Towards Migration-Free Data Preservation using Synthetic DNA

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Eugenio Marinelli

Publicly defended on 24/06/2024 before a committee composed of:

President of the Jury/Reviewer	**Prof. Marc ANTONINI**	CNRS, FR
Examiner/Reviewer	**Prof. Laura CONDE-CANENCIA**	Doralia Technologies, USA
Examiner	**Dr. Maria GIRONE**	CERN, CH
Examiner	**Dr. N. Axel NAUMANN**	CERN, CH
Examiner	**Dr. Dina ZIELINSKI**	WhiteLab Genomics, FR
Co-Supervisor	**Prof. Raja APPUSWAMY**	EURECOM, FR
Thesis Director	**Prof. Pietro MICHIARDI**	EURECOM, FR

Abstract

The growing adoption of AI and data analytics across various sectors has made data preservation a cross-sectoral challenge, affecting everyone from data-driven enterprises to memory institutions. Preserving information over time requires initially maintaining data for a sufficiently long period and being able to retrieve this data from the storage medium in the future. Unfortunately, all contemporary storage media have fundamental density and durability limitations, leading to expensive and cumbersome periodic remastering. This process involves migrating data from an older generation of archival media to a newer one, making cost-effective data archival challenging. In this thesis, we present our vision for achieving migration-free data archival in the context of data preservation using synthetic DNA. In doing so we highlight the challenges in using DNA for data archival and we introduce an innovative, end-to-end DNA storage pipeline we have put in place to overcome those challenges. Thanks to its motif-based encoding method, columnar layout, and integration of consensus calling and decoding, our DNA storage system achieves lower read and write costs compared to the state-of-the-art. We validate our DNA storage system through both simulated and wet-lab experiments, demonstrating that our system outperforms the state of the art in terms of reading and writing costs.

Acknowledgements

At the end of my PhD journey, I would like to express my deepest gratitude to all those who, through their constant support, helped me reach the finish line.

First and foremost, I would like to thank my supervisor, Prof. Raja Appuswamy, for his unwavering support, insightful guidance, and patience. Your expertise and dedication have been fundamental in shaping my research and academic growth.

A heartfelt thanks to all the members of my jury, Prof. Marc Antonini, Prof. Laura Conde-Canencia, Dr. Maria Girone, Dr. N. Axel Naumann, Dr. Dina Zielinski, and Prof. Pietro Michiardi, for taking the time to evaluate my work. Your insightful feedback has significantly enriched my dissertation.

No words can truly express how grateful I am to my family—my mom, dad, and my little brother—for their constant support, not only during this PhD but throughout my entire studies. Your support made all the difference in this long and sometimes difficult journey. I am deeply thankful of the sacrifices you have all made to help me reach this point, and I want you to know that this achievement is as much yours as it is mine. Thank you for always being by my side, no matter what.

Finally, I want to thank my partner for her constant understanding and support, especially during the challenging times; my colleagues for their collaborative spirit and inspiring discussions; and all my friends for the memorable moments we shared that made this journey lighter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Preserving the world around us in order to transmit our heritage to future generations is what characterizes us as human beings. However, in today's increasingly digital world, this drive has become a significant challenge. Nowadays, safely storing digital information to make it accessible in the future represents a critical and open challenge, as digital data is generated at an unprecedented exponential rate. This process, which we can shortly refer to as digital preservation, is vital across several domains, from memory institutions like museums and archives that aim to preserve culturally significant documents to enterprises that need to retain documents for "just-in-case" safety, legal, and regulatory compliance requirements.

To preserve information over time, it is first necessary to maintain data for a sufficiently long period. It is then crucial to be able to retrieve these data from the medium in which they are stored. Finally, when these data are accessed, they must be meaningful to the reader, implying that the format in which they were stored can be correctly interpreted even in the distant future. Unfortunately, the technologies currently used for this purpose, such as HDDs or magnetic tape, cannot guarantee these requirements. None of these technologies can assure a long enough time window to store data without the risk of degradation. Furthermore, none can keep pace in terms of storage density with the exponential increase in data generation, expected to reach 175 ZB by 2025 [5]. In other words, the amount of data is increasing much faster than the capacity of current technologies. Additionally, each new generation of these devices eventually breaks compatibility with some past generations, rendering data stored on, for example, a generation of magnetic disks unreadable by new disk readers designed for newer generations. A small-scale example is the obsolescence of floppy disks, now virtually impossible to read due to the lack of available readers.

Enterprises are the first to be negatively impacted by all this. Recent

studies have found that the amount of data produced is set to increase exponentially. Half of the data generated will be data that enterprises must preserve to meet various compliance requirements. Over 80% of these data are archival data [6], growing at a 60% cumulative annual growth rate and need to be stored for several decades [7]. As mentioned, none of the current technologies can guarantee more than 10 years on the same device, due to both the lifespan of the device itself and the break in compatibility by new devices. Consequently, every decade, enterprises would be forced to undergo costly data migrations from one generation of devices to another.

Another sector greatly impacted by digital preservation issues is memory institutions, such as museums and national archives. The impact is already visible: given the high costs of migration, many are already forced to choose what to continue to preserve and what not. An example is the film industry, where several independent productions are no longer being archived on tape due to rising migration costs [8].

Given these problems, researchers have begun to explore different storage alternatives that allow long-term archiving without data migration. A fundamental requirement for this is to find a storage medium that primarily does not suffer from obsolescence, has high storage density, and can last more than a few decades. Synthetic DNA has recently gained a lot of attention as such a medium. Synthetic DNA has three fundamental advantages over traditional storage technologies: it is extremely dense, eight orders of magnitude more than the best projections for magnetic tape; it can last millennia if properly stored; and it does not suffer from obsolescence, as the methods for reading DNA are also widely used in life science domain.

In this work, we present our vision for migration-free long-term data archival using synthetic DNA, an obsolescence-free biological medium. We will show how synthetic DNA opens up new research avenues by highlighting several open challenges that need to be addressed to make long-term data archival durable and cost-effective. We take the first steps towards concretely realizing this vision by presenting a new end-to-end pipeline for data archival using synthetic DNA, along with a motivating use case of digital preservation of culturally significant data from a national museum using synthetic DNA.

But first, we will set the stage for our research by presenting in more details the limitations affecting traditional storage technologies, along with the alternative solution for obsolescence-free data archival. We will show the limitations that need to be overcome to use DNA as a storage medium and present the state-of-the-art pipelines currently used for achieving data archival using synthetic DNA.

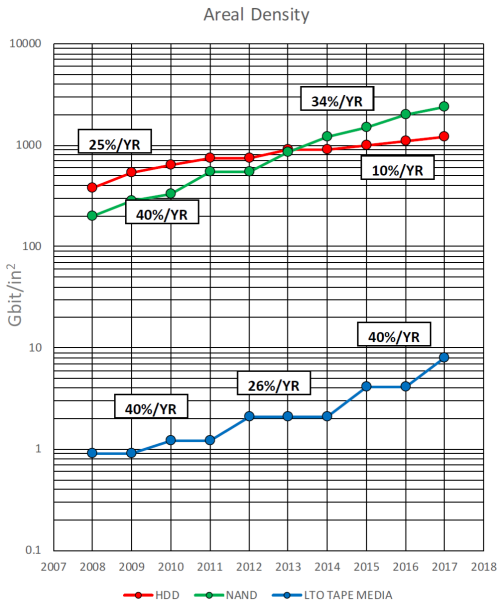

Figure 1.1: Annual rate of storage density improvements for different magnetic technologies [9].

1.1 Motivating Migration-free Data Archival

Traditional magnetic technologies used for data storage, such as hard disk drives (HDDs) and magnetic tapes, face challenges in meeting the requirements of digital preservation. The first limitation is storage density. As the annual size of all data generated is exponential and expected to reach 125ZB by 2025, magnetic technologies struggle to keep the pace with this data growth. Figure 1.1 depicts the trends of Kryder's rate (the analogue of Moore's law for HDD storage density) across various magnetic technologies. While HDD density increases at a rate of 10% per year, magnetic tapes show a slightly better trend, with a 40% increase. However, this increase is insufficient to keep pace with the overall growth in data generation. As the demand for storage space in data centers increases, it necessitates migrating all data to newer devices to accommodate higher storage needs.

The second aspect negatively impacting magnetic technologies for data archival is their limited lifespan. Even if data centers have enough storage devices to meet all data demands, these devices tend to degrade after a few years. In terms of device longevity, the best we can expect is from magnetic tapes, which can guarantee a lifespan of a few decades. However, also in this

case, the major consequence is the need to migrate data to newer devices to prevent data loss.

This leads to the third limitation: media obsolescence. New generations of devices often break compatibility with older generations. An example is represented by magnetic tapes that tend to be readable up to two generations for reading and one generation for writing. This means that to ensure data accessibility, we must migrate data to newer device generations.

All these limitations lead to a common consequence: the need for expensive, cumbersome, and periodic remastering.

1.2 Towards Obsolescence-Free Storage

In order to make long-term data archival cost effective, it is necessary to eliminate the non-scalable, expensive, periodic data migration procedure. Recently, several new initiatives have emerged from both industry and academia in an effort to develop new long-term storage technologies that can overcome the media decay and obsolescence issues faced by contemporary media.

1.2.1 Analog medium.

Historically, analog media like microfilm and paper have been used by libraries and museums for protecting journals across several decades [10], [11]. More recently, film has been used for the preservation of the Declaration of Children's Rights document in collaboration with the UN in the Arctic World Archive [11]. The advantage of analog media is the longevity. For instance, LE-500 rated microfilms and ISO 9706 rated archival paper, are designed to last 500 years. Some analog media, like paper, also does not have obsolescence issues, as any scanning technology can be used to read data off the medium. However, others still suffer from obsolescence as they require dedicated readers that are customized to the media technology which can become obsolete. A major disadvantage with analog media is its density, as all current analog technologies have density much lower than tape (O(KB-MB) per unit for paper/microfilm and O(GB) per unit for film).

1.2.2 Optical medium.

On the optical front, optical discs (like BluRay, Panasonic Archival Disc, etc.) have been used for data archival by cloud-scale systems in production [12]. Recent efforts, like Microsoft Project Silica [13], are pushing the limits of optical storage by using femtosecond lasers to create layers of three-dimensional,

nanoscale deformations in quartz glass. Data is read back by shining polarized light through the glass and analyzing the retrieved image to decode back digital data. Albeit being in its nascency, project Silica has demonstrated the feasibility and durability of glass by storing 76GB of data. While optical media provides much higher density than analog media, their resistance to obsolescence is unclear, as they still require dedicated readers to retrieve data.

1.2.3 Biological medium.

On the biological front, a medium that has received a lot of attention recently is synthetic Deoxyribo Nucleic Acid (DNA). DNA is a macro-molecule that is composed of four submolecules called *nucleotides (nts)* (Adenine (A), Guanine (G), Cytosine (C), Thymine (T)). DNA used for data storage is a single-stranded sequence of these nucleotides. In order to use DNA as an archival medium, digital data is mapped from its binary form into a quaternary sequence of nucleotides using an encoding algorithm. Once encoded, the nucleotide sequence is used to manufacture actual DNA molecules, also referred to as *oligonucleotides* (oligos), through a chemical process called *synthesis* that assembles the DNA one nucleotide at a time. Data stored in DNA is read back by *sequencing* the DNA, which essentially reads out the nucleotide composition of each oligo to produce strings called *reads*, and then decoding the information back from the reads into the original binary form. DNA possesses several key advantages over current storage technologies. First, it is a three-dimensional storage medium with a capacity of storing 1 Exabyte/mm^3 which is $10^8\times$ higher than tape [14]. Theoretically, the Kryder's rate for synthetic DNA is 0, as their density is biologically fixed and cannot increase over the time. However, it is significantly higher than the storage density reachable by magnetic tape. Second, DNA is very durable and can last millennia when stored at room temperature under proper conditions [15]. Third, the technologies used for storing data in DNA (*synthesis*) and reading data from DNA (*sequencing*) have eternal relevance, as there will always be the need to synthesize and sequence DNA for biological applications. Further, as a storage medium, DNA is decoupled from the reader (sequencer), as DNA can be read by any sequencing platform. Hence, DNA does not suffer from media obsolescence.

1.2.4 Technical Errors in DNA Data Storage

There are several challenges in designing encoders for DNA storage. First, there are biological constraints that must be respected during encoding to ensure that DNA molecules can be synthesized and sequenced: (i) experiments

have demonstrated that DNA sequences that have a high number of repeated nucleotides, also known as homopolymers (e.g. TTTT) create problems for sequencing [16]. Thus, the encoder must avoid long homopolymer repeats; (ii) oligos with a low GC-ratio (fraction of Cs and Gs in the oligo) are known to be unstable, while those with a high GC-ratio are known to have higher melting temperatures and create problems for synthesis and sequencing [16]. Thus, the encoder must maintain GC-ratios in a well-defined range.

Second, current synthesis processes cannot synthesize oligos longer than a few hundred nucleotides. Thus, as a single oligo cannot store more than a few hundred bits at best, it is necessary to fragment the data and encode it across several oligos. As DNA molecule itself has no addressing, it is necessary to add addressing information explicitly in the oligo during encoding in order to be able to reorder the oligos later during decoding.

Third, synthesis and sequencing are error prone. There can be insertion errors, where extra nucleotides are added to the original oligo resulting in a sequenced read being longer than the oligo, deletion errors where nucleotides are deleted resulting in shorter reads, and substitution errors. DNA storage also suffers from a *coverage bias* [17]. When DNA strands are sequenced, several noisy copies of the same DNA strand are generated (the *reads*). The average number of reads per oligo generated after sequencing is called *coverage*. Coverage bias refers to the fact that some oligos can be covered by multiple reads, and others can be completely missing as they are not covered by any reads. Coverage bias happens due to the fact DNA synthesis itself creates multiple copies of each DNA molecule. On top of this "physical" redundancy, DNA is also amplified using library preparation steps (like Polymerase Chain Reaction) before sequencing. This amplification creates multiple copies of each synthesized DNA molecule, adding further redundancy. As these amplification procedures are stochastic, different DNA molecules get copied at different rates, leading to coverage bias.

