from 14 oligos to 2594 oligos. The average popfrac change is 0.71 (ideally 1) with a stdev of 0.73 (ideally 0); some smaller tables are significantly under represented (min. popfrac change of 0.01), while some large tables are over represented (max. popfrac change of 1.72). For the TPCH database, it is similar, with a min. popfrac change of 0.03, max. of 2.4, average of 0.92, and stdev of 0.94. In contrast to these two databases, the SYN database with its uniform table sizes exhibits a uniform distribution of popfrac change (shown as '+' in Figure 3 with overlapping points), with an average of 1, and a stdev of 0.14.

These observations expose a natural limitation of naively storing objects on DNA and using PCR-based random access without considering their sizes. Coverage bias can result in drastically different popfrac change for objects of different sizes, and smaller objects can get significantly under represented. Such under representation can in turn affect the effectiveness of random access by making smaller objects difficult to retrieve leading to data loss. In contrast, the SYN database tables show that using uniformly-sized units of storage can substantially minimize this bias and ensure a more uniform representation of oligos.

3 DESIGN

Having described the reliability and coverage bias issues of DNA storage, we now present CMOSS. Our approach to archiving data in CMOSS differs from SOTA based on the key observation that the separation of consensus and decoding is a direct side-effect of the data layout, that is the way oligos are encoded. Mapping a coded block of data to a group of oligos results in that group becoming a unit of recovery. Thus, before data can be decoded, the entire group of oligos must be reassembled by consensus, albeit with errors. The two key ideas in our system are to (i) use short sequences we refer to as motifs as building blocks of oligos instead of individual nucleotides, and (ii) change the layout from the horizontal, row-style SOTA layout (Figure 4(c)) to a vertical, column-style cross-oligo layout (Figure 4(d)). Our DNA storage system encodes and decodes data as a collection of motifs vertically across several oligos instead of horizontally. The key benefits of these two changes, as we show later in this section, are the fact that (i) we can merge decoding and consensus into a single step, where the error-correction provided by decoding is used to improve consensus accuracy, and the improved accuracy in turn reduces the burden on decoding, thereby providing a synergistic effect, and (ii) it naturally leads to a random access organization where each unit of random access is a fixed-size extent instead of a variable-sized object, thereby benefiting automatically from a more balanced coverage across extents. In the rest of this section, we will explain the design of our system and its advantages by presenting its read and write pipelines.

3.1 Write Pipeline

The top half of the Figure 5 shows the data writing pipeline of CMOSS. The input to the write pipeline is a stream of bits. Thus, any binary file can be stored using this pipeline. The first step in processing the input involves grouping bits into chunks. The chunk size depends on the error-correcting code adopted, as we will see later in this section. Each chunk of input is then pseudo-randomized in order to maximize the separation between the encoding oligos. This approach enhances the accuracy of read clustering in the data decoding stage, as detailed in Section 3.2. After randomization, error correction encoding is applied to each chunk to protect the data against errors. Due to this encoding, each chunk functions as a unit of error control, becoming the smallest recovery unit in CMOSS. The choice of code is orthogonal to the design of CMOSS, and any large-block length code can be used to add redundancy. In our system, we support both Reed-Solomon (RS) and Low-Density Parity Check (LDPC) codes. We parameterize the RS code with the same block length and symbol size used by Organick et al. [38](65,536 symbols with 16 bits per symbol) in the work on random access in DNA storage. We parameterize the LDPC code with a chunk size of 256,000 bits, similar to prior work by Chandak et. al. [9], which has demonstrated that such a large-block-length LDPC code is resilient to both substitution/indel errors, that cause reads to be noisy copies of original oligos, and synthesis/sequencing-bias-induced dropout errors, where entire oligos can be missing in reads due to lack of coverage [9, 36]. For the rest of this section, we will focus on using the LDPC code to discuss the rest of the stages. In Section 4.4, we present an evaluation of CMOSS with both LDPC and RS codes.

The LDPC encoded bit sequence is fed into the *oligo-encoder*, which converts bits into oligos. While SOTA approaches design each oligo as a random collection of nts, our oligo-encoder designs oligos using composable building blocks called *motifs*. Each motif is a short oligo that obeys all the biological constraints enforced by synthesis and sequencing. Multiple motifs are grouped together to form a single oligo. We use motifs rather than single nts as building blocks because, as discussed later in Section 3.2, the integration of decoding and consensus relies on alignment which cannot be done over single nt. To convert bits into motifs, the oligo-encoder maintains an associative array with a 30-bit integer key and a 16 nucleotide-length motif value. This array is built by enumerating all possible motifs of length 16nt (AAA, AAT, AAC, AAG, AGA...) and eliminating motifs that fail to meet a given set of biological constraints. We configure our encoder to admit motifs that have up to two homopolymer repeats (AA,CC,GG, or TT), and GC content in the range of 0.25 to 0.75 [35, 37]. We also experimented with avoiding secondary structure formation at the motif level. These help reduce errors during synthesis and sequencing, and enhances the stability of the synthesized DNA sequences [38]. Thanks to the randomization of input, the motifs selected in each oligo are randomized as well. So we do not have secondary structures issues at the oligo level as well. With these constraints, using 16nt motifs, out of 4^{16} possible motifs, we obtain 1,405,798,178 that are valid. By mapping each motif to an integer in the range of 0 to $2^{30} - 1$, we can encode 30-bits of data per motif. Thus, at the motif level, the

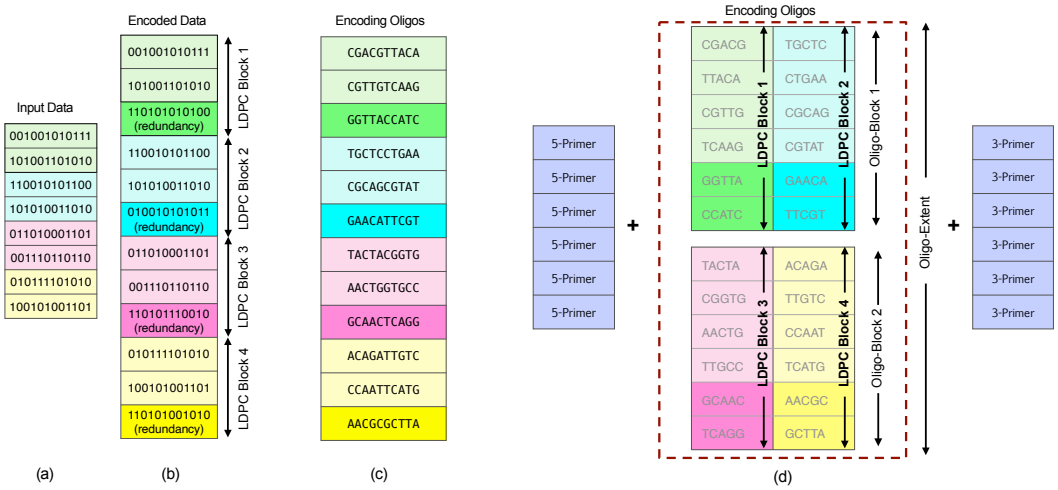

Figure 4: Figure shows the raw input data being grouped into blocks (a), and each block being encoded to generate parity (b). (a) and (b) are common to both SOTA and CMOSS. (c) shows each encoded block being mapped to multiple oligos with SOTA. (d) shows each block being mapped to one column of motifs with CMOSS. Multiple such columns of motifs form one Oligo-Block. Multiple Oligo-Blocks are grouped into one Oligo-Extent. Every Oligo-Extent is extended with a pair of primers, that makes it addressable.

encoding density is 1.875 bits/nt. Although it is possible to increase this density by increasing the motif size, we have limited ourselves to this configuration for two reasons: (i) the memory limitation of our hardware, as the current associative array itself occupies 100GB of memory, (ii) the motif design is orthogonal to the vertical encoding, which is the focus of this work.