1.2.5 Format obsolescence

Data stored over long time periods suffers not only from media obsolescence (where data stored on a storage media can no longer be read), but also format obsolescence (data, even if it can be read back from the media, cannot be understood as it is in an "extinct" file format). While DNA solves the media obsolescence problem, it does not solve format obsolescence. Modern data lakes use sophisticated file formats that are optimized for fast querying of data, like Arrow, Parquet, Iceberg, and DeltaLake. The open source nature of these formats and associated tooling is certainly a step towards eliminating format obsolescence. However, they are far from ideal as archival storage formats,

as they are continuously evolving without a focus on ensuring backwards compliant data access (future query engines being able to access data stored in older format versions).

1.3 Main Contributions and Thesis Outline

In this thesis work we present our vision to achieve migration-free data archival using synthetic DNA. In doing so, we make the following contributions. (i) We present a compelling use case in which we encoded and manufactured actual DNA strands to archive documents from a national archive. (ii) We introduce an innovative, end-to-end DNA storage pipeline that, thanks to its motif-based encoding method, columnar layout, and integration of consensus calling and decoding, achieves lower read and write costs compared to the state-of-the-art. (iii) We propose three heuristics to compute Log-Likelihood Ratios (LLRs) in the context of Low-Density Parity-Check (LDPC) decoding to enhance error correction capabilities during data retrieval. (iv) Finally, we validate our DNA storage system through both simulated and wet-lab experiments, demonstrating that our system outperforms the state of the art in terms of reading and writing costs. Having explicitly mentioned the contributions, we now present the thesis outline.

Following this introduction, Chapter 2 introduces the necessary background concepts and the current state-of-the-art in DNA data storage. It provides a detailed review of existing technologies and methodologies, offering insights into their limitations and opportunities for advancements. This chapter serves as a critical foundation for understanding the subsequent innovations introduced in the thesis.

Chapter 3 focuses on the collaboration between OligoArchive (the European project that supported this thesis work) and the Danish National Archive in storing culturally significant data. In this context, we introduce motif-based encoding/decoding methods to encode bits into nucleotides and a new consensus algorithm that reconstructs the original oligos with high accuracy but much lower computational time compared to the state-of-the-art. As we will show, the motif-based implementation meets all the constraints imposed by the chemical processes behind the writing and reading of DNA strands while maintaining high storage density (bits/nucleotide).

Chapter 4 exploits the motif-based encoding method introduced in Chapter 3 along with a new data layout, which dictates how bits are stored into the oligos. This new data layout allows us to integrate consensus and decoding in one step. The key idea is that state-of-the-art pipelines rely on consensus calling and decoding as two separate stages. In this chapter, by exploiting

the new data layout and encoding method, we integrate consensus calling and decoding into one step, where the oligos are built progressively while consensus improves decoding, and decoding makes consensus more accurate.

Chapter 5 focuses on LLR estimation for a motif-based DNA storage pipeline. As we will see, DNA storage pipelines rely on two different encoding/decoding steps. The first encoding adds redundancy to data stored to recover those data despite errors; the second encoding maps bits to nucleotides. Given the particular encoding design (based on motifs), and relying internally on LDPC-encoding for redundancy, in this chapter, we extend the work of the previous chapter by providing three heuristics to estimate LLR for the LDPC decoder given the motif-based encoding/decoding method.

Chapter 6 adds the last tassel to the pipeline, introducing support for random access. In real-world cases, we will only access a subset of the billions of files stored in one DNA pool. However, enabling random access is not straightforward. In this chapter, we present an overview of the challenges and the errors a pipeline must handle to support random access and extend our pipeline with a hierarchical structure to allow selective retrieval of only a portion of data.

Chapter 7 concludes the manuscript by synthesizing the key findings and contributions of the research. This chapter not only reflects on the accomplishments and implications of the work presented but also envisions the future of DNA-based data storage. It serves as a bridge between the current state of research and the vast potential for future explorations in the field.

The work presented in this manuscript has been the subject of various publications. The content of Chapter 3 was published in *Proceedings of the 17th International Conference on Digital Preservation, (iPRES 2021)* [1] and *Transactions on Large-Scale Data- and Knowledge-Centered Systems (TLDKS 2021)*, [3]. Chapter 4 was primarily based on content published in *Proceedings of the VLDB Endowment, (VLDB 2023)* [4] and some of the material is in the process of publication. The content of Chapter 5 appears in *24th International Conference on Digital Signal Processing (DSP 2023)* [2]. Finally, content of Chapter 6 is in the process of publication.

Chapter 2

Technical Aspects of DNA Data Storage

DNA, or deoxyribonucleic acid represents a viable alternative to the traditional storage media. DNA is a fundamental molecule that carries hereditary information in humans and almost all other organisms. Every cell in an organism contains identical DNA, located primarily in the nucleus of the cell. DNA is composed of two long strands that wrap around each other to form a double helix, as shown in Figure 2.1. The structure of DNA is based on smaller units known as nucleotides. Each nucleotide includes a sugar molecule (deoxyribose), a phosphate group, and a nitrogenous base. The four nitrogenous bases in DNA are adenine (A), guanine (G), cytosine (C), and thymine (T). Their pairing - A with T and C with G - creates the DNA's double helix structure. The sequence of these bases along the DNA strand is what encodes genetic information. Human DNA comprises about 3 billion bases, of which more than 99 percent are identical in all individuals. The sequence of these bases is critical in determining the information for building and maintaining an organism, similar to the way letters form words and sentences. This molecule, fundamental to the existence of all known living organisms, is not only a carrier of genetic information, but also an efficient means of storing data. DNA's ability to store large amounts of information in a small space has led to the development of a new technology: DNA-based data storage. The remainder of this chapter will introduce the main steps implemented by any state-of-the-art pipeline for data storage on synthetic DNA.

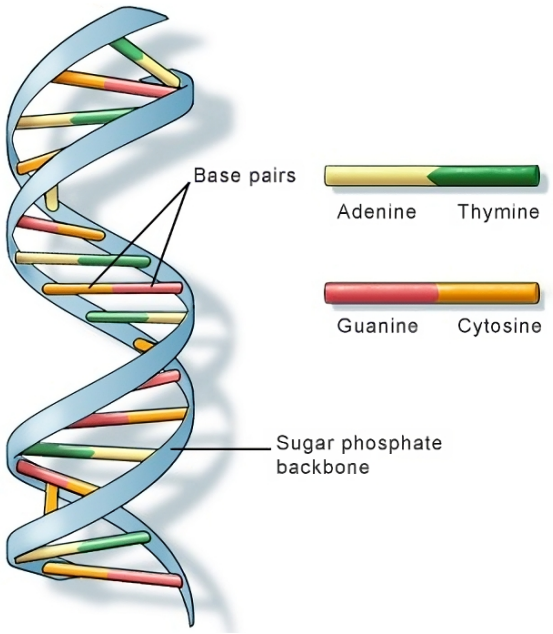

Figure 2.1: DNA double helix structure. [18]

2.1 Traditional DNA Data Storage Pipeline

2.1.1 Data Encoding

Figure 2.2 provides an overview of state-of-the-art (SOTA) DNA storage pipelines. Digital data is stored on DNA by first encoding bits into quaternary sequence of *nucleotides* (Adenine, Guanine, Cytosine, Thymine) – building blocks of the DNA macromolecule. These sequences are then used to create DNA molecules, or oligonucleotides (oligos), via a chemical process called

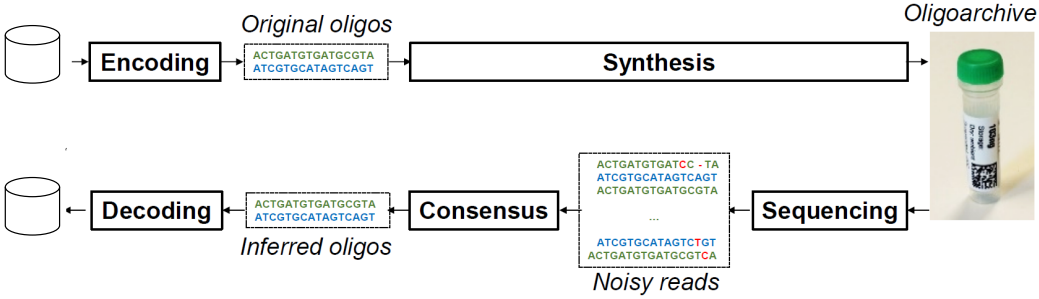

Figure 2.2: Read/write pipelines of SOTA DNA storage solutions.

synthesis. Data stored in DNA is read back via *sequencing*. Both synthesis and sequencing are approximate in nature and prone to errors. Thus, what is retrieved back from DNA are noisy copies of the original sequence referred to as *reads*. Thus, SOTA pipelines use consensus calling to infer original sequences from these reads. The inferred sequences are then decoded back into bits.

As we mentioned in the previous sections, there are several limitations in using DNA as a digital storage medium. First, not all DNA molecules can be synthesized or sequenced but there are some biological constraints that must be respected (G-C constraint, homopolymer repeats, secondary structure formation, etc.) during encoding to ensure downstream compatibility: (i) in order to minimize synthesis errors, the generated sequences cannot have a high number of repeated nucleotides (e.g. TTTTT) or repeated sequences (e.g. ATATAT); (ii) the ratio of Cs and Gs in the oligos must be balanced – i.e., between 30% and 70% – to make synthesis and sequencing possible; (iii) in order to successfully infer oligos, we need *reads* that belong to the same oligo to be similar to each other than reads belonging to other oligos. Thus, the accuracy of inferring the original oligos improves if avoid positionally-similar nucleotide sequences across oligos. SOTA approaches handle these restrictions by using pseudo-randomization of the input bits or developing constrained codes that map bits into valid oligonucleotide sequences. The length of oligonucleotides is another important consideration. Typically, each oligo consists of a few hundred nucleotides, resulting in the binary data to be encoded by several short oligos. Since DNA molecules do not inherently contain addressing information, a portion of each oligo is reserved for encoding indexing information. This indexing is vital for organizing and later retrieving the stored data.

Finally, the actual writing and reading of data to and from the storage medium (i.e., the DNA molecule) occur through two highly error-prone chemical processes: synthesis and sequencing. Consequently, when reading out our DNA strands, we may encounter insertion errors, where extra nucleotides are added to the original oligo, resulting in reads that are longer than the oligo, deletion errors where nucleotides are missing, leading to shorter reads, and substitution errors. The error rates can vary depending on the technology used. In order to ensure reliable data storage despite these errors, SOTA encoding methods rely on two distinct functionalities: (i) error control coding and (ii) consensus calling, as we will present in Section 2.1.4). During the write pipeline, input data bits are grouped into blocks (Fig 2.3(a)), and each block is encoded using error-correction codes, like Reed Solomon codes, LDPC, or fountain codes, to generate parity bits (Fig 2.3(b)). The original data and parity bits are then fragmented to divide them across oligos and

Figure 2.3: Oligo Layout in SOTA Pipeline. The input binary data are divided into logical blocks as highlighted with different colors (a). Each block undergoes encoding to add redundancy in the form of parity bits, and is then further divided into fragments, each assigned an index (b). Finally, each fragment is encoded by mapping bits to nucleotides (c).

indexed (Fig 2.3(b)). Finally, each indexed fragment is converted into an oligo (Fig 2.3(c)).

After the encoding is complete, the generated nucleotide sequences are synthesized into actual DNA strands. As briefly introduced at the beginning of this section, the chemical process used for creating DNA strands in a laboratory is known as synthesis. In the following subsection, we will introduce more details about this technique, providing a brief yet comprehensive overview of its underlying principles.

2.1.2 DNA Synthesis Techniques for Data Encoding

Although DNA naturally occurs in all living organisms as a fundamental component of their cells, DNA can also be artificially created in laboratory through specific chemical processes, commonly known as *synthesis*.

The most common technique used to manufacture DNA is the phosphoramidite-based synthesis – also known as chemical synthesis [19]. This method involves

the sequential addition of nucleotides to a DNA strand affixed to a solid support. This technique has the advantage of being highly accurate in manufacturing custom DNA sequences. However, it tends to be slow and highly expensive. In an effort to significantly lower production costs, recent advancements have introduced enzymatic DNA synthesis [20]. These methods utilize enzymes, which are natural biological catalysts, for DNA synthesis. They show potential for creating longer DNA strands and offer a faster, more cost-effective alternative to chemical synthesis. Yet, enzymatic synthesis tends to be less accurate and more error-prone compared to the chemical synthesis.

Both the previous methodologies involve building DNA one nucleotide at time. An alternative way to build DNA is synthesis by ligation [21]. It consists in using a library of short DNA strands (also referred to as *motifs*). Then, these motifs are linked together to form longer DNA strands. This is done using an enzyme called DNA ligase, which acts like a glue to bond the DNA fragments at their ends. The advantage of this method is that the motif library is built only once using a chemical synthesis; then it can easily replicated multiple times, as copying DNA is way more efficient than manufacturing it. Given this advances, it becomes crucial to develop efficient error correction mechanisms to allow DNA storage systems to accommodate for higher error rates in order to exploit the cost benefits of evolving synthesis techniques.