While Figure 4(d) shows all columns of motifs storing only the LDPC blocks, a small subtlety in the practical implementation is that the first column in every oligo-block is dedicated to storing indexing information. This indexing information orders oligos during encoding and hence enables reordering during decoding.

The second major difference of our approach compared to SOTA is the layout of motifs, which spread vertically in columns across a set of oligos. The motifs generated from an error-control coded data block are used to extend oligos by adding a new column as illustrated in Figure 4(d). This process is repeated until the oligos reach a configurable number of columns, after which the process is reset to generate the next batch of oligos starting again from the first column. We refer to such a batch of oligos as a *oligo-block (OB)*. OB represents the minimum granularity of decoding in our CMOSS pipeline, where all columns (i.e. LDPC blocks) belonging to an OB must be encoded before starting a new OB. Similarly, all columns (i.e. LDPC blocks) belonging to an OB must be decoded in order from left to right to guarantee successful data recovery as detailed in Section 3.2.

To scale to large datasets, we designed CMOSS as a flexible, hierarchical DNA storage system. For this purpose, we group one or more OBs into a higher-level structure, termed as *oligo-extent (OE)*. While an OB serves as the unit of encoding and decoding in CMOSS, an OE functions as the unit of random access. Each OE is designed to be a self-contained, fixed-size (configurable during encoding) and addressable DNA storage partition. Thus, each OE becomes randomly addressable by adding an unique pair of primers at two extremities of every oligo in all OBs within that OE. All oligos within the OE are addressed using the indexing strategy as described earlier.

This hierarchical storage approach offers several advantages. First, it facilitates the storage of exceptionally large files while maintaining a relatively low primer count, as the number of primer pairs required is proportional to the number of OEs, which can be minimized by grouping more OBs into a single OE. Second, it allows flexibility in choosing random access granularity during design time, ranging from one OB to multiple OBs. Third, using OE as the granularity of random access is equivalent to partitioning the DNA into fixed-sized storage units. As described in Section 2.2, this fixed-sized, extent-based random access approach mitigates the impact of PCR coverage bias.

3.2 Read Pipeline

Data stored in DNA is read back by sequencing the DNA to produce reads, which are noisy copies of the original oligos that can contain insertions, deletions and substitutions.

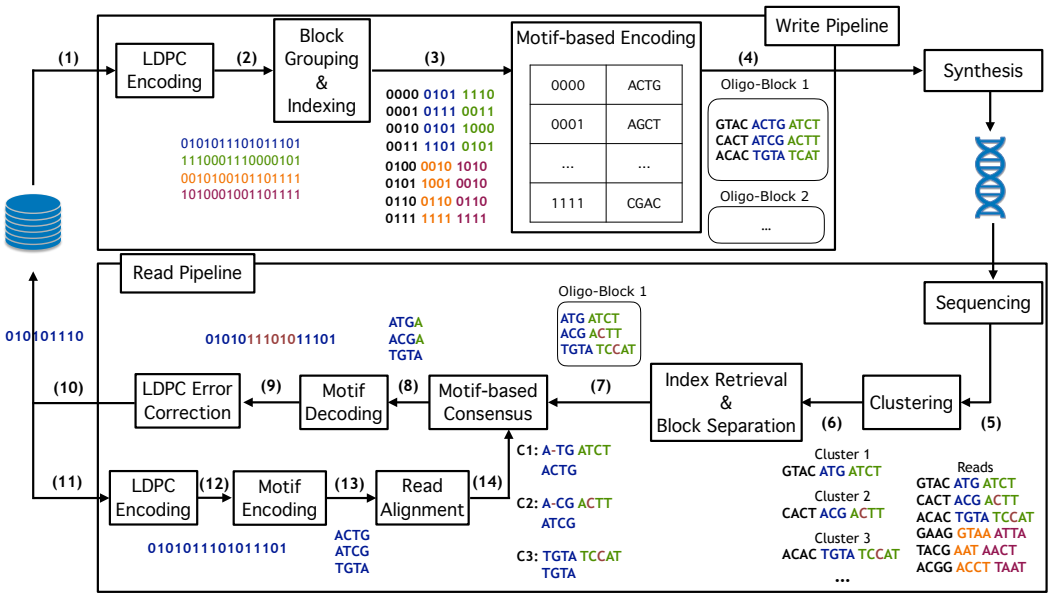

Figure 5: CMOSS data writing pipeline (top) showing various steps in the encoding path using an example where input data is LDPC encoded (2), indexed and split into 4-bit chunks (3), and mapped into motifs to produce oligos in a vertical fashion (4). CMOSS reading pipeline (bottom) showing DNA to binary decoding path using an example that assumes 1× coverage (1 read per oligo) from sequencing (5), applies clustering to separate reads (6) and indexing to group reads per oligo-block (7). Then, each oligo-block is decoded separately in a vertical fashion by performing motif-based consensus to get a column of motifs (8), motif decoding to map motifs to bits (9), and LDPC decoding for error correction (10) to recover input bits. The corrected bits are reencoded (11, 12) to generate the correct column of motifs, which is then used to fix errors in reads via alignment (14). The whole process starting from motif consensus (8-14) is repeated for the next column of motifs (green nts) until all columns in one oligo-block are processed.

Due to the hierarchical structure, decoding begins by first grouping the reads based on their OE. To achieve this, the reads are aligned at both ends to unique primer pairs that designate each OE. Since data within different extents are independent, decoding can proceed concurrently across multiple OEs, significantly speeding up the operation. For sake of simplicity, in the rest of this section we focus on decoding of a single OE with a single OB.

Recall that an OB consists of multiple oligos organized as several columns of motifs. Since each oligo can be covered by multiple reads, the first step in decoding is clustering to group related reads together. We utilize a string clustering solution that uses randomized embedding similar to the one developed in prior work [42]. It should be noted that any other read clustering solution can be applied and is independent of the work presented here. The output of this algorithm is a set of clusters of reads, each corresponding to an unknown original sequence.

After the clustering stage, other SOTA methods typically apply consensus methods within each cluster to infer consensus sequences from reads. This is then followed by decoding using the consensus sequences. During decoding, SOTA methods use error-correction codes to recover from any residual errors that might be present after consensus. Thus, decoding ultimately produces the original input bits. It is important to note that SOTA methods do not use the decoded bits from one error-control block to improve the decoding of further downstream encoded blocks. To illustrate this with an example, let us consider the first three sequences (in green) in Figure 4(c). These sequences encode the LDPC-block-1 depicted in Figure 4(b). In order to decode and correct this block of bits, SOTA approaches first perform consensus calling to infer the first three full sequences; then, they can convert the inferred sequences into encoded bits, and finally perform decoding with error-control codes to recover back the original input bits. However, once the original bits for the green block are retrieved, they are only used as part of the final output, which is the reconstructed original input file.

In CMOSS, we leverage the motif design and vertical layout of oligos to integrate consensus and decoding progressively, as shown in the lower part of Figure 5. Unlike other approaches, our system processes reads column by column. Thus, the first step is motif-based consensus where the set of reads is processed to generate the first column of motifs. The choice of consensus algorithm is orthogonal to CMOSS design. We employ an alignment-based algorithm for motif-based consensus calling previously proposed in literature [5]. This algorithm extracts a motif-length segment from each read within a cluster, aligns them, and derives a consensus motif by a position-wise majority voting. As each cluster corresponds to an oligo, this process is done for each cluster to determine one consensus motif per cluster, and hence all consensus motifs for the first column of motifs. These motifs are then fed to our *CMOSS oligo-decoder*, which is the inverse of the encoder, as it maps the motifs into their 30-bit values. Notably, despite consensus, inferred motifs can still contain errors leading to incorrect 30-bit values. These errors are rectified by the *LDPC-decoder*, which takes the 30-bit values corresponding to one LDPC block as input and outputs the error-corrected, randomized input bits. Subsequently, these bits undergo derandomization to retrieve the original input bits for that block.

Different from SOTA, in CMOSS, the decoded bits are reencoded again by passing them through the LDPC-encoder and the oligo-encoder. This process reconstructs the correct first column of motifs, similar to the encoding process in the write pipeline. Subsequently, these correct motifs are utilized to realign reads within each cluster, ensuring that the subsequent round of decoding for the second column start at the correct offset. This entire process is iteratively repeated for all subsequent columns.

The intuition behind realignment is as follows. An insertion or deletion error in a consensus motif affects not only that motif but also all subsequent motifs due to variations in length. For instance, consider the sequenced reads in step (6) of Figure 5, we see a deletion error in the first read $GTACA-TGATCT$ which should have been $GTACA\mathbf{C}TGATCT$ (according to the oligos generated in step (4) of the same figure). This results in the second motif being incorrectly interpreted as $ATGA$ instead of $ACTG$. Left uncorrected, this error will spill over to the third motif which will be read as $TCT...$ (instead of $ATCT$). Thus, with SOTA approaches, an error early in a consensus oligo keeps propagating leading to the reliability bias as explained in Section 2.1. On the contrary, in CMOSS, every column stores a full LDPC block, and we decode one column at a time. Thus, we can use the decoded bits from one column to regenerate the correct motifs by reencoding them during decoding. We can use the correct motifs to fix these errors by realigning the correct motifs against reads. This realignment will determine the position where current motif ends and the next motif begins, and hence, determine the starting point for the next column. As a result, any consensus errors in one column can be fixed by realignment and do not propagate downstream limiting the impact of positional bias.

Notably, this realignment is only possible because we use motifs, as two sequences can be aligned accurately only if they are long enough to identify similar subsequences. Thus, vertical layout without motifs, or with just nts, would not make realignment possible. Similarly, integrating consensus and decoding is possible only because of the vertical layout, as the SOTA layout that spreads a LDPC block across several oligos cannot provide incremental reconstruction.

Finally, a refinement to the decoding procedure we have described so far is the special handling about indexing information, particularly when an OE contains multiple OBs. Recall that OE serves as the unit of random access, while OB is the unit of decoding. And the indexing information is stored in the first column of motifs across the entire OE. Therefore, in cases where an OE comprises multiple OBs, primers are used to identify OE, while this index is used to indirectly identify the OB of each oligo within that OE. Decoding the first column of motifs is distinctive because it generates the indexing information across all OBs within an OE. This index information is crucial for separating reads into constituent OBs. And from the second column, we switch to per-OB processing. The whole process is illustrated with a simple example in Figure 5.

4 EVALUATION

In this section we present a thorough evaluation of CMOSS. First, we present the results from two wet lab experiments to study the ability to achieve full data recovery (Sec. 4.1 and Sec. 4.2). Next, we compare CMOSS with various SOTA approaches with respect to read cost and write cost to show that our design can lead to substantial cost reduction (Sec. 4.3). Following this, we isolate and analyze the advantage of using a vertical design by comparing CMOSS with a horizontal pipeline (Sec. 4.4).We conduct all the experiments on a local server equipped with a 12-core CPU Intel(R) Core(TM) i9-10920X clocked at 3.50GHz, 128GB of RAM. The core components shown in Figure 5 have been implemented in C++17.

4.1 Small-Scale Wet-Lab Validation

As the first prototype test, we used the TPC-H DBGEN utility to generate a compressed 1.2MB database which was then encoded by CMOSS, configured with 30% LDPC redundancy, into 44376 oligos of length 200nt partitioned into 16 OEs. The oligos were synthesized by Twist Biosciences and subsequently sequenced using Oxford Nanopore PromethION

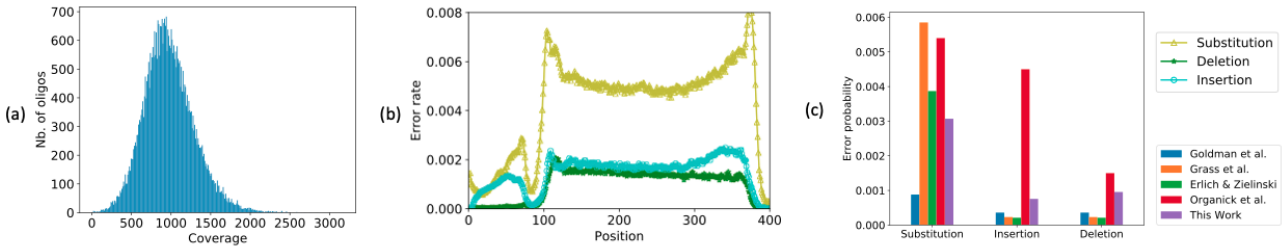

Figure 6: Statistics for Exp. 2. a) The histogram of coverage across oligos (x-axis: sequence coverage, y-axis: number of oligos with that coverage). b) Error rates per read position. c) Comparison of the error rates with previous work.

platform with Ligation Sequencing Kit V14 (SQK-LSK114), producing a total of 43M reads (**Exp. 2**).