2.1.3 DNA Sequencing and Data Retrieval

Synthesis techniques allow us to write digital data, encoded as sequences of nucleotides, in form of DNA strings. Also data retrieval requires a chemical process called *sequencing*. Sequencing takes an actual DNA sample as input and generates a dataset of strings, representing a digital version of the DNA sample. In the rest of the thesis, we refer to these strings as *reads*. Ever since Frederick Sanger pioneered the initial sequencing technique in 1977, a variety of other methods have emerged over the years. These have evolved through three generations of sequencing technologies. In order to provide a concise understanding of DNA sequencing technologies without going too deep into their historical development, we present two widely used methodologies at a high level: sequencing-by-synthesis and nanopore sequencing. Sequencing-by-synthesis is a key technique in second-generation sequencing, often known as Next-Generation Sequencing (NGS). The process begins by breaking down DNA into smaller pieces, typically between 100 and 200 nucleotides in length. These fragments are then fixed onto a solid surface and undergo amplification, a process typically achieved using PCR (Polymerase Chain Reaction), as we will see in Section 2.2. PCR is a chemical procedure that generate several

copies of the same DNA strands. This amplification creates clusters of identical DNA fragments, thus enhancing the sequencing signal and resulting in single-stranded DNA templates prepared for sequencing. In the sequencing stage, modified nucleotides and DNA polymerase (an enzyme necessary for appending nucleotides to the developing chain) are introduced into the flow cell. Each nucleotide bears a unique fluorescent marker. Leveraging the principle that each nucleotide base (adenine, thymine, cytosine, and guanine) consistently pairs with a specific complementary counterpart (A with T, C with G), the DNA polymerase adds nucleotides to a growing strand that complements the template strand. As these nucleotides are integrated into the DNA strand during synthesis, they emit a detectable signal, which is then captured and recorded. Next-Generation Sequencing technologies allow for high throughput due to their ability to achieve massive parallelism in sequencing, with an error rate lower than 1% [22], [23]. However, a drawback is that they are limited to processing short sequences, typically not exceeding 500 nucleotides. Nanopore sequencing is one of the methodologies characterizing the third-generation sequencing technology. A DNA sample is placed into a device containing a nanopore, a tiny hole only a few nanometers in diameter. An electrical current is applied across the membrane. As the DNA molecule moves through the nanopore, changes in the electrical current occur. These changes are detected and recorded. This variation is unique for each of the four nucleotides. The sequence of the DNA is determined by analyzing the pattern of current disruption caused by each nucleotide passing through the nanopore. The advantage of this approach is that PCR amplification is not required during sample preparation. Furthermore, it enables the sequencing of longer DNA fragments, often several thousand nucleotides in length, compared to the shorter reads typical of second-generation sequencing. However, this advantage comes with a higher error rate, which can be up to 15% [24]. Therefore, there is a trade-off between the accuracy and cost of reading data from DNA. Longer reads offer several benefits: (i) lower cost and (ii) the requirement for only a small amount of additional information, such as indexes for reordering. However, sequencing techniques that generate longer oligos tend to have higher error rates. Therefore, to reduce the cost of reading data from DNA, we need to implement a system that is able to account for a high error rate.

2.1.4 Clustering and Consensus Calling

In sequencing our DNA samples, we do not directly recover the precise original oligos used for encoding digital data. What we actually receive are several replicas of these initial oligos, as illustrated in the Figure 2.4. Therefore,

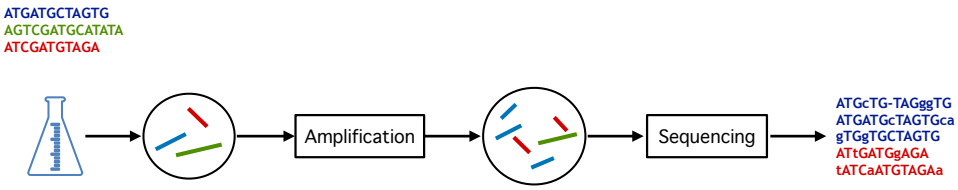

Figure 2.4: Chemical processes in read pipeline: the sinthesized oligos are amplified and sequenced. The resulting dataset contains some oligos (the red and blue) duplicated many times and others completely missing (the green one). The duplicates in the resulting dataset – referred to as reads – contains insertions, deletions and substitutions errors.

the initial step in the decoding process is to organize these sequences by clustering duplicates of the same string together, and clearly separating them from others. Considering that the errors we encounter include insertions, deletions, and substitutions, a straightforward character-by-character comparison between strings is insufficient. Instead, a metric that accounts for indels (insertions/deletions) is utilized: the edit distance. The edit distance between two strings represents the minimum number of insertions, deletions, or substitutions required to transform the first string into the second one, and vice versa. It is a highly complex metric, with its complexity growing quadratically with the length of the strings involved. This complexity is further exacerbated when considering that this metric must be computed during clustering, especially given that it is used each time any two strings are compared for similarity. The *reads* dataset generated by sequencing, which involves millions of strings, significantly amplifies this computational challenge.

After clustering, we process each cluster separately with the goal of determining the most probable original sequence for each cluster. This involves working with an unknown sequence, s, of length L, comprised of nucleotides A, C, T, G. Given N distorted copies of s, where each nucleotide is altered with a probability p (representing deletions, insertions, or substitutions), the task is to reconstruct s by finding a sequence that minimizes the total edit distance to all given inputs. As the noisy input strings originate from the same original oligo, this challenge can be categorized as a trace reconstruction problem in information theory. One notable solution in this domain is the Bitwise Majority Alignment algorithm proposed by Batu et al. [25]. This algorithm effectively reconstructs the original string from its subsequences

Figure 2.5: Example of consensus calling applied to a cluster of three reads in case of (a)substitution only, or insertion and/or deletions (b)-(d).

or traces by employing majority voting and appropriate shifts for each bit of the string. It is particularly adept at handling situations where strings have undergone random deletions, making it a valuable tool for decoding and reconstructing DNA sequences from noisy data. Figure 2.5 show an example of consensus calling when applied to a cluster of three strings. Ideally, if the reads contains mainly substitution errors and the number of reads in the cluster is high enough, consensus calling can simply and accurately infer the correct nucleotide at each position through majority voting. In the example in Figure 2.5(a), we can safely assume that in the first position the correct nucleotides is A, as the first and third string contains an A, while only the second one contain a T. Handling cases with insertions or deletions is more complex. The strategy involves identifying the type of error that occurred at each column and correcting it in cases of deletion or insertion. In Figure 2.5(b)-(d)the reads contain also insertions and substitutions. In the first position we can safely assume that the right nucleotides is A. In the second position we see that in the second and third string the second nucleotide is the T, but the first string contain an A (highlighted in red). Since the first string contains ahead the dinucleotide CG, similarly to the second and third string we can assume that the T in the first string (second position) was an extra insertion. We can delete the extra nucleotide and consensus by majority voting. Notice that we just made an assumption for the type of error and correct it accordingly. However, there is a possibility of misinterpreting the error type. If this happens, the original error, compounded by our corrective attempt, may propagate toward the end of the reads. Consequently, our ability to accurately infer the correct nucleotide decreases as we move further along the index of the read. In the following chapters, we will show our way to deal with this error propagation and the benefits of the proposed solution with respect to the state of the art.

2.1.5 Decoding

The final step in the DNA storage pipeline is decoding the consensus sequences back into the original digital data format. This step ensures that the stored data are accurately retrieved from the DNA sequences. It is a two-phase procedure. First, the nucleotides of the oligos reconstructed during consensus are converted back from nucleotides to bits. Then, it utilizes the additional parity bits, which were integrated during the data encoding phase, to correct errors that consensus could not fix. The error-control decoder plays a critical role in this process. These parity bits provide redundancy that allows the decoder to identify and correct errors, ensuring the integrity and accuracy of the retrieved data. We conclude this chapter by presenting a third chemical process involved in DNA data storage: PCR (Polymerase Chain Reaction). This process, we already mentioned in the Section 2.1.3, is the core mechanism that allows for creating millions of copies of data stored on DNA. It is also the key mechanism that enables random access in DNA data storage. Despite its advantages, PCR introduces another source of error in the DNA storage pipeline. We present the key principles of this process in the following section, while more details on errors introduced by the PCR process and our methods for dealing with them will be discussed in more details in Chapter 6.

2.2 Role of Polymerase Chain Reaction (PCR)

One of the advantages of storing data on DNA is the possibility to create millions of copies in a relatively short time. This can be done using a process that we already mentioned in previous sections: Polymerase Chain Reaction (PCR). This techniques is widely used in molecular biology to produce (amplify) billions of copy a specific region of a DNA sample. For this reason, it is extremely useful in the context of using DNA as data storage. In fact, through PCR we can copy our data and create a backup in a easy and cheap way. In addition to provide a fast and cheap backup mechanism, PCR is also used before sequencing to selectively amplify the DNA strands that we want to read out. At very high level PCR works as follows.

PCR involves a series of temperature changes, carried out in cycles. First, the double-stranded DNA is heated to separate, or denature, into two single strands. Then, the temperature is lowered to allow the primers (short pieces of single-stranded DNA that are complementary to the start and end regions of the target sequence) to bind to their complementary sequences on the single-stranded DNA. Primers serve as the starting point for DNA synthesis. Finally, the temperature is adjusted to the optimal range for DNA polymerase

which adds nucleotides to the primed sequences, synthesizing new strands of DNA complementary to the original strands. Typically, the PCR process consist in 20-40 cycles, where at the end of each cycle the number of DNA samples double. This leads to an exponential growth of DNA samples. As we mentioned, this process plays a crucial role in the context of DNA data storage, by enabling random access for only a portion of data stored on DNA, as we will see in Chapter 6. In the next chapters, we will present our approach to DNA data storage. We start by presenting our first version of a DNA data storage pipeline we put in place to address data archival challenge in the context of a collaboration with the Danish National Archive.

Chapter 3

Digital Preservation with Synthetic DNA

3.1 Introduction

Today, we live in an increasingly digital society. Digital data pervades all disciplines and has established itself as the bed rock that drives our society, from enabling data-driven decisions based on machine learning, to encoding our collective knowledge compactly in a collection of bits. Thus, preservation of digital data has emerged as an important problem that must be addressed by not just memory institutions today, but also by institutions in several other sectors. In order to preserve digital data, it is necessary to first store the data safely over a long time frame. Historically, this task has been complicated due to several issues associated with digital storage media. All current media technologies suffer from density scaling limitations resulting in storage capacity improving at a much slower rate than the rate of data growth. For instance, Hard Disk Drive (HDD) and magnetic tape capacity is improving only 16-33% annually, which is much lower than the 60% growth rate of data [26]. All current media also suffer from media decay that can cause data loss due to silent data corruption, and have very limited lifetime compared to the requirements of digital preservation. For instance, HDD and tape have a lifetime of 5–20 years. A recent survey by the Storage and Networking Industry Association stated that several enterprises regularly archive data for much longer time frames [27]. Thus, the current solution for preserving data involves constantly migrating data every few years to deal with device failures and technology upgrades. A recent article summarized the financial impact of such media obsolescence on the movie industry [8].

In this thesis project, we explored a radically new storage media that has

received a lot of attention recently—Deoxyribo Nucleic Acid (DNA) [28]–[31]. DNA is a macro-molecule that is composed of smaller molecules called *nucleotides*(nt). There are four types of nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). DNA used for data storage is typically a single-stranded sequence of these nucleotides, also referred to as an *oligonucleotide* (oligo). DNA possesses several key advantages over current storage media. First, it is an extremely dense three-dimensional storage medium with a capacity of storing 1 Exabyte/mm^3 which is eight orders of magnitude higher than magnetic tape, the densest medium available today [14]. Second, DNA is very durable and can last millennia in a cold, dry, dark environment. A recent project that attempted to resurrect the Woolly Mammoth using DNA extracted from permafrost fossils that are 5000 years old is testament to the durability of DNA even under adverse conditions [32]. Thus, data once stored in DNA can be left untouched without repeated migration to deal with technology upgrades. Third, as long as there is life on earth, we will always have the necessity and ability to sequence and read genomes, be it for assembling the genome of a previously-unknown species, or for sequencing the genome to detect diseases causing variations. As a result, unlike contemporary storage technologies, where the media that stores data and the technology to read data are tightly interlinked, DNA decouples media (biological molecules) from read technology (sequencing), thus reducing media obsolescence issues.

In this chapter, we provide an overview of the collaboration with the Danish National Archive in demonstrating a holistic solution for long-term preservation of culturally significant data using DNA. We present a motivating use case for long-term digital preservation, outline the challenges involved in using DNA as a digital storage medium, and present the end-to-end pipeline we have put in place to overcome these challenges.

3.2 Context and Background

3.2.1 Danish National Archive Use Case

The Danish National Archives is a knowledge center documenting the historical development of the Danish society. The archive collects, preserves and provide access to original data with the purpose of supporting current and future possible needs of the Danish community - public authorities as well as private citizens. A huge part of this work includes the preservation of digitally created and retro-digitized data securely and cost-effectively. Thus, the archive has received and preserved such data since the 1970s.

The archival material used for this work consists of selected hand-drawings

Figure 3.1: Hand-drawing made by the Danish king Christian IV. The image data back to 1583-1591 and it is part of a larger archive unit we encoded using synthetic DNA.

made by the Danish king Christian IV (1577-1648). An example of one of such drawings is showed in Figure 3.1. Although his reign was marked by military defeat and economic decline, Christian IV stands out as one of the most prominent, popular and admired characters in the line of Danish kings. The hand-drawings date to the period 1583-1591 where the king was 6–14 years old. The material is a part of a larger archival unit consisting of numerous documents and records [1]. The specific image used for this experiment presents a naval battle between several warships. Besides emphasizing the young king's admiration for warfare and naval tactics, the material further indicates his high level of cultural education as well as his talent for drawing. At Danish National Archive, the material is thus ranked as having "Enestående National Betydning" (meaning unique national significance).

3.2.2 DNA storage challenges

Using DNA as a digital storage medium requires mapping digital data from its binary form into a sequence of nucleotides using an encoding algorithm. Once encoded, the nucleotide sequence is used to *synthesize* DNA using a

[1] https://www.flickr.com/photos/statensarkiver

chemical process that assembles the DNA one nucleotide at a time. Data stored in DNA is read back by *sequencing* the DNA molecules and decoding the information back to the original digital data.