Error Pattern. To perform error characterization, we aligned the sequenced reads generated from **Exp. 2** to the original oligos using Accel-Align [52] sequence aligner. 99.9999% reads were aligned to a reference oligo, indicating a very high quality of the generated read set. Figure 6.a shows the coverage histogram and it can be observed that each reference oligo is covered by at least one read, with a median coverage of 951×, minimum coverage of 5×, and a maximum coverage of 2500×. We deliberately sequenced the oligos at such high coverage to test recovery at various coverage levels as we present later.

The average error rates are 0.003 for substitutions, 0.0008 for deletions, and 0.001 for insertions computed by BBmap [8]. The error rate per position is illustrated in Figure 6.b. Note that while the data-carrying payload had a length of 160, our reads are longer as they include the primers that were appended at both ends of the oligo for sequencing and other sequencing adapters that were used for multiplexed sequencing. As these primers and adapters get trimmed out during read preprocessing, the error rate of relevance to us is the middle portion of the read which corresponds to the encoded, data-carrying portion of the oligo. We see that in this portion, the substitution rate is dominant, which is 3× higher than insertion and deletion rates. Figure 6.c compares our error rates with those reported in prior work on DNA storage [19, 21, 23, 24, 38]. We calculated these statistics using raw reads without any quality-based filtering. As can be seen, our substitution error rate is lower than Grass et al., Erlich et Zielinski. and Organick et al., while the inserion and deletion rate are slight higher due to the use of array synthesis and Nanopore sequencing, although they have improved in accuracy over the past few years, with Nanopore PromethION platform offering single-read accuracy of over 99% with LSK114 kit[50]. It is hard to attribute a precise fraction of improvement in error rate to synthesis and sequencing without an isolated comparison of each with other studies. However, we can see that the overall trends of relative errors are similar.

Data Recovery. In order to test end-to-end decoding, we first used the full 43M read dataset generate from **Exp. 2** as input to the decoding pipeline. We were able to achieve full data reconstruction, given the ability of CMOSS to handle much lower coverage levels and higher error rates. In order to stress test our decoding pipeline and identify the minimum coverage that allows fully reconstruction of data, we repeated the decoding experiment on smaller reads sets which were derived by randomly sampling a fraction of reads from the 43M read dataset. In doing so, we found that CMOSS was able to perform full recovery using just 200K reads, which corresponds to a coverage of 4×. At this coverage, nearly 3500 out of 44376 reference oligos were completely missing. However, the LDPC code and vertical decoding were able to successfully recover data. As further reduction in coverage led to data loss, we validate 4× as the minimum coverage CMOSS can handle with our wet lab experiment. Computing the costs for minimum coverage, we get a read cost of 2.82 nts/bit, and a write cost of 0.70 nts/bit.

4.2 Large-Scale Wet-lab Validation

As a large-scale test of random access, we stored a 13MB tar archive containing culturally significant documents, including images, PDF files, and text documents, sourced from a national archive (**Exp. 3**). Employing a methodology similar to that of **Exp. 2**, we encoded the input with just 10% LDPC redundancy, lower than the one adopted in **Exp. 2**. The reason of a lower redundancy is that in **Exp. 2** we already proved a full reconstruction with a very low coverage at 30% redundancy. Thus, for this experiment we tested our system with a lower redundancy overhead. This resulted in a total of 262,836 sequences of 240nt stored in 14 OB. For the purpose of this experiment, given the limited number of extents, we made the number of OE and OB identical. This means that with each OB and hence, each OE, store 468KB of information (15 256000 bit LDPC blocks per OB). This becomes the unit of random access. To identify each OE/OB individually, we added a 20nt 5'-primer and a 20nt 3'-primer to each sequence (for a total of 280nt oer oligo)

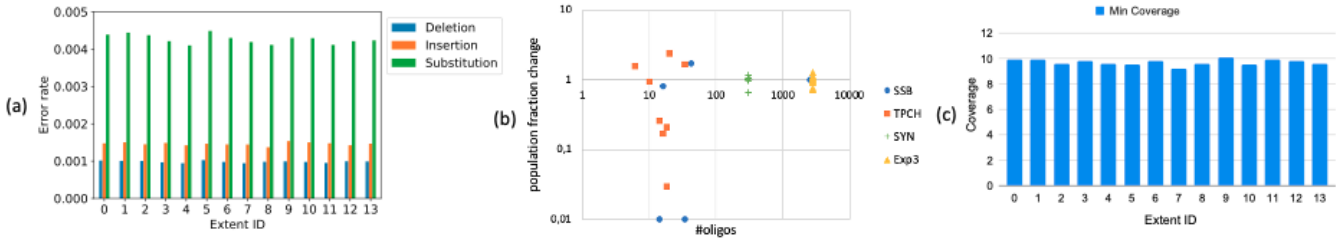

Figure 7: Statistics for Exp. 3. a) The error rate of the reads per extent. b) The population fraction change (y-axis) and #oligos (x-axis) per extent. The yellow points are overlapping as Exp. 3 has 14 extents of uniform size and their population fraction changes are all close to 1. Metrics from Exp. 1 (Figure 3) are also included for comparison. c) The min. coverage required for full data recovery per extent.

Table 3: Write cost $(\#nts - in - oligos)/(\#bits)$ and read cost $(\#nts - in - reads)/(\#bits)$ of CMOSS vs SOTA.

Reference	Binary Size	#Oligos	Oligo Length (nt)	Recovery Coverage	Write Cost (nt/bit)	Read Cost (nt/bit)
Church et al. [14] (2012)	658 KB	54,898	115	3000	1.17	3513.66
Goldman et al. [21] (2013)	650 KB	153,335	117	51	3.37	171.83
Grass et al. [23] (2015)	85 KB	4,991	117	372	0.84	311.97
Bornholt et al. [7] (2016)	150 KB	16,994	80	40	1.11	44.26
Yazdi et al. [54] (2017)	3.55 KB	17	1000	200	0.58	116.91
Erlich and Zielinski [19] (2017)	2.11 MB	72,000	152	10.4	0.62	6.43
Organick et al. [38] (2018)	200 MB	13,448,372	110/114	5	0.91	4.57
Anavy et al. [1] (2019)	22.5 B	1	42	100	0.23	23.33
Choi et al. [12] (2019)	135.4 KB	4,503	111	250	0.45	112.66
S. Chandak et al. [10] (2019)	192 KB	11,892	n.a.	5	0.78	4.46
THIS WORK (Exp2)	1.2 MB	44,376	160	4	0.70	2.82
THIS WORK (Exp3)	13 MB	262,836	240	9.5	0.57	5.49

which were synthesized with Twist Biosciences. To evaluate data recovery per extent, we conducted 14 independent wet lab experiments. Each wet lab used one extent's distinct left and right primers during PCR amplification to randomly select that extent. Subsequently, the amplified oligos were sequenced using the same Oxford Nanopore PromethION platform to produce 6.1M reads.