A simple way to convert bits into nucleotides is to adopt a direct mapping that converts 2 bits into a nucleotide, for instance 00 to A, 01 to T, 10 to G, and 11 to C. This way a binary sequence is translated to an arbitrary sequence of nucleotides. However, such a simple approach is not feasible due to several biological limitations imposed by DNA synthesis and sequencing steps. First, DNA synthesis limits the size of an oligo between hundred to few thousands of nucleotides. Therefore, data must be divided into several pieces, with each piece being stored in an oligo. However, unlike current storage devices, oligos do not have logical addressing. Hence, indexing information that can help to identify the order in which the oligos, and hence the corresponding data bits, must be reassembled back during recovery must be stored together with the data bits and integrated in each oligo.

Second, oligos with repeat sequences (like ACACAC), or long consecutive repeats of the same nucleotide (like AAAAA), and oligos with extreme GC content, where the ratio of Gs and Cs in the oligo is less than 30% or more than 70%, are known to be difficult to synthesize, sequence, and process correctly. Thus, when constructing oligos, constraints need to be enforced to minimize homopolymer repeats and balance GC content. Further, care must be taken to minimize similarity across oligos as having too many oligos with positionally-similar nucleotide sequences can exacerbate sequencing errors and make it difficult to identify the original oligo.

Third, sequencing and synthesis are not error free even for well-formed oligos, as they introduce substitution errors, where a wrong nucleotide is reported, or indel (insertion/substitution) errors, where spurious nucleotides are inserted or deleted. Both sequencing and synthesis also introduce bias. Some oligos are copied multiple times during synthesis, while others are not. Similarly, some oligos are read thousands of times during sequencing while others are not sequenced at all. Thus, it is important to use error correction codes in order to recover data back despite these errors.

In addition to the aforementioned media-level challenges in using DNA as a digital storage medium, there are also other problems associated with digital preservation that DNA does not solve. Any digital file stored on DNA is an encoded stream of bits whose interpretation makes sense only in the context of the application used to render, manipulate, and interact with that file format. While DNA might be able to store data for millennia, the associated applications and file formats might become obsolete. Thus, in addition to preserving data, it is also necessary to preserve the meaning of data by ensuring that data is stored in a preservation-friendly, non-proprietary

format. Digital data can also be altered due to a variety of reasons and additional data-integrity techniques should be put in place to ensure that data retrieved from DNA can be trusted to be the same as the original source. The digital preservation community has long pioneered file formats, information systems, and operational methodologies for solving such format obsolescence issues [33]. Thus, a holistic DNA-based preservation solution should build on such techniques to solve both media and format obsolescence issues.

3.3 Design

In this section, we will describe the end-to-end pipeline we have put in place to overcome the aforementioned challenges.

3.3.1 Overcoming format obsolescence with SIARD-DK

In Denmark, all public institutions and organizations that produce data worthy of persevering are legally bound to submit them to a public archive. As the vast majority of data in the Danish public sector are organized as databases with or without files in various formats, the focus has been on archiving these data in a standardized, system-independent and cost efficient manner. As a result, the archive has implemented a Danish version of the SIARD format (Software Independent Archiving of Relational Databases) [34] named SIARD-DK for storing of such data. SIARD is an open format, designed for archiving relational databases in a vendor-neutral form and is used in the CEF building block "eArchiving".

The first step in preserving data is extracting it and creating an SIARD-DK Archival information Package (AIP). In the creation of this particular AIP, the digitized material was converted to TIFF format. Information relevant to the images such as the preservation format of the files, their title, creator and original size, descriptive information, etc., was extracted and packaged together with relevant documentation in the AIP-format. This was done using proprietary tools developed at the Danish National Archive. The usage of SIARD for storing the files guarantee that the material is preserved in a rich format with relevant metadata stored in a standardized, system and vendor independent way. The resulting AIP is a single ZIP64 file that internally contains the TIFF images, in addition to XML and XSD files that store the schema of the archive and metadata information. This allows for strict

validation of the AIP. Further, an MD5 value of each file is stored inside the archive and serves as the fixity to verify data integrity on retrieval.

3.3.2 DNA data storage pipeline

The end-to-end DNA media storage pipeline is presented in Figure 3.2. In the rest of this section, we will provide an overview of both the write path that takes as input the SIARD zip file and stores it in DNA, and the read path that restores back the zip file from DNA.

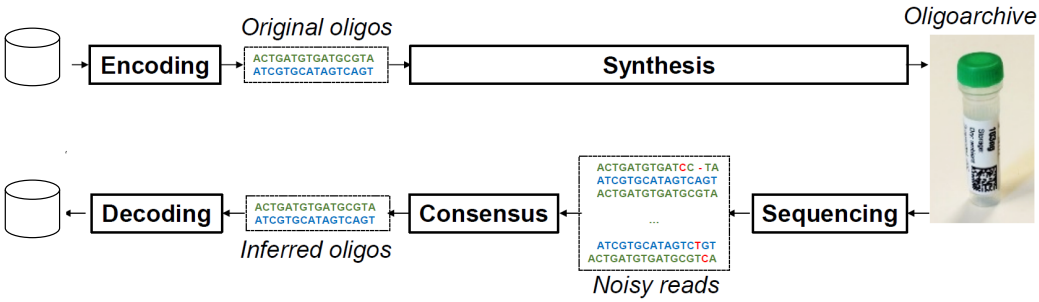

Figure 3.2: DNA Storage Pipeline.

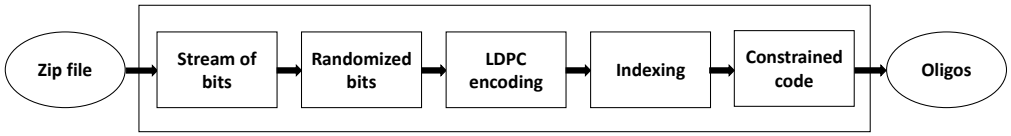

Figure 3.3: Encoding bits into oligos.

Write Path

In order to store the archive on synthetic DNA, the zip file is first encoded from binary into a quaternary sequence of oligonucleotides, and then synthesized to generate synthetic DNA. The steps for encoding the SIARD archive file into oligos is presented in Figure 3.3. During encoding, the file is read as a stream of bits and pseudo randomized. In other encoding methods, randomization is used as a way to limit the number of homopolymer repeats in each oligo. In our encoding, homopolymer repeats are handled by an inner constrained code that we explain later. Thus, we do not need randomization for avoiding homopolymer repeats. We use randomization primarily to improve the accuracy of our clustering and consensus methods in the data decoding stage.

As mentioned before, 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 insertion, deletion, or substitution errors. Our read clustering and consensus methods rely on the fact that the original oligos are well separated in terms of edit distance so that the distance between a noisy read and its corresponding oligo is much smaller than the distance between two oligos. This assumption makes it possible to cluster similar reads and infer the original oligo with a high accuracy. A long sequence of zero or one bits can violate this assumption as they can lead to multiple oligos being similar, or even identical, to each other. A long sequence of bits can also lead to oligos with repetitive sequences (example: ACACACAC). This can pose problems during data decoding, especially if paired-end sequencing is used, where the DNA is partially read from either direction (5" to 3" and 3" to 5") with overlap. In such a case, the two reads corresponding to each orientation must be merged into a single representative read. Repetitive sequences can create issues during this merging process. While we can develop more advanced solutions to perform merging, randomization provides a simple solution that eliminates such issues while ensuring that similarity across oligos is also minimized.

After randomization, error correction encoding is applied to protect the data against errors. We use large-block length Low-Density Parity Check (LDPC) codes [35] with a block size of 256,000 bits as the error correction code, as it has been shown to be able to recover data in the presence of intra-oligo errors, or even if entire oligos are missing [36]. We configure LDPC to add 10% redundancy to convert each sequence of 256,000 bits into 281,600 bits with data and parity. Each 281,600 bit sequence is then used to generate a set of 300-bit sequences, where each 300-bits is composed of 281 data bits and a 19-bit index that is used to order the sequences. Each 300-bit sequence is then passed to a constrained code that converts it into an oligonucleotide sequence.

The constrained code essentially views each oligo as a concatenation of several shorter oligonucleotide sequences, that we henceforth refer to as *motifs*. In our current configuration, the constrained code breaks up each 300-bit sequence into a series of ten 30-bit integers. Each 30-bit integer is fed as input to a *motif generator* that takes the 30-bit value and produces a valid 16nt (nucleotides) motif as output. The motif generator does this mapping by pre-constructing an associative array where the 30-bit value is the key and a 16nt motif is the value. This array is built by first enumerating all possible motifs of length 16nt. Then, all motifs that fail to meet a given set of biological constraints are eliminated.

Our current motif generator is configured to allow up to two homopolymer

repeats (AA,CC,GG, or TT), and admits motifs with a G-C content in the range 0.25 to 0.75. With these constraints, using 16nt motifs, out of 4^{16} possible motifs, we end up with only 1,405,798,178 unique, valid motifs with which we can encode any possible 30-bits of data (as the number of all possible 30-bits values is $2^{30} = 1,073,741,824$ which is lower than 1,405,798,178). Thus, we use the billion motifs as values in the array corresponding to keys in the range 0 to 2^{30}. The 30-bit value is thus used as the key into this associative array to produce the corresponding 16nt motif. Thus, at the motif level, the encoding density is 1.875 bits/nt. The reason we limited ourselves to 16nt and 30bits is the fact that this associative array occupies around 100GB of memory, which we can easily meet using our current hardware. The motif generator can be extended to larger motif sizes and more relaxed biological constraints which can lead to higher bit densities. But as this would require the use of external storage, we leave this open to future work.

Using the associative array, each 300-bit sequence is encoded as concatenation of ten motifs, each with a length of 16nt, leading to an oligo that is 160 nucleotides long. We would like to explicitly point out here that the length of an oligo is a configurable parameter. Thus, while we use 160nts in our current system due to favorable pricing provided by our synthesis provider, our encoder can generate shorter or longer oligos if necessary, and automatically adjust various aspects (like the 19-bit index and 281-bit data size) based on desired oligo length.

To better visualize the motif-based encoding, Figure 3.4 provides a small-scale example. In this example, 6 bits at a time are processed and encoded, each set highlighted in red, blue, and green colors, using motifs that are 5 nucleotides long. For each group of bits, we look up the corresponding motif in an associative array and concatenate the motif to the previous one. Thus, the resulting oligo is composed of the concatenation of these three motifs, resulting in a total of 15 nucleotides. Note that this specific configuration is unique to this example. Throughout the rest of the paper, we refer to a different configuration (groups of 30 bits and motifs of 16 nucleotides), which is our default setup used in the experimental section.

Read Path

To retrieve back the SIARD archive, the DNA is sequenced in order to retrieve back the nucleotide sequence of oligos. As mentioned before, sequencing produces reads, which are noisy copies of the original oligos. Thus, we need a consensus procedure to infer the original oligos from the reads. In prior work, we structured this process as sequence of three algorithms, as depicted in Figure 3.5. First, we identify all pairs of strings that are similar to each

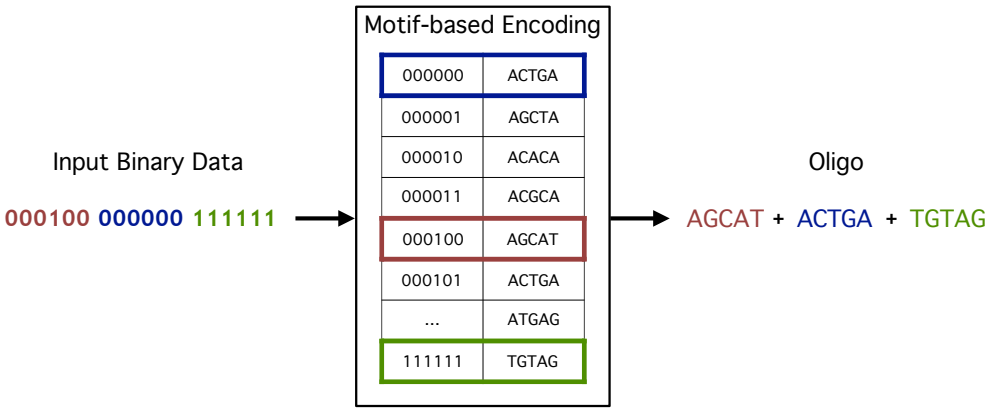

Figure 3.4: Example of motif-based encoding.

other. As modern sequencers produce hundreds of millions of reads, this first task is extremely computationally intensive due to use of the edit distance as a metric for comparing strings. Thus, we have developed an efficient similarity join algorithm, called OneJoin [37], that exploits the fact that due to randomization during encoding, reads corresponding to the same original oligo are "close" to each other despite some errors and "far" from the reads related to other oligos. The results obtained from the join algorithm are then used to quickly identify clusters of strings that are similar to each other. Each cluster thus groups all reads belonging the same oligo. Finally, we apply a position-wise consensus procedure that uses multiple reads to infer the original oligo in each cluster.