Error Pattern. Figure 7.a shows the average substitution, deletion, and insertion error rates of reads per extent. As can be seen, the rates are similar across extents and comparable to the results of **Exp. 2** shown in Figure 6.b.

Coverage Bias. As we explained in Section 2.2, file-based random access suffered from a high coverage bias when files are of varying sizes. To investigate bias under block-based random access with CMOSS, we aligned all 6.1M sequenced reads from **Exp. 3** to their original oligos using Accel-Align in order to determine their original extents. We used this alignment to calculate population fraction change. Figure 7.b is an extension of Figure 3 with the points for each of the fourteen extents from **Exp. 3** added. As each extent has the same number of oligos, all points cluster together on the x-axis. Due to the uniform extent size, the population fraction change across all extents is close to 1, with a standard deviation of 0.278. This result is in clear contrast to TPCH and SSB database results, where population fraction change varies a lot with standard deviations of 0.71 and 0.94. The low standard deviation in **Exp. 3** case with CMOSS signifies that the oligos now have a more uniform coverage across extents after the PCR process due to the fact each unit of random access has an identical number of oligos, just like the simulated SYN database with uniform table sizes.

Data Recover per Extent. We utilized all available reads for each extent to independently reconstruct the data blocks stored within them using the CMOSS read pipeline. The average coverage of each extent is 30x, with a minimum coverage of 17x and a maximum coverage of 42x. We compared the decoded bits with corresponding segments of the original

binary file to confirm that every segment of the file was accurately reconstructed.

To evaluate the robustness of our system, we conducted an experiment to determine the minimal read coverage required at the extent level for complete data recovery. This was achieved by progressively reducing the number of reads sampled from available pool of reads of each extent until the lowest count necessary for full extent recovery was identified. Utilizing this dataset, we calculated the minimum coverage per extent. The findings, illustrated in Figure 7.c, reveal the minimum coverage necessary to achieve full recovery for each extent. From our analysis, we can see that the minimum coverage across all extents is around 9.5x, and this is consistent for each individual extent, demonstrating the uniformity and reliability of CMOSS.

4.3 SOTA Comparison

Having discussed the results from our real-world wet lab experiments, we will now present a comparison of CMOSS with SOTA approaches in terms of reading and writing cost [9, 30, 38]. Writing cost is defined as $\frac{\#nts-in-oligos}{\#bits}$, where the numerator is the product of the number of oligos and the oligo length, and the denominator is the input data size. Thus, higher the redundancy and encoding overhead, higher the write cost. The reading cost is defined by $\frac{\#nts-in-reads}{\#bits}$. The numerator is the sum total of all read lengths, and denominator is the input size. Thus, higher the coverage required, higher the read cost.

Table 3 shows the read and write cost for CMOSS and other SOTA algorithms. We would like to emphasize here that our goal in reporting these results is not to directly compare our work with SOTA based on these metrics; an apples-to-apples comparison is not possible given differences in all stages of the DNA storage pipeline. Rather, our goal is to position our results in the broader context. For CMOSS, we compute these costs based on **Exp. 2** and **Exp. 3**. We only include these two results as they are from real wet lab experiments and not simulation studies. For **Exp. 2**, we compute the write cost using the 44376 oligos synthesized to encode a 1.2MB archive and for read cost the minimum number of reads (corresponding to a coverage 4x) needed to fully reconstruct the original data. Similarly, for **Exp. 3** we computed the write cost by considering the 262,836 oligos used to encode the 13MB archive while the read cost was based by considering the minimum coverage that allows us to fully recover the entire archive. We do not report data for **Exp.1** in Table 3, as it was used to demonstrate coverage bias and did not use CMOSS to encode data. For SOTA approaches, we reproduce the costs from their publications where available. There are several observations to be made.

First, let us compare CMOSS with horizontal SOTA approach that also uses LDPC (by S. Chandak et al. [9]). Both these cases use the same LDPC encoder configured with 30% redundancy. The cost reported here is for around 1% error rate in both cases. Clearly, the CMOSS approach has both a lower write and read cost. The difference in write cost can be explained due to the fact that in the horizontal LDPC approach, the authors also added additional redundancy in each oligo in the form of markers which they used in their decoder. CMOSS is able to achieve 100% data reconstruction using the same LDPC encoder at a much lower coverage level without such markers as demonstrated by the lower read cost.

Among SOTA, two other pieces of related work that have competitive read/write cost are the large-block RS coding by Organick et al. [38] and fountain codes by Erlich et al. [19]. Comparing CMOSS with these, we see that CMOSS **Exp. 2** with 30% redundancy provides better read cost than both, but worse write cost than the fountain coding approach. CMOSS **Exp. 3** has worse read cost than Organick et. al. but a better write cost than both as it uses 10% redundancy. As we mentioned earlier, we can further improve the write cost for CMOSS using several approaches. First, the CMOSS results from **Exp. 2** in Table 3 were obtained with a 30% redundancy based on its ability to handle even 12% error rate. For lower error rates (less than 1%), as was the case with the Fountain coding work, even 10% redundancy would be able to fully restore data at extremely low coverage (3× as shown in Figure 8). Second, as mentioned in Section 3, scaling the motif set by using longer motifs (17nt and 33 bits) could allow us to increase bit-level density further from 1.87 bits/nt to over 1.9 bits/nt. These two changes would lead to further reduction in write cost without any adverse effect on the read cost. As this work was predominantly about reducing the read cost, we leave these optimizations to future work.

Lin et al. [30] recently presented the Gini architecture which interleaves nts across oligos in order to minimize the impact of consensus errors. We also tried to compare CMOSS with Gini, but we could not derive the read/write cost for Gini, which was also not reported, due to lack of statistics about reads. However, as our evaluation methodology is identical to Gini, we present a direct comparison of results in terms of minimum coverage required by both approaches. Figure 10 shows the minimum coverage required by CMOSS, Gini, and a baseline without Gini reported by Lin et al. [30], to perfectly recover data at various error rates. At 18.4% redundancy based on RS coding, the reported baseline needed a coverage of 30× to recover data at 12% error rate. Gini, in contrast, provided a 33% improvement as it needed a minimum coverage of 20× at 12% error rate to guarantee full recovery. CMOSS configured at 30% redundancy with

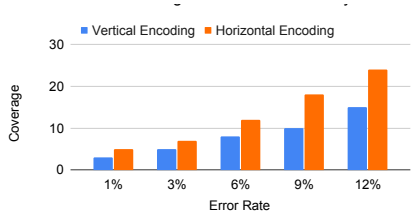

Figure 8: Min. coverage required by our vertical and horizontal implementations at 10% redundancy.

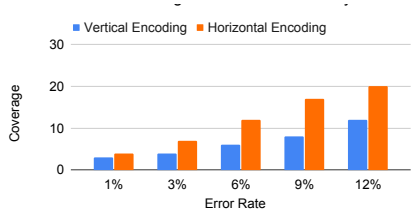

Figure 9: Min. coverage required by our vertical and horizontal implementations at 30% redundancy.