While our prior approach was able to infer original oligos with a high accuracy, there were two problems. First, we found that under some datasets, particularly for high coverage reads, OneJoin's memory and computational usage were too high. As OneJoin is a string similarity join, it produces as output all possible pairs of reads that are similar to each other. Thus, given a coverage N, the computational and memory requirements of OneJoin were $O(N^2)$. Second, the use of a general purpose string similarity join led to functionality repetition at multiple places in the read path. For instance, OneJoin internally uses an edit distance check to filter out strings that are not similar. Later in the read pipeline, we had to repeat the edit distance

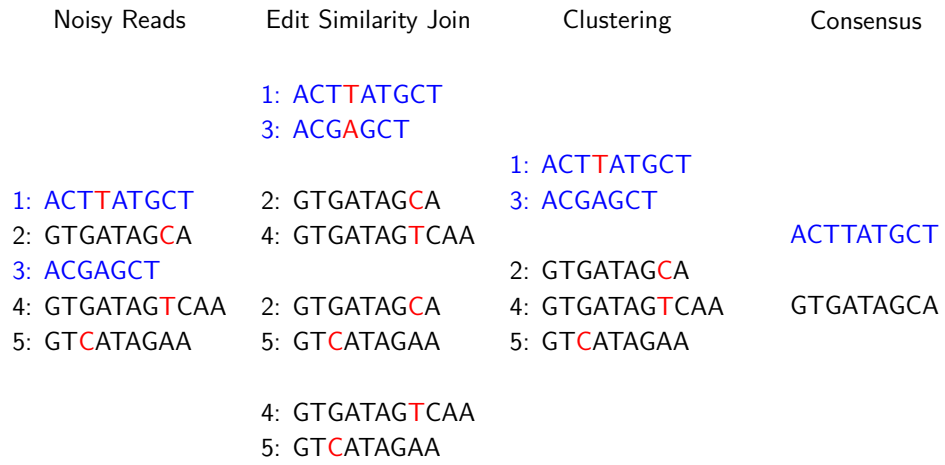

Figure 3.5: Various steps in the OneJoin consensus procedure.

computation in the alignment stage once to get position-wise consensus. This repetition led to needless overhead. To solve these problems, we have developed a new consensus procedure that we refer to as *OneConsensus*.

In the following sections, we describe the key stages of OneConsensus algorithm. In order to efficiently identify and group all the similar reads, our algorithm relies on two well-known algorithmic tools that allow to drastically cut down the computational time: CGK-Embedding and Locality Sensitive Hashing (LSH).

CGK Embedding.

As we mentioned in the previous section, the similarity metric used in OneConsensus is the edit distance. Given two strings x and y, the edit distance is defined as the minimum number of edit operations i.e., insertions, deletions and substitutions, necessary to transform x in y. Another metric, commonly used to compare strings is the Hamming distance. However, the latter takes into account only the number of mismatches between the two strings, or in other words the number of substitutions to transform x in y. For example, given the two strings $ACACT$ and $GACAC$, their Hamming distance is 5 since there are no matches, but the edit distance is 2 since it suffices to add G and remove T.

From these definitions, we can make the observation that the edit distance takes into account information about the ordering of characters and captures the best alignment between two strings. However, while Hamming distance has complexity that is linear with the string length, edit distance has complexity that is quadratic. While several dynamic programming optimizations exist for

accelerating edit distance computations [38], they often rely on pre-specified distance thresholds and are unable to provide performance competitive with fine-tuned Hamming distance computations in practice. Given the complexity of this metric, we rely on randomized embedding techniques to minimize the overhead of the edit distance computations.

Randomized embedding refers to a set of methods that map a complex metric space into a simpler one. `CGK-embedding` algorithm, recently proposed by Chakraborthy et al. [39] is one such algorithm that can map problems from an edit space into a Hamming space. Given two strings x, y of length N taken from an alphabet $\sum$ such that $d_E(x, y)$, the edit distance between x and y, is less than K, CGK-embedding is a function f: $\sum^N \rightarrow \sum^{3N}$ that maps strings x and y into $f(x)$ and $f(y)$ such that, with probability at least 0.99, the Hamming distance of $d_H(f(x), f(y))$ is bounded by K^2 when $d_E(x, y) < K$. This implies that the distortion D, defined as the ratio $D(x, y) = \frac{d_H(f(x),f(y))}{d_E(x,y)}$, is at most K. Thus, as long as the edit distance is small, the distortion of embedding is small. This implies that the Hamming distance of embedded strings will accurately track the edit distance of the original strings, thereby making it possible to replace the expensive edit distance computation with cheap Hamming distance computation [40].

Algorithm 1 CGK-embedding

Input: A string $S \in \{A, C, G, T\}^N$, a random string $R \in \{0, 1\}^{3N}$ and a char for padding $P = 0$
Output: The embedded string $S^{'} \in \sum^{3N}$

```
i ← 0
for j = 0 → 3N − 1 do
    if i < N then
        S'_j ← S_i
    else
        S'_j ← P
    end if
    i ← i + R_j
end for
return S'
```

The pseudo-code of embedding algorithm is shown in Algorithm 1. In this case the procedure is applied to all strings of length N that are composed of the characters A, C, G, T, representing the DNA alphabet. Given an input string, the algorithm builds the corresponding embedded representation by appending one character at time taken from the input string. The character appended can be the repetition of the previous character or the next character

in the input string according to the value of a binary random string. In other words, the pointer of the current character in the input string increases or remains the same depending on the random string value, that can be 0 or 1. When the pointer to the input string goes beyond the string length, the embedded string is padded with a special character P. In general, P can be any character that is not included in the alphabet of the given dataset S. For sake of simplicity, in Algorithm 1 we use 0 for padding. In essence, what we get as the output of embedding is a string where characters from the input string can be repeated one or more times.

LSH for Hamming Distance.

One of the main advantages of moving from the edit distance space to the Hamming space is being able to use some useful algorithms that are valid in the Hamming space only and instead not applicable to the edit distance. One of these algorithms is Locality Sensitive Hashing (LSH) [41].

We call a family $\mathcal{H}$ of functions (d_1, d_2, p_1, p_2)-*sensitive* for a distance function D if for any p, $q \in U$ (where U is the item universe):

- if $D(p, q) \leq d_1$ then $\mathbb{P}[h(p) = h(q)] \geq p_1$, that is, if p and q are close, the probability of a hash collision is high;
- if $D(p, q) \geq d_2$ then $\mathbb{P}[h(p) = h(q)] \leq p_2$, that is, if p and q are far, the probability of a hash collision is low;

where $h \in_r H$ are hash functions randomly sampled from the family of hash functions $\mathcal{H}$

Considering two bit-string p and q of length N. In the Hamming distance case, the hash function is defined as the i^{th} bit of these strings. Thus, if their Hamming distance is $d_H(p, q)$, that is, the number of bits that differ position wise in the two strings, then the probability that any given bit at a random position is the same in both strings is $1 - \frac{d_H(p,q)}{N}$. Thus, the bit-sampling LSH family for Hamming distance, defined as:

$$\mathcal{H}_{\mathcal{N}} = \{h_i : h_i(b_1 ... b_N)) = b_i \mid i \in [N]\}$$

is $\left(d_1, d_2, 1 - \frac{d_1}{N}, 1 - \frac{d_2}{N}\right)$-sensitive for the two Hamming distances $d_1 < d_2$.

We use Hamming LSH over embedded reads to separate out the reads into different buckets such that with a very high probability, reads within a bucket are similar to each other, and hence correspond to the same reference.

Clustering based on Edit Distance

At this stage, we have all reads grouped in hash buckets based on their similarity. However, we still have two problems to solve. (1) LSH can produce false positives, meaning that two dissimilar reads can end up in the same bucket. The main consequences is that if reads are very different, the consensus procedure lead to the wrong result. (2) Reads can have different lengths due to insertions and deletions errors. Thus, we need to adjust the reads in order to make their lengths uniform while taking into account possible insertion/deletion errors. Both these problems can be solved by aligning the reads in each bucket. More specifically, given a bucket, we sort the reads based on length such that reads with length matching the reference oligos are moved to the front of the bucket. Then, starting with the first read in the bucket, we align all the following reads to the first one. The intuition behind sorting the reads is as follows. If a read has the same length as the reference oligos (160nt), either the read has no insertion/deletion errors, or there are an even number of insertion and deletion errors. Since the probability of errors is low with short-read sequencing, the former scenario is more likely. Thus, by picking reads that are of the correct length and aligning the rest of the reads to it, we increase the probability of finding the correct original reference oligo.

The alignment of reads gives two pieces of information: the edit distance and the Compact Idiosyncratic Gapped Alignment Report (CIGAR). The edit distance allows us to identify reads that are actually dissimilar even if they are in the same buckets. We group only similar reads to form a cluster. The CIGAR contains the base-by-base alignment information (the sequence of matches, insertions and deletions) needed to align one read to the other. Using this information, we adjust the reads by adding gaps where there is a deletion error, or deleting nucleotides where there is an insertion error. Once all similar reads are found within the bucket, we save the cluster and remove the reads from the bucket. Any dissimilar reads that were not a part of the cluster are still left in the bucket. In order to deal with false positives by LSH, the remaining reads are then processed again in the same way, with the procedure being repeated until the bucket is empty.

Position-Wise Consensus

The result of the previous stages is a set of clusters. All reads within a cluster are noisy copies of the same reference oligo with some errors in random positions. At this point of the algorithm the only missing step is the consensus procedure. The consensus algorithm works on a per position basis. The key

idea is that despite some random errors, all reads in a cluster are aligned and adjusted based on the edit distance and CIGAR. Thus, if we consider a specific position across all reads, it is likely that the majority of reads in that position will contain the correct nucleotide while only some of them report the wrong one. This implies that for any given position, it will be enough to take the most-frequent nucleotide as the consensus outcome. Repeating this procedure for each position, we produce one inferred oligo per cluster. We would like to point out here that not all oligos need to be correctly inferred. In fact, as we show later, some original oligos might not appear at all in the inferred set, and other inferred oligos might have errors. We rely on the parity added by LDPC codes at a higher level to recover data despite these errors.

The inferred oligos are then passed to the decoder which reverses the steps shown in Figure 3.3. The constrained code is first used to convert each 160nt oligo back into 300-bit sequences by converting each individual 16-nt motif into its corresponding 30-bit value. The index stored in each 300 bits is used to reassemble bits back in the correct order. The LDPC decoder is then used to recover back data even if some bits were wrongly decoded, or some bits were zeroed out as corresponding oligos were missing. The decoded data is then derandomized to obtain a stream of bits that corresponds to the original input.

3.4 Evaluation

At this stage of our collaboration, we have assembled the entire pipeline. In this chapter we do not have results from real experiments yet, but we will provide a preliminary, simulation-driven evaluation in this section. Note that we simulate only the synthesis and sequencing steps in Figure 3.2. We encode/decode the real dataset using our pipeline. The real wet-lab experiment conducted with this dataset will be presented in Chapter 6.

3.4.1 Experimental Setup

The raw SIARD archive that is fed as input to our pipeline is 12.9MB in size. With redundancy added by LDPC, the resulting binary data to be stored on DNA is 15.19MB in size. We encode the SIARD archive generating 405,212 oligos, each with a length of 160nts. Using these original oligos, we then generate four million reads by using a short-read simulator[2] tool, that adds random errors such as insertion, substitution, and deletion in each read to mimic the actions of an Illumina DNA sequencer. This corresponds to an

[2]https://sourceforge.net/projects/bbmap/

average coverage of 10×, meaning that each oligo, on average is covered by 10 noisy copies.

3.4.2 Benchmark with sequence alignment

We begin our analysis by visualizing the coverage across oligos. While the average coverage is 10×, the overall coverage typically follows a negative binomial distribution, with some oligos being covered hundreds of times, and some being not covered at all. To visualize this, we aligned the reads to reference with BWA-MEM v0.7.17 [42], a state-of-the-art short-read aligner. Based on the alignment result from BWA-MEM, we show the histogram of coverage across oligos in Figure 3.6. The x-axis is the coverage (or number of reads that map to an oligo), and y-axis is the number of oligos with that coverage in log scale. As can be seen, the coverage distribution spans a range from 1 to 26, with majority of oligos being covered 5–15× as expected given that the simulator was configured to produce 10× coverage.

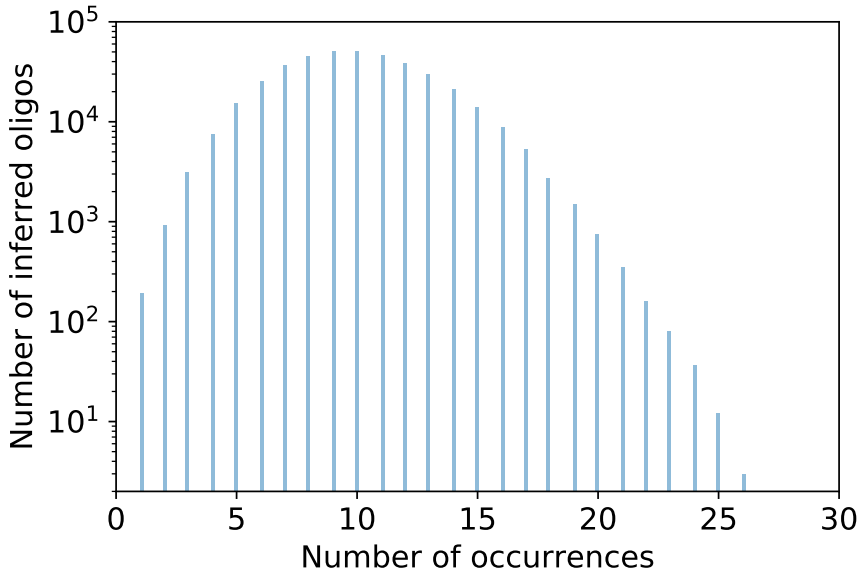

Figure 3.6: Histogram of occurrence of inferred oligos.

We also use the alignment result to show the histogram of number of errors (indels and substitutions) in Figure 3.7, where x-axis represents the number of errors, and y-axis represents the number of oligos with that error count in log scale. As can be seen, the error distribution is right skewed, with a majority of reads having fewer than 2 mismatches, and a few reads having

as many as 8 mismatches which accounts for a 5% error rate given the oligo length of 160. Finally, we also used the alignment result to verify that each read uniquely maps to an oligo (absence of 'XA' field in the SAM output file produced by BWA-MEM). This shows that oligos are "far" from each other in terms of edit distance due to randomization.