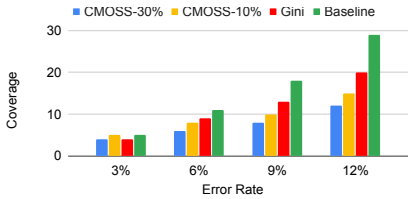

Figure 10: Min. coverage required by CMOSS, Gini and the Gini-baseline [30] at various error rates.

LDPC encoding provides a 40% improvement over Gini, it requires only 12× coverage. Comparing Figure 10 with Figure 8, we see that CMOSS provides 25% less coverage (15×) even at 10% redundancy compared to Gini. Thus, CMOSS has a much lower read cost, thanks to the integrated consensus and decoding enabled by vertical organization.

Finally, we would like to mention that there are other SOTA approaches that we tried to add to Table 3 [2, 4, 31, 39, 47, 48]. But we could not find all the information necessary for computing the read and write costs. Hence, we did not report these methods in Table 3.

4.4 Benefits of Vertical Design

In order to ensure that the benefits of CMOSS are due to the vertical design and not other parameters, we have developed a horizontal version of the pipeline shown in Figure 5, where we fixed all other parameters (clustering and consensus algorithms, LDPC block size, motif set, etcetera), and only changed two aspects to make it similar to SOTA: (i) replace CMOSS encoder with horizontal encoder that maps one LDPC block to multiple oligos, (ii) perform consensus to infer entire oligos first, and then decode separately.

In order to compare the vertical and horizontal pipelines, we perform an end-to-end DNA storage simulation study using both pipelines. First, we use both pipelines to generate the oligos for a 3MB TPC-H archive file (3MB size was chosen based on calculations that ensure that both pipelines produce the same number of oligos). We configure LDPC encoder to generate two datasets, with 10% and 30% redundancy. Then, we encode the two datasets using both pipelines, while fixing the oligo length to 50 motifs per oligo (800nt), generating four oligos datasets, two containing 18773 oligos (horizontal/vertical at 10% redundancy), and the other two containing 22187 (horizontal/vertical at 30% redundancy) oligos.

We compare the horizontal and vertical pipelines by evaluating the minimum coverage required at 10% and 30% redundancy levels to achieve 100% error-free reconstruction of the input data at various the error rates (1% to 12%). Similarly to SOTA [9, 30], for each error rate and for each of the four oligo sets, we generate read datasets at various coverage levels(1× to 25×). First, we duplicate each oligo according to the coverage levels, Then, we inject random errors (insertion, deletion and substitution with an equal probability) at random positions in each read. The number of errors injected per read follows a normal distribution with mean set to the configured error rate. We then decode the read datasets using both pipelines and identify the minimum coverage level required to fully recover the original data. We repeated each experiments three times and the minimum coverage level remained constant for every run. We reported these values in Figure 8 and Figure 9.

Figure 8 shows the minimum coverage for data encoded with 10% redundancy. Clearly, vertical encoding outperforms the horizontal one, as it reduces the coverage required up to 40% for high error rates. This reduction in minimum coverage can be intuitively explained as follows. Horizontal encoding maps an LDPC block into multiple oligos. This implies that a single erroneous oligo can lead to a data loss of up to 1500 bits (50 motifs per oligo × 30 bits per motif). As explained in Section 3, all that is required for an oligo loss is a single insertion/deletion error in the first motif after consensus. On the other hand, an oligo loss in CMOSS only causes a loss of 30 bits in each of the LDPC blocks, thanks to the vertical encoding. Further, the integrated consensus and decoding can fix consensus errors in early rounds so that they do not affect future rounds. Due to these reasons, the LDPC decoder works much more effectively when paired with vertical layout rather than horizontal encoding. The results are similar for data encoded with 30% redundancy as well, as shown in Figure 9. Notice that in the 30% case, both horizontal and CMOSS pipelines have a minimum coverage lower than the 10% redundancy case. This is expected, as a higher redundancy implies a higher tolerance to errors.

Our work is orthogonal to current efforts in designing optimal codes. Our core contributions include the vertical layout, integrated consensus, and block-based random access, all of which can be applied to any error-control codes. To demonstrate this, as we mentioned earlier, we have also

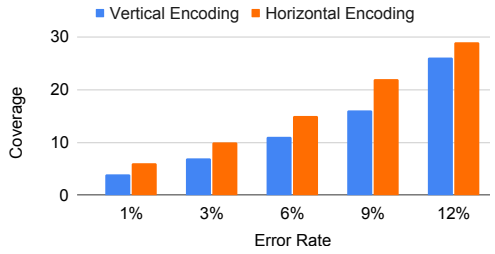

Figure 11: Min. coverage required by our vertical and horizontal encoders with RS code at 10% redundancy.

implemented RS code with the same block length and symbol size as used in Organick et al. [38](65,536 symbols with 16 bits per symbol) in both vertical and horizontal encoding implementations of CMOSS. For this experiment, we encoded the same 3MB TPC-H archive file, using the same block length as Organick et al.[38], while maintaining an oligo length of 800nt. As a result, we generated 34,951 encoding oligos with the redundancy for RS code configured to 10%. We used our simulator to vary the error rate between 1% and 12% similar to the LDPC experiment. For each error rate, and for each of the two layouts (vertical/horizontal), we generated read datasets at various coverage levels (from 1x to 25x). As shown in Figure 11, the trend of minimum coverage for various error rates is similar to the experiment conducted using LDPC; vertical encoding with its integrated consensus outperforms the horizontal implementation even for RS as it requires lower coverage to fully decode the input data for all the error rates simulated. This shows that the vertical layout and integrated consensus aspects of CMOSS design are orthogonal to the choice of error-control codes.

In order to compare the performance of our vertical implementation with the horizontal one, we measured the run time for the experiments presented in Figure 9. By varying the error rates and therefore the coverage required to reconstruct the data, the vertical layout runtime varies between 35-39 minutes while the horizontal implementation takes 5-6 minutes. The difference in time is due to the fact that in the vertical version, every LDPC block stored in columns is re-encoded during the decoding process as shown in Figure 5 in steps (11)-(12). Moreover the freshly generated column of motifs is aligned against reads to fix the starting point of the next column of motifs (Figure 5, steps (12)-(13)). Given the same number of oligos and coverage, vertical encoding will have a constant decoding time overhead compared to horizontal encoding. However, this is not a scalability issue because (i) sequencing takes much longer than decoding, (ii) OB decoding can be easily parallelized, and (iii) in the context of long-term storage, decoding will be done once after several years or decades, and hence the performance of decoding is not a limiting factor.

Finally, we conclude this section by mentioning that in our large-scale wetlab experiments (**Exp. 3**) we limit the size to 13MB due to budget limitations, given the high cost of DNA synthesis. However, we also did a larger-scale simulation study where we converted a random 100GB binary file into 586M oligos spread across 26,421 OB. Using these oligos, we simulated random access with errors using a DNA storage simulator and successfully tested the ability of CMOSS to decode a few specific OB. We omit further details here due to lack of space, but would like to mention that we have validated the efficacy and scalability of CMOSS in decoding large files.

5 CONCLUSION

All SOTA approaches for DNA data archival use an object-based interface and a nucleotide-based, horizontal layout approach for mapping input bits onto oligos. In this paper, we showed how these assumptions (i) amplify PCR coverage bias under a complex pool with files of multiple sizes, and (ii) lead to a strict separation of consensus calling and decoding, which, in turn, results in lost opportunity for improving read/write cost. We presented CMOSS, an end-to-end DNA storage pipeline that uses a vertical oligo layout using motifs as building blocks, and a fixed-size, block/extent-based random access over DNA storage. Using a full system evaluation, we highlighted the benefit of our design and showed that CMOSS can reduce read-write costs compared to SOTA approaches. The CMOSS pipeline is publicly available at https://github.com/paper-submi/dna-storage-system.git.

ACKNOWLEDGMENTS

This work was funded by the European Union's Horizon research and innovation programme projects Glaciation (Grant No. 101070141), SYCLOPS (Grant No. 101092877), and ANR PEPR program MoleculArxiv.

REFERENCES

[1] Leon Anavy, Inbal Vaknin, Orna Atar, Roee Amit, and Zohar Yakhini. 2019. Data storage in DNA with fewer synthesis cycles using composite DNA letters. *Nature Biotechnology* 37 (10 2019). https://doi.org/10.1038/s41587-019-0240-x

[2] Philipp L Antkowiak, Jory Lietard, Mohammad Zalbagi Darestani, Mark M Somoza, Wendelin J Stark, Reinhard Heckel, and Robert N Grass. 2020. Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction. *Nature communications* 11, 1 (2020), 1–10.

[3] Ergude Bao, Tao Jiang, Isgouhi Kaloshian, and Thomas Girke. 2011. SEED: efficient clustering of next-generation sequences. *Bioinformatics* 27, 18 (2011), 2502–2509.

[4] Daniella Bar-Lev, Itai Orr, Omer Sabary, Tuvi Etzion, and Eitan Yaakobi. 2021. Deep DNA storage: Scalable and robust DNA storage via coding theory and deep learning. *arXiv preprint arXiv:2109.00031* (2021).

[5] Tuundefinedkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. 2004. Reconstructing Strings from Random Traces. In

SODA.

[6] Meinolf Blawat, Klaus Gaedke, Ingo Hutter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W. Pruitt, and George M. Church. 2016. Forward Error Correction for DNA Data Storage. *Procedia Comput. Sci.* 80, C (2016).

[7] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. 2016. A DNA-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. 637–649.

[8] Brian Bushnell. 2014. *BBMap: a fast, accurate, splice-aware aligner*. Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[9] Shubham Chandak, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, Tsachy Weissman, and Hanlee Ji. 2019. Improved read/write cost tradeoff in DNA-based data storage using LDPC codes. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing*.

[10] Shubham Chandak, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, Tsachy Weissman, and Hanlee Ji. 2019. 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)*. IEEE, 147–156.

[11] Yuan-Jyue Chen, Christopher N Takahashi, Lee Organick, Callista Bee, Siena Dumas Ang, Patrick Weiss, Bill Peck, Georg Seelig, Luis Ceze, and Karin Strauss. 2020. Quantifying molecular bias in DNA data storage. *Nature communications* 11, 1 (2020), 1–9.

[12] Yeongjae Choi, Taehoon Ryu, Amos Lee, Hansol Choi, Hansaem Lee, Jaejun Park, Suk-Heung Song, Seojoo Kim, Hyeli Kim, Wook Park, and Sunghoon Kwon. 2019. High information capacity DNA-based data storage with augmented encoding characters using degenerate bases. *Scientific Reports* 9 (04 2019). https://doi.org/10.1038/s41598-019-43105-w

[13] George M. Church, Yuan Gao, and Sriram Kosuri. 2012. Next-Generation Digital Information Storage in DNA. *Science* 337, 6102 (2012).

[14] George M Church, Yuan Gao, and Sriram Kosuri. 2012. Next-generation digital information storage in DNA. *Science* 337, 6102 (2012), 1628–1628.

[15] Semiconductor Research Corporation. 2018. 2018 Semiconductor Synthetic Biology Roadmap. https://www.src.org/program/grc/semisynbio/ssb-roadmap-2018-1st-edition_e1004.pdf.

[16] David Reinsel, John Gantz, and John Rydning. 2018. Data age 2025: the digitization of the world from edge to core.

[17] Robert C Edgar. 2010. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics* 26, 19 (2010), 2460–2461.

[18] Alex El-Shaikh, Marius Welzel, Dominik Heider, and Bernhard Seeger. 2022. High-scale random access on DNA storage systems. *NAR genomics and bioinformatics* 4, 1 (2022), lqab126.

[19] Yaniv Erlich and Dina Zielinski. 2017. DNA Fountain enables a robust and efficient storage architecture. *science* 355, 6328 (2017), 950–954.

[20] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. 2012. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics* 28, 23 (2012), 3150–3152.

[21] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. 2013. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *nature* 494, 7435 (2013), 77–80.

[22] Parikshit S. Gopalan, Sergey Yekhanin, Siena Dumas Ang, Nebojsa Jojic, Miklos Racz, Karin Strauss, and Luis Ceze. 2019. Trace reconstruction from noisy polynucleotide sequencer reads. (2019). https://patents.google.com/patent/WO2017189469A1/en

[23] Robert N Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J Stark. 2015. Robust chemical preservation of digital information on DNA in silica with error-correcting codes. *Angewandte Chemie International Edition* 54, 8 (2015), 2552–2555.

[24] Reinhard Heckel, Gediminas Mikutis, and Robert N Grass. 2019. A characterization of the DNA data storage channel. *Scientific reports* 9, 1 (2019), 1–12.

[25] Intel. [n.d.]. Cold Storage in the Cloud: Trends, Challenges, and Solutions. White Paper. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cold-storage-atom-xeon-paper.pdf

[26] Benjamin T James, Brian B Luczak, and Hani Z Girgis. 2018. MeShClust: an intelligent tool for clustering DNA sequences. *Nucleic acids research* 46, 14 (2018), e83–e83.

[27] Jaeho Jeong, Seong-Joon Park, Jae-Won Kim, Jong-Seon No, Ha Hyeon Jeon, Jeong Wook Lee, Albert No, Sunghwan Kim, and Hosung Park. 2021. Cooperative sequence clustering and decoding for DNA storage system with fountain codes. *Bioinformatics* 37, 19 (2021), 3136–3143.

[28] S. Kannan and A. McGregor. 2005. More on reconstructing strings from random traces: insertions and deletions. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.* 297–301. https://doi.org/10.1109/ISIT.2005.1523342

[29] Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal. 2019. Trace Reconstruction: Generalized and Parameterized. *CoRR* abs/1904.09618 (2019). arXiv:1904.09618 http://arxiv.org/abs/1904.09618

[30] Dehui Lin, Yasamin Tabatabaee, Yash Pote, and Djordje Jevdjic. 2022. Managing Reliability Skew in DNA Storage. In *ISCA*.