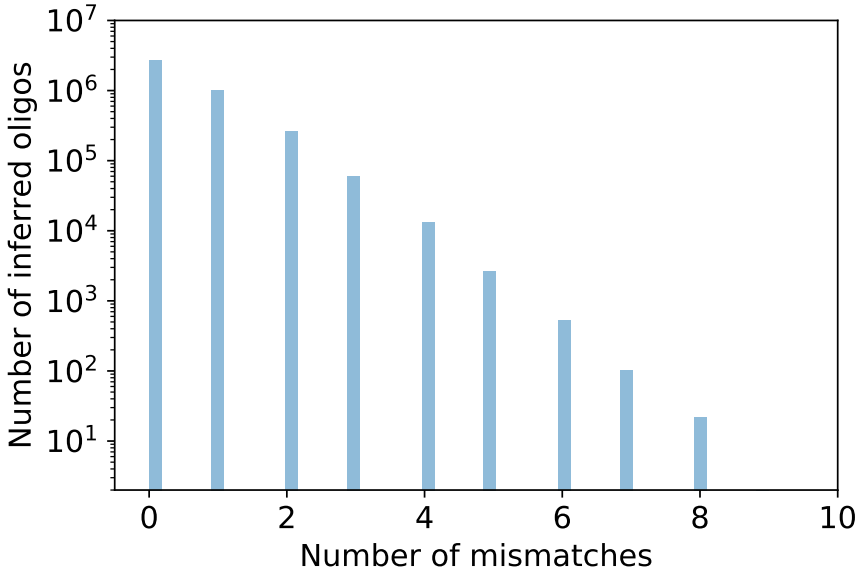

Figure 3.7: Histogram of mismatches.

BWA-MEM performs base-by-base, edit-distance-based alignment and reports CIGAR and alignment score. As a result, it is computationally very intensive. While such an alignment is required for genomic data analysis in order to determine the exact location of a read in the genome, for the purpose of DNA storage, we found that it is sufficient to simply map each read to an oligo without full alignment. Accel-Align (v1.1.1; [43]) is a short-read aligner that we have developed in the context of project OligoArchive that supports alignment-free mapping-only mode that can quickly map reads to oligos. Thus, we present a comparative analysis of BWA-MEM and Accel-Align here with the goal of presenting it as a open-source tool that can be used by other researchers for both DNA storage and more broadly, for analyzing genomic data.

We use the BWA-MEM and Accel-Align to align/map reads to oligos. Using BWA-MEM's alignment as the gold standard, we evaluate Accel-Align's accuracy. Table 3.1 shows the execution time and the percent of correctly aligned reads. It shows that Accel-Align can perform mapping 6x faster than

	Exec. time (second)	Correctly aligned (%)
BWA-MEM	25.5	100
Accel-Align	3.8	99.9

Table 3.1: Performance and accuracy of BWA-MEM and Accel-Align(map mode).

BWA-MEM at a slight drop in accuracy of 0.004%. On futher analysis, we found this drop to be due to Accel-Align's inability to map reads with very high error rate. Accel-Align is specifically built to trade off performance and accuracy for Illumina-based short-read sequencing reads whose error rate is typically much lower than 5%. As our simulated reads offer a more pessimistic error model, Accel-Align experiences a slight drop in accuracy. Despite this, as the coverage histogram with Accel-Align and BWA-MEM are near identical, we have found Accel-Align to be a very useful tool for analyzing both DNA storage reads and more broadly, genomic data [43].

3.4.3 End-to-end Decoding Results

Having presented an analysis of the reads, we will now present the decoding results. Using OneConsensus procedure described earlier, we obtain the inferred oligos using the simulated dataset. Table 3.2 shows the error statistics for these oligos. The figures are obtained by comparing the inferred oligo for a certain index with the corresponding original reference oligo. We see that we are able to infer 404,075 oligos that correspond to 99.7% of the original oligos perfectly without errors. In addition, 1010 oligos were inferred with some errors and 127 oligos were completely missing. Note that errors in an oligo does not imply that the entire oligo is different from the original, but differs only with respect to a few motifs. For this reason, we also report the difference between inferred data and original encoded file in terms of number of bits.

We then use the constrained code to convert these inferred oligos into 300-bit sequences, and reassemble them in order based on the 19-bit index. At this stage, we will certainly have situations where an oligo is missing due to sequencing simulation bias, or an oligo could not be converted back into 300-bits due to errors. In both cases, there will be a corresponding index whose data bits cannot be recovered. We insert a sequence of zero bits for such indices and use the reconstructed binary together with the LDPC decoder to restore the original input. Despite the errors reported in Table 3.2, the LDPC decoder was able to recover back the original archive completely, thanks to

the additional parity information added during encoding.

#Original Oligos	405212
#Correctly Inferred Oligos	404075
#Incorrectly Inferred Oligos	1010
#Missing Oligos	127
#Incorrect bits	42678

Table 3.2: Statistics for decoding of SIARD archive.

Finally, in this section we compare OneConsensus with OneJoin-based consensus algorithm. Table 3.3 compares the two algorithms in terms of the accuracy achieved in inferring the encoding oligos. In terms of the number of oligos correctly inferred in their entirety, meaning an exact match between the inferred oligo and the original reference oligo, we see that OneConsensus slightly underforms OneJoin. But as we mentioned earlier, oligos can also differ by just a few motifs only, making the statistic about the correctly inferred oligos insufficient to determine the actual accuracy of the two algorithms. For this reason, we report also the number of missing oligos and the number of bits wrongly inferred by the two algorithms. The number shows that OneConsensus outperforms OneJoin, as it mistakes only misses 127 oligos, while OneJoin 595 oligos. Overall, OneConsensus leads to 42678 bit errors, compared to the 95935 bit errors produced by the OneJoin-based consensus.

In terms of memory consumption, we observed that OneJoin reaches a peak of 2.5GB, while OneConsensus requires only 1.1GB. While the difference may not seem too striking for this dataset, we would like to point out that while OneConsensus memory consumption grows linearly with the dataset size, OneJoin requires an amount of memory that is quadratic with coverage.

	OneConsensus	OneJoin
#Correctly Inferred Oligos	404075	404104
#Incorrectly Inferred Oligos	1010	513
#Missing Oligos	127	595
#Incorrect bits	42678	95935

Table 3.3: Statistics for OneConsensus and OneJoin-based consensus.

3.5 Conclusion

In this chapter, we provided an overview of the collaboration with the Danish National Archive in using DNA to preserve culturally significant digital data.

Building on prior work on molecular information storage and digital preservation, we presented a holistic, end-to-end pipeline for preserving both data and the meaning of data on DNA. In this chapter we tested the pipeline using simulation studies, while the wet-lab experiment is presented in Chapter 6, where the full pipeline is complete and optimized. In the next chapter we will present an improved design of this DNA storage pipeline, where we investigated various optimizations to both encoding and consensus algorithms to support alternative synthesis and sequencing technologies with potentially higher error rates.

Chapter 4

Columnar Design for Error-Tolerant Database Archival

4.1 Introduction

In the previous chapters, we introduced the problems related to long-term digital data archival. These limitations are mainly inherited directly from the limitations affecting traditional storage technologies: low storage density, limited lifetime, and media obsolescence. All this has led researchers to investigate alternative storage media. Among the different solutions proposed, synthetic DNA stands out. DNA as a storage medium is seven orders of magnitude denser than tape [14] and can store up to 1 Exabyte of data in a cubic millimeter [28]. Using common, well-established biochemical techniques, DNA can be replicated rapidly, allowing for easy copying of data stored in DNA. It is extremely durable and can last several millennia when stored under proper conditions. Moreover, DNA is read through the sequencing process, and the sequencing technology used is decoupled from DNA itself, the storage medium. Thus, DNA will not suffer from obsolescence issues, as we will always be able to read back data stored in DNA. Given these benefits, in Chapter 3, we presented a holistic solution to archive culturally significant data from the Danish National Archive. We achieved this by implementing an end-to-end DNA data storage pipeline that exploits a motif-based encoding/decoding to overcome the limitations of DNA-based media encoding and using a vendor-neutral file format to store the archive. The primary obstacle to DNA storage adoption today is the prohibitive cost of reading and writing data. The biochemical processes used for writing (*synthesis*) and reading (*sequencing*)

DNA today were originally designed for biological applications that require very high precision and low scale. Using DNA as a storage medium requires a different trade off, as one can tolerate more errors in synthesis and sequencing for improved cost efficiency and scaling.

In order to provide reliable data storage on DNA despite such errors, SOTA approaches rely on using a significant amount of redundancy in both writing (in the form of parity bits generated by error control coding) and reading pipelines (in the form of very 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 reduce overall cost.

In this chapter, we present *OligoArchive-DSM(OA-DSM)*, an end-to-end pipeline for DNA storage that provides substantially lower read/write costs than SOTA approaches. OA-DSM builds on top of the pipeline presented in Chapter 3 and for this reason it exploits the motif-based encoding/decoding to convert bits into nucleotides. The core contributions of this work, and the two key aspects of OA-DSM that distinguish it from SOTA approaches are: (i) a novel, database-inspired, columnar encoding method for DNA storage, and (ii) an integrated consensus and decoding technique that exploits the columnar organization. In this chapter, we provide an overview of challenges in DNA storage (Section 4.2), present the aforementioned aspects of OA-DSM design in detail (Section 4.3), and demonstrate their ability to achieve better accuracy and higher error-tolerance than SOTA methods using both simulation studies and a real wet-lab validation experiment where we successfully encoded and decoded a 1.2MB compressed TPC-H database archive.

4.2 Background

When used as a storage medium, DNA introduces several errors at different stages of the read and write pipelines. These errors are common to all DNA storage pipelines. In this section, we will provide an overview of these errors.

In all SOTA pipelines, binary data is stored in DNA by transforming the binary input into a quaternary sequence of nucleotides (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 quaternary sequences that correspond to the nucleotide composition of oligos. A software decoder is then used to convert the quaternary sequences into the original binary input.

Given a set of N sequences generated by the encoder, we would ideally

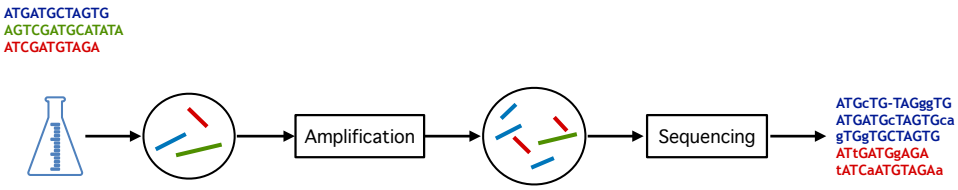

Figure 4.1: Duplicates generations during data retrieval.

expect synthesis to produce N oligos, and sequencing to produce N reads, with the reads being identical to the original sequences generated during encoding. However, it is well known that both synthesis and sequencing procedures are not precise as they introduce both duplication and errors, with the extent of duplication and types of errors (substitution, insertion, and deletion) varying depending on the technology used. As a result, as the outcome of sequencing, we actually receive several noisy replicas of the original encoded sequences (Figure 4.1). Thus, in all SOTA pipelines, the first step in the decoding process is to organize these sequences by clustering duplicates of the same string into distinct groups. Several clustering algorithms have been proposed for this purpose. CD-HIT [44] and UCLUST [45] leverage greedy algorithms for incremental sequence clustering, offering speed at the expense of optimality. Bao et al. [46] and Antkowiak et al. [47]'s solutions employ advanced hashing techniques, prioritizing efficient indexing and location-sensitive clustering. Starcode [48] and Meshclust [49] provide solutions based on precise distance calculations, with Starcode utilizing edit matrices for Levenshtein distances and Meshclust employing the mean-shift algorithm to address parameter sensitivity. The method of Jeong et al. [50] focuses on Hamming distance-based clustering. Microsoft's approach [51] uses a minimal hash algorithm that enables accurate clustering over large datasets. Clover [52] has enhanced previous incremental algorithms in terms of accuracy and scalability, using a trie-based data structure for sequence comparison. Finally, in prior work [37], [53], we proposed a clustering algorithm based on embedding and locality-sensitive hashing to scale clustering over large datasets.

After clustering, each cluster is processed separately in order to determine the most probable original sequence. We refer to this task as *consensus calling*. This task can be formalized as follows. The original oligo s, of length L, comprises the nucleotides A, C, T, G. Given N distorted copies of s, where each nucleotide is altered with a probability p (representing deletions,

insertions, or substitutions), the task is to reconstruct s by finding a sequence that minimizes the total edit distance to all given inputs. As the noisy input strings originate from the same original oligo, this challenge can be categorized as a trace reconstruction problem in information theory. Several solutions have been proposed in literature for this problem [54]–[61]. An example is the Bitwise Majority Alignment algorithm proposed by Batu et al. [25] which is used for oligo reconstruction in prior DNA storage experiments [62]. This algorithm effectively reconstructs the original string from its subsequences or traces by employing majority voting and appropriate shifts for each bit of the string.

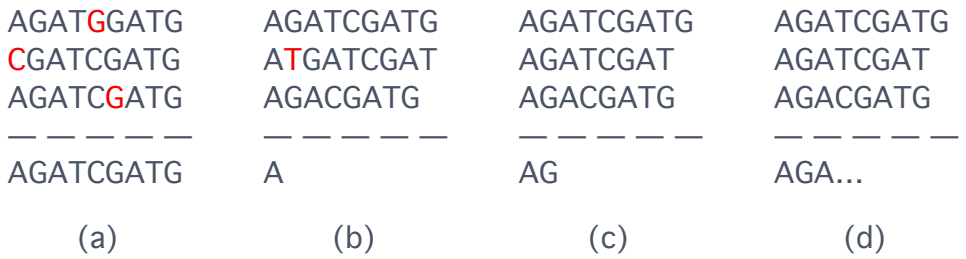

Figure 4.2: Example of consensus algorithm applied to a cluster of three strings in case of substitution errors only (a) and insertion/deletion errors (b)-(d).

Figure 4.2 shows an example of consensus algorithm applied to a cluster of three strings. When the noisy reads contain mostly substitution errors (Figure 4.2(a)), and coverage (roughly defined as the number of times an oligo appears in the sequenced reads) is sufficient, we can infer the correct nucleotide at each position through majority voting. For instance, in the example, we can assume that the first nucleotide is A, as both the first and third strings have an A as their first nucleotide. This same procedure applies to the rest of the column (nucleotides).

Handling cases with insertions or deletions is more complex. In Figure 4.2(c)-(d) we have the same three strings but with insertion and deletion error. When we apply consensus to the first character, as none of the strings present any error in the first position and they all have an A, we can assume that the first nucleotide is A (Figure 4.2(b)). Continuing with the algorithm for the second position (Figure 4.2(c)), we see that the three strings differs as the first and third string contain the nucleotide G, while the second string has the nucleotide T. At this point we have to make an assumption. If we look one character ahead, we see that the second strings have the dinucleotide GA, similarly to the first and second strings. We can assume that the T was

an insertion in the second string. We can correct the insertion, by deleting the red nucleotide T in the second string and and assume G as nucleotide in the consensus resulting string.

Notice that, a different consensus algorithm could have assumed something different, for example substitution error in the second string, where instead of G we have T. Similarly, the the procedure is repeated for the remaining nucleotides (Figure 4.2(d)). Notice that every attempt to correct the error in our strings is based on an assumption which means that there is a possibility of misinterpreting the error type. If this happens the original error and our wrong corrective attempt propagates toward the end of the reads. As the consensus calling works symmetrically whether we start the beginning or from the end of the reads, most of the errors will accumulate in the middle of the reads. Lin et al. [63] called this the reliability bias and showed that is not due to a specific consensus algorithm but it can be observed even if we use an optimal consensus algorithm. Hence, it is an intrinsic property of the trace reconstruction problem when insertion and deletion errors are present.

The reliability bias carries significant repercussions for DNA data storage. Regarding the synthesis of DNA, as techniques improve and enable the creation of longer oligos (like Nanopore long-read sequencers), the consensus bias issue becomes more pronounced. This is because the extent of the bias is directly related to the length of the oligos. In terms of reading, while sequencing technologies are becoming more cost effective (like Nanopore long-read sequencers), they are also experiencing an increase in error rates. This trend makes the consensus algorithm less reliable due to this bias, necessitating higher sequencing coverage to effectively manage these errors. Recall that the term *coverage* refers to the number of time the same original oligo is processed during sequencing, i.e., the number of duplicates per original oligo. It is important to note that increased coverage translates to higher sequencing costs.

4.3 Design

Having described the reliability bias issue of DNA storage, we now present OligoArchive-DSM (OA-DSM). All SOTA pipelines share two characteristics: (i) an error-control-coded block of input data is encoded to generate a group of oligos that forms a unit of recovery, (ii) an isolated consensus step is performed before decoding to infer oligos from noisy reads. The decoding and consensus stages are independent steps in all SOTA pipelines and do not interact with each other. Our approach to archiving data in OA-DSM differs from SOTA based on the key observation that the separation of consensus

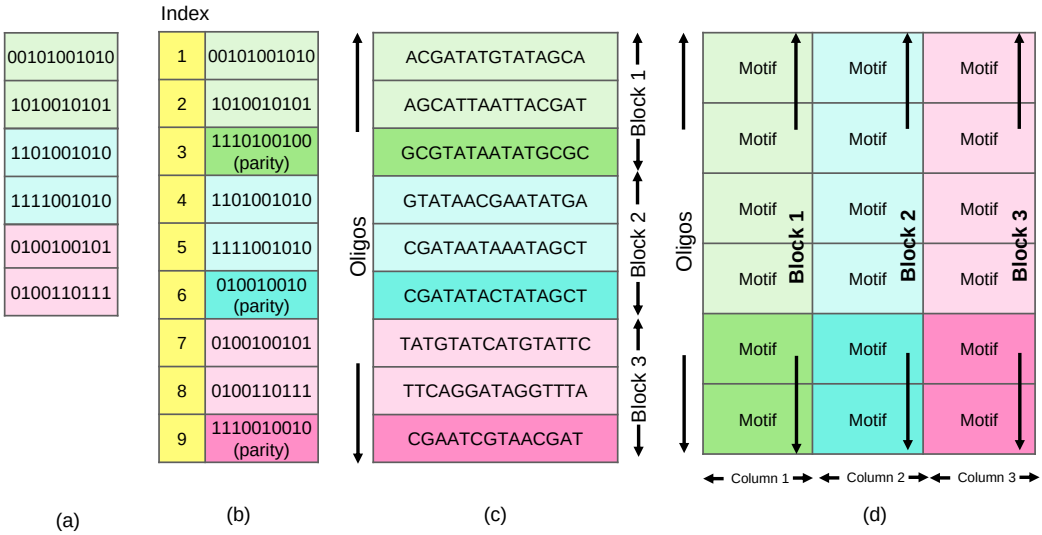

Figure 4.3: Comparison of SOTA versus OA-DSM columnar layout of oligos. The figures shows the raw input data being grouped into blocks (a), each block encoded to generate parity and indexed (b). (c) shows each block of input being mapped to multiple oligos with SOTA approaches. (d) shows each block being mapped to one column of motifs with OA-DSM.

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 key idea in our system is to change the layout from the horizontal, row-style SOTA layout (Figure 4.3(c)) to a vertical, column-style cross-oligo layout (Figure 4.3(d)). Our DNA storage system encodes and decodes data vertically across several oligos instead of horizontally. A set of oligos is viewed like a relation, with each oligo being a row. OA-DSM encodes and decodes data one column at a time. The key benefit of this, as we show later in this section, is the fact that OA-DSM can integrate 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. In the rest of this section, we will explain the OA-DSM design in more detail by presenting its read and write pipelines.

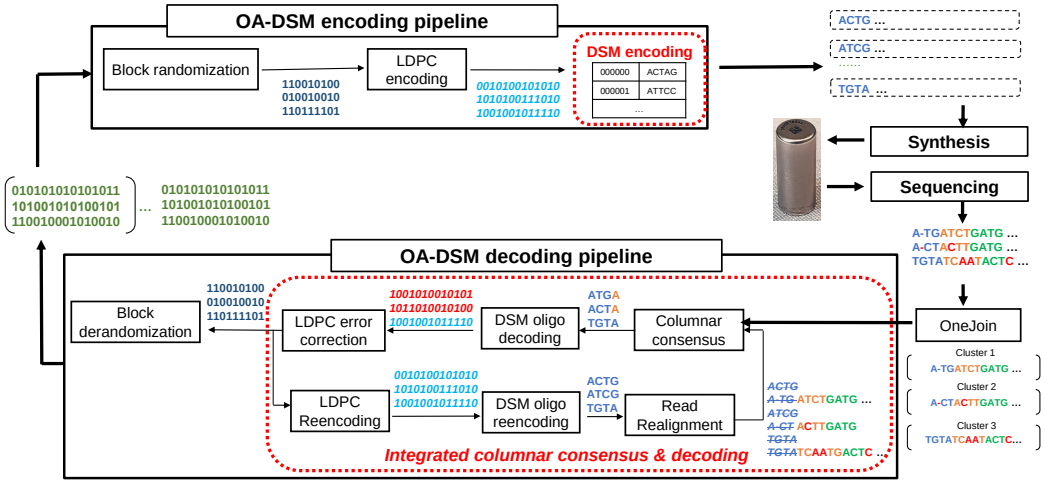

Figure 4.4: OA-DSM data writing pipeline (top) showing binary to DNA encoding path, and long-term DNA storage in an encapsulated container like Imagene DNAShell™. OA-DSM reading pipeline (bottom) showing DNA to binary decoding path. The blocks in red are unique to OA-DSM (versus SOTA).

4.3.1 OA-DSM Write Pipeline

The top half of the Figure 4.4 shows the OA-DSM data writing pipeline. 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 it into blocks of size 256,000 bits. Each block of input is then randomized. While it is not relevant to this discussion, we use randomization similar to SOTA to improve the accuracy of read clustering in the data decoding stage as explained in Section 4.3.2. After randomization, error correction encoding is applied to protect the data against errors. We use Low-Density Parity Check (LDPC) codes [35] with a block size of 256,000 bits. Prior work 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 [36].

The LDPC encoded bit sequence is fed as input to the *DSM-oligo-encoder* which converts bits into oligos. While SOTA approaches design each oligo as a random collection of nucleotides, the DSM-oligo-encoder designs oligos using composable building blocks called *motifs*. Each motif is itself 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 nucleotides as building blocks because, as we will see later

in Section 4.3.2, integration of decoding and consensus relies on alignment which cannot be done over single nucleotides.

In order to perform the conversion of bits into motifs, the DSM-oligo-encoder maintains an associative array with a 30-bit integer key and a 16 nucleotide-length (nt) 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 0.25 to 0.75. With these constraints, using 16nt motifs, out of 4^{16} possible motifs, we end up with 1,405,798,178 that are valid. By mapping each motif to an integer in the range 0 to $2^{30} - 1$, we can encode 30-bits of data per motif. Thus, at the motif level, the encoding density is 1.875 bits/nt. While we can increase this density by increasing motif size, or relaxing biological constraints, we limited ourselves to this configuration due to two reasons: (i) memory limitation of our current hardware, as the current associative array itself occupies 100GB of memory, (ii) the motif design is orthogonal to the columnar encoding which is the focus of this work. In future work, we plan to increase this bit density by expanding to large motif sets. The use of motifs as building blocks renders a distinct relational organization to oligos–just as a set of attributes form a tuple, and a set of tuples form a relation, a set of motifs (attributes) forms an oligo (row), and a set of oligos constitutes an OligoArchive. Thus, the second major difference of our approach to SOTA is the layout of motifs across oligos which is reminiscent of Decomposition Storage Model (DSM), or columnar data layout, adopted by modern analytical database engines. The motifs generated from an error-control coded data block are used to extend oligos by adding a new column as shown in Figure 4.3(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 again from the first column. The generated oligos can then be synthesized to produce DNA molecules that archive data.

Figure 4.5 illustrates an example of the columnar layout of motifs, clearly distinguishing it from a linear approach. In our columnar layout, motifs from the same LDPC block are arranged column-wise. For instance, motifs from LDPC-block-1 form the first column, with each motif belonging to a different row. LDPC-block-2 is then encoded to form a second column, which is concatenated alongside the first one. Essentially, the motifs within an LDPC block are concatenated vertically to form a column, and multiple columns expand the oligos dataset horizontally. In contrast, a linear layout appends motifs directly in a horizontal sequence; when the maximum length for an oligo is reached, a new oligo begins. It is important to note that in the

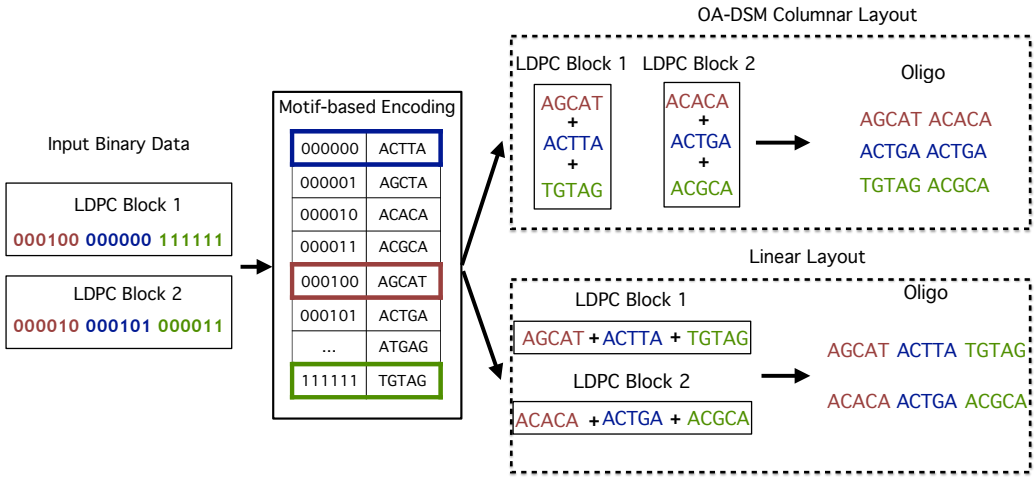

Figure 4.5: Different data layout (columnar vs linear) in the encoding process.

example, each LDPC block is represented as a different oligo in the horizontal layout, but in practical scenarios, each block is spread across several oligos. While the figure shows all columns as being of the same size, a small subtlety in the practical implementation is the distinction between the first column and the rest. As we need to index the oligos to enable reordering during decoding, the first column of motifs is generated by using a 15-bit address and a 15-bit data to generate a 30-bit integer. Thus, the first LDPC encoded block is decomposed into 15-bit integers. However, from the second column, there is no need to add addressing information. Thus, rest of the LDPC blocks are decomposed into 30-bit integers. Note that with 15-bit addresses, we can address up to 32,768 oligos. However, we will see in Chapter 6 how we extended OA-DSM to develop a block-addressed, randomly-accessible, DNA storage system, that allows us to view a column like a disk block, and a collection of columns like an extent. The 15-bit address here provides intra-extent addressing. Extents themselves will be addressed separately using a separate mechanism (nested primers). We will address the extent design in Chapter 6, but we explicitly mention this here to clarify that OA-DSM can scale to much larger oligo pools. But for the rest of this chapter, we focus on columnar design and consensus.

4.3.2 OA-DSM Read Pipeline

As mentioned before, 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 insertion, deletion, or substitution errors. As each oligo can be covered by multiple reads, the first in decoding is clustering to group related reads together. In prior work, we developed an efficient clustering technique based on edit similarity joins [37], [53] that exploits the fact that due to randomization during encoding, reads corresponding to the same original oligo are "close" to each other despite errors and "far" from the reads related to other oligos. The output of this algorithm is a set of clusters, each corresponding to some unknown original oligo.