[31] Kevin Lin, Kevin Volkel, James Tuck, and Albert Keung. 2020. Dynamic and scalable DNA-based information storage. *Nature Communications* 11 (06 2020). https://doi.org/10.1038/s41467-020-16797-2

[32] Abram Magner, Jarosław Duda, Wojciech Szpankowski, and Ananth Grama. 2016. Fundamental Bounds for Sequence Reconstruction From Nanopore Sequencers. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 2, 1 (2016), 92–106. https://doi.org/10.1109/TMBMC.2016.2630056

[33] Eugenio Marinelli and Raja Appuswamy. 2021. OneJoin: Cross-architecture, scalable edit similarity join for DNA data storage using oneAPI. In *ADMS*.

[34] Eugenio Marinelli, Eddy Ghabach, Yiqing Yan, Thomas Bolbroe, Omer Sella, Thomas Heinis, and Raja Appuswamy. 2022. Digital Preservation with Synthetic DNA. *Transactions on Large-Scale Data- and Knowledge-Centered Systems* (2022).

[35] Eugenio Marinelli, Eddy Ghabach, Yiqing Yan, Thomas Bolbroe, Omer Sella, Thomas Heinis, and Raja Appuswamy. 2022. Digital Preservation with Synthetic DNA. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems LI: Special Issue on Data Management-Principles, Technologies and Applications*. Springer, 119–135.

[36] Eugenio Marinelli, Virginie Magnone, Marie-Charlotte Dumargne, Pascal Barbry, and Raja Appuswamy. 2023. Using Soft Information to Improve Error Tolerance of Motif-Based DNA Storage Systems. In *2023 24th International Conference on Digital Signal Processing (DSP)*. 1–5. https://doi.org/10.1109/DSP58604.2023.10167945

[37] Eugenio Marinelli, Yiqing Yan, Virginie Magnone, Charlotte Dumargne, Pascal Barbry, Thomas Heinis, and Raja Appuswamy. 2023. Towards Migration-Free" Just-in-Case" Data Archival for Future Cloud Data Lakes Using Synthetic DNA. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1923–1929.

[38] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, et al. 2018. Random access in large-scale DNA data storage. *Nature biotechnology* 36, 3 (2018), 242–248.

[39] William Press, John Hawkins, Stephen Jones, Jeffrey Schaub, and Ilya Finkelstein. 2020. HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints. *Proceedings of the National Academy of Sciences* 117 (07 2020), 202004821. https://doi.org/10.1073/pnas.2004821117

[40] Guanjin Qu, Zihui Yan, and Huaming Wu. 2022. Clover: tree structure-based efficient DNA clustering for DNA-based data storage. *Briefings in Bioinformatics* 23, 5 (08 2022), bbac336. https://doi.org/10.1093/bib/bbac336 arXiv:https://academic.oup.com/bib/article-pdf/23/5/bbac336/45937680/bbac336.pdf

[41] R. Appuswamy, and Lebrigand, Kevin, and Barbry, Pascal, and Antonini, Marc, and Madderson, Oliver and Freemont, Paul, and MacDonald, James, and Heinis, Thomas. 2019. OligoArchive: Using DNA in the DBMS storage hierarchy. In *CIDR*.

[42] Cyrus Rashtchian, Konstantin Makarychev, Miklós Rácz, Siena Dumas Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. 2017. Clustering billions of reads for DNA data storage. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. 3362–3373.

[43] C. Rashtchian, K. Makarychev, M. Z. Rácz, et al. 2017. Clustering billions of reads for dna data storage. In *Advances in Neural Information Processing Systems (NIPS)*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.), Vol. 2017. Curran Associates Inc., Red Hook, NY, USA, 3360–3371.

[44] Omer Sabary, Guy Shapira, Eitan Yaakobi, and Alexander Yucovich. 2023. Reconstruction algorithms for DNA-storage systems. US Patent App. 17/447,066.

[45] Sundara Rajan Srinivasavaradhan, Sivakanth Gopi, Henry D. Pfister, and Sergey Yekhanin. 2021. Trellis BMA: Coded Trace Reconstruction on IDS Channels for DNA Storage. In *2021 IEEE International Symposium on Information Theory (ISIT)* (Melbourne, Australia). IEEE Press, 2453–2458. https://doi.org/10.1109/ISIT45174.2021.9517821

[46] Horison Information Strategies. 2015. Tiered Storage Takes Center Stage. Report. http://horison.com/publications/tiered-storage-takes-center-stage/

[47] Kyle Tomek, Kevin Volkel, Elaine Indermaur, James Tuck, and Albert Keung. 2021. Promiscuous molecules for smarter file operations in DNA-based data storage. *Nature Communications* 12 (06 2021), 3518. https://doi.org/10.1038/s41467-021-23669-w

[48] Kyle J Tomek, Kevin Volkel, Alexander Simpson, Austin G Hass, Elaine W Indermaur, James M Tuck, and Albert J Keung. 2019. Driving the scalability of DNA-based information storage systems. *ACS synthetic biology* 8, 6 (2019), 1241–1248.

[49] Krishnamurthy Viswanathan and Ram Swaminathan. 2008. Improved string reconstruction over insertion-deletion channels. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, Shang-Hua Teng (Ed.). SIAM, 399–408. http://dl.acm.org/citation.cfm?id=1347082.1347126

[50] Kris A. Wetterstrand. 2024. Ligation Sequencing Kit V14, SQK-LSK114. https://store.nanoporetech.com/eu/ligation-sequencing-kit-v14.html. Accessed: 2024-04-06.

[51] Claris Winston, Lee Organick, David Ward, Luis Ceze, Karin Strauss, and Yuan-Jyue Chen. 2022. Combinatorial PCR method for efficient, selective oligo retrieval from complex oligo pools. *ACS Synthetic Biology* 11, 5 (2022), 1727–1734.

[52] Yiqing Yan, Nimisha Chaturvedi, and Raja Appuswamy. 2021. Accelalign: a fast sequence mapper and aligner based on the seed–embed–extend method. *BMC bioinformatics* 22, 1 (2021), 1–20.

[53] Yiqing Yan, Nimisha Chaturvedi, and Raja Appuswamy. 2022. Optimizing the Accuracy of Randomized Embedding for Sequence Alignment. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 144–151.

[54] S. M. Hossein Tabatabaei Yazdi, Yongbo Yuan, Jian Ma, Huimin Zhao, and Olgica Milenkovic. 2015. A Rewritable, Random-Access DNA-Based Storage System. *Nature Scientific Reports* 5, 14318 (2015).

[55] Eduard Zorita, Pol Cusco, and Guillaume J Filion. 2015. Starcode: sequence clustering based on all-pairs search. *Bioinformatics* 31, 12 (2015), 1913–1919.