After the clustering stage, other SOTA methods apply consensus in each cluster followed by decoding in two separate phases as shown in Figure 2.2. In OA-DSM, we exploit the motif design and columnar layout of oligos to iteratively perform consensus and decoding in an integrated fashion as shown in Figure 4.4. Unlike other approaches, OA-DSM processes the reads one column at a time. Thus, the first step is columnar consensus which takes as input the set of reads and produces one column of motifs. The choice of consensus algorithm is orthogonal to OA-DSM design. We use an alignment-based bitwise majority algorithm we developed previously for consensus [53], as we found this to provide accuracy comparable to other state-of-the-art trace reconstruction solutions [25]. The motifs obtained from consensus are then fed to the *DSM-oligo-decoder* which is the inverse of the encoder, as it maps the motifs into their 30-bit values. Note here that despite consensus, the inferred motifs can still have errors. These wrong motifs will result in wrong 30-bit values. These errors are fixed by the *LDPC-decoder*, which takes as input the 30-bit values corresponding to one LDPC block and produces as output the error-corrected, randomized input bits. These input bits are then derandomized to produce the original input bits for that block.

As mentioned earlier, SOTA methods do not use the error-corrected input bits during decoding. OA-DSM, in contrast, uses these bits to improve accuracy as shown in the bottom part of the integrated columnar consensus in Figure 4.4. The error-corrected bits produced by the LDPC-decoder are reencoded again by passing them through the LDPC-encoder and DSM-oligo-encoder. This once again produces a column of motifs as it would have been done during input processing. The correct column of motifs is used to realign reads so that the next round of columnar decoding starts at the correct offset. The intuition behind this realignment is as follows. An insertion or deletion error in the consensus motifs will not only affect that motif, but also all downstream motifs also due to a variation in length. For

instance, if we look at the example in Figure 4.4, we see a deletion error in the sequenced read (bottom-right of the figure) $A-TGATCTG..$ which should have been $ACTGATCTG....$ This results in the first motif being incorrectly interpreted as $ATGA$ (instead of $ACTG$, and second motif as $TCTG$ (instead of $ATCT$). Thus, an error early in consensus keeps propagating. Without a knowledge of the correct motif, there is no way to fix this error. But in OA-DSM, by reencoding the error-corrected bits, we get the correct motifs. By aligning these motifs against the reads, we can ensure that consensus errors do not propagate. Note here that such 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, columnar layout without motifs, or with just nucleotides, would not make realignment possible. Similarly, integrating consensus and decoding is possible only because of the columnar layout, as the SOTA layout that spreads a LDPC block across several oligos cannot provide incremental reconstruction.

4.4 Evaluation

In this section, we will present the results from our experimental evaluation of the OA-DSM pipeline. The evaluation is structured as follows. First, we present the results from a small-scale wetlab experiments to validate the end-to-end OA-DSM pipeline (Sec. 4.4.1). Then, we compare OA-DSM with various SOTA approaches with respect to read cost and write cost to show that our design can lead to substantial cost reduction (Sec. 4.4.2). Finally, we show the advantage of using a columnar design by comparing OA-DSM with a row-based pipeline (Sec. 4.4.3).

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 of the OA-DSM pipeline shown in Figure 4.4 has been implemented in C++17. We use TPC-H dbgen utility to generate compressed, synthetic data that we treat as the archival file that must be stored on DNA. We parameterize dbgen to control the generated database size according to experimental requirements as mentioned later.

4.4.1 Small-Scal Wet-Lab Validation

As the first prototype test, we used the TPC-H DBGEN utility to generate a compressed database of 1.2MB. The cardinality of various relations in the archive is reported in Table 4.1. Using OA-DSM configured with 30% LDPC redundancy, we encoded the archive file to generate 44376 oligos, with each

Table Name	Number of Rows	Size [Byte]
customer	900	142782
lineitem	31220	3717358
nation	25	2199
orders	9000	982200
partsupp	4800	690415
part	1200	140596
region	5	384
supplier	60	8242

Table 4.1: TPCH database summary for WetLab experiment.

oligo of length 160nts (length chosen to optimize synthesis cost). The oligos were synthesized by Twist Biosciences. We sequenced the synthesized oligos using Oxford Nanopore PromethION platform generating approximately 43 million noisy reads.

To perform error characterization, we aligned the 43M reads to the original oligos using Accel-Align seqence aligner[64], [65]. 99.9999% reads were aligned to a reference oligo, indicating a very high quality of the generated read set. Figure 4.6 shows the coverage histogram (number of oligos that have a given coverage). 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.

Figure 4.7 shows the substitution, insertion and deletion rate per position (as computed by BBmap [66]). Note that while the data-carrying oligo had a length of 160nts, our reads are longer as they include the primers that were appended at both ends of the oligo for sequencing. As these primers 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 substition rate is dominant, which is 3× higher than insertion and deletion rates. Figure 4.8 compares our error rates with those reported in prior work on DNA storage [62], [67]–[70]. While the actual rates vary due to differences in synthesis and sequencing steps, we see that the overall trends are similar. Using the aligned reads, we also report the indel distribution in Figure 4.9 which shows a histogram of edit distances between the reads and references. As can be seen, 96.97% reads have edit distance less than 10, indicating that the error rate is less than 6%.

In order to test end-to-end decoding, we first used the full 43M read

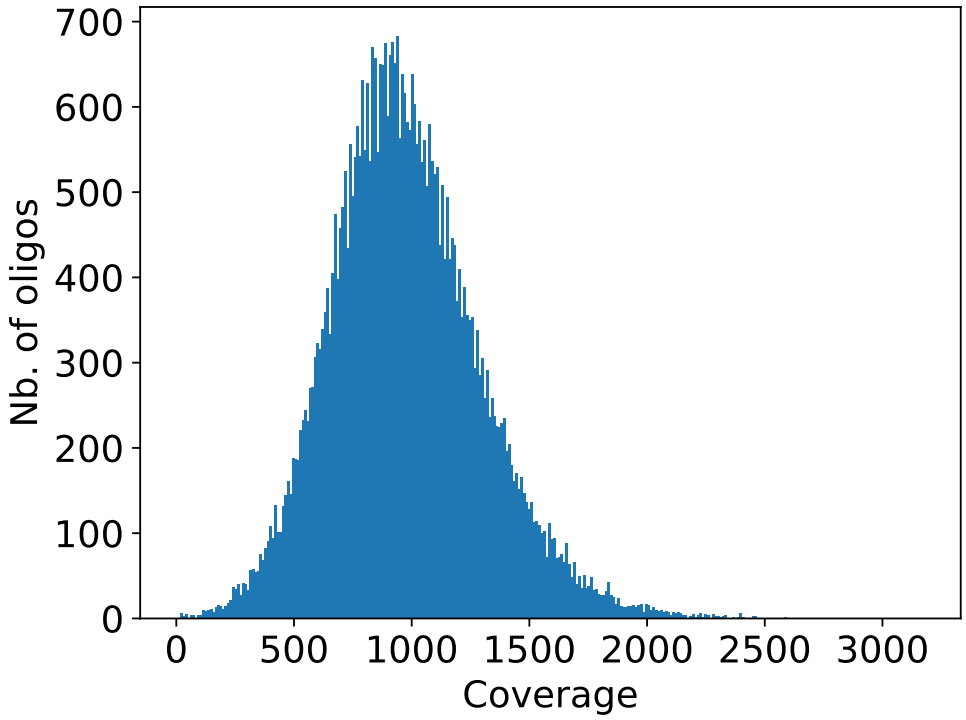

Figure 4.6: Histogram of coverage across oligos in wetlab experiment.

dataset as input to the decoding pipeline. As a result, we were able to achieve full data reconstruction using the entire dataset of sequenced reads. 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 read datasets which were derived by randomly sampling a fraction of reads from the 43M read dataset. In doing so, we found that OA-DSM 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 columnar decoding were able to successfully recover data. As further reduction in coverage led to data loss, we validate 4× as the minimum coverage OA-DSM can handle with our wetlab experiment.

4.4.2 SOTA comparison

Having discussed the results from our real-world wetlab experiments, we will now present a comparison of OA-DSM with SOTA approaches in terms of reading and writing cost [36], [62], [63]. Writing cost is defined as $\frac{\#nts-in-oligos}{\#bits}$,

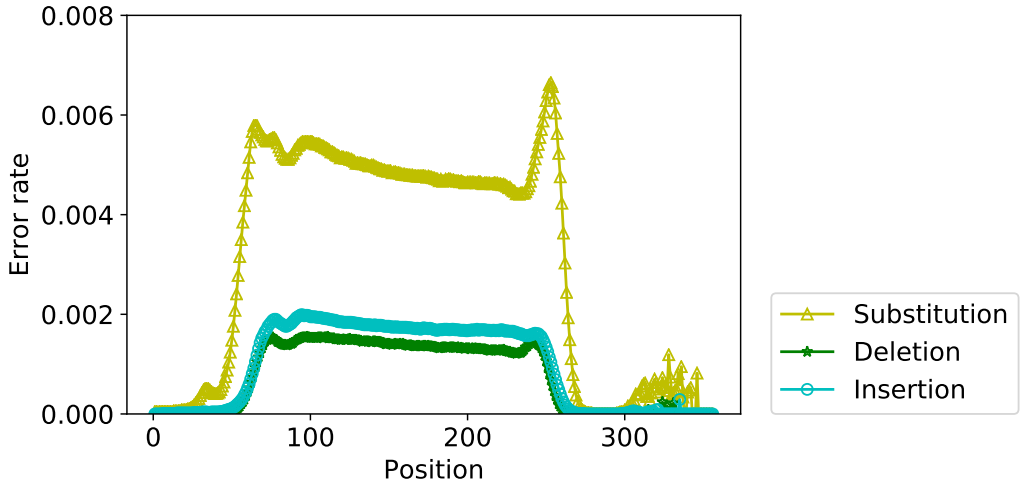

Figure 4.7: Substitution, insertion and deletion rate per position.

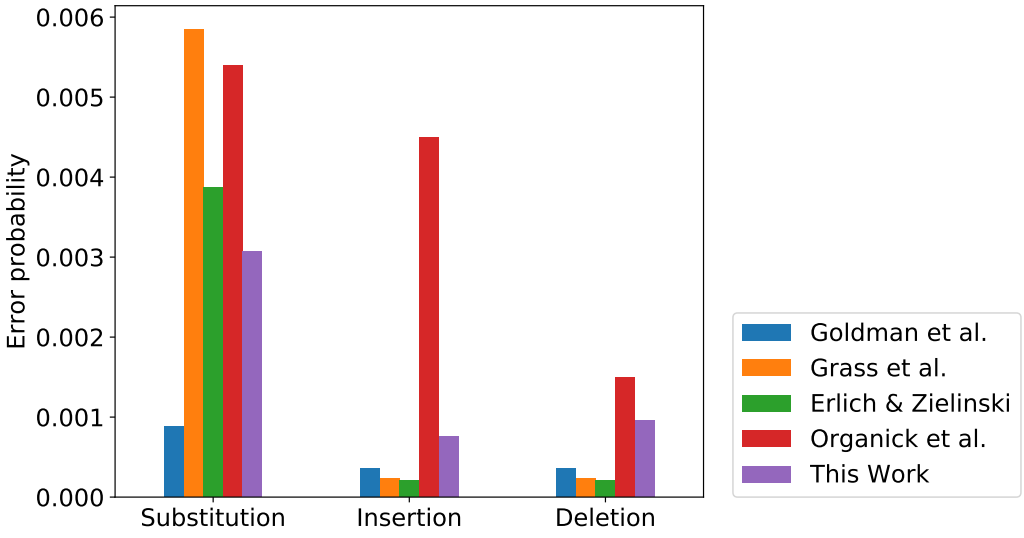

Figure 4.8: Comparison of errors with previous work.

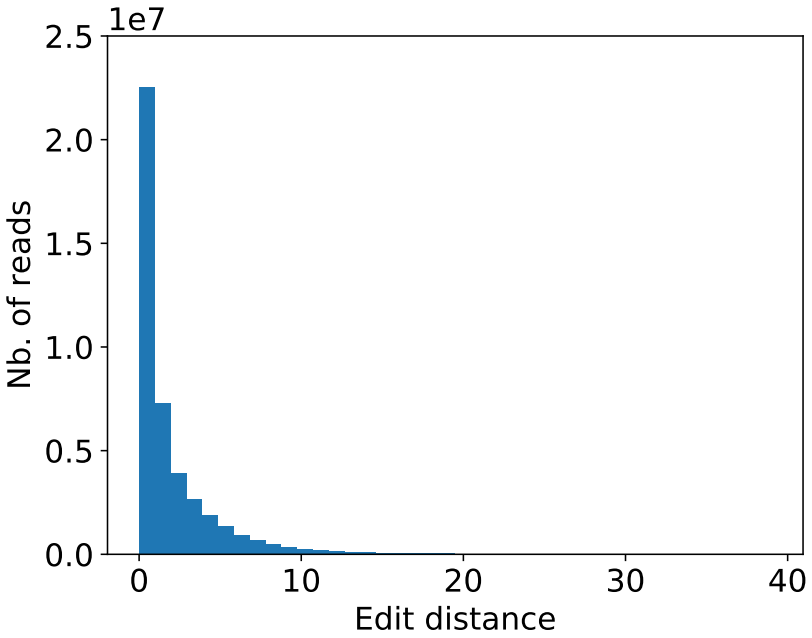

Figure 4.9: Distribution of edit distance

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.

Figure 4.10 shows the read and write cost for OA-DSM and other SOTA algorithms. For OA-DSM, we compute these costs based on the wet-lab experiment presented in the previous section, where for write cost we used 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. As a result, we get a read cost of 2.82 nts/bit (considering 4x as min coverage), and a write cost of 0.70 nts/bit. For SOTA approaches, we reproduce the costs from their publications. There are several observations to be made. First, let us compare OA-DSM with horizontal SOTA approach that also uses LDPC (by S. Chandak et al. [36]). 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 OA-DSM 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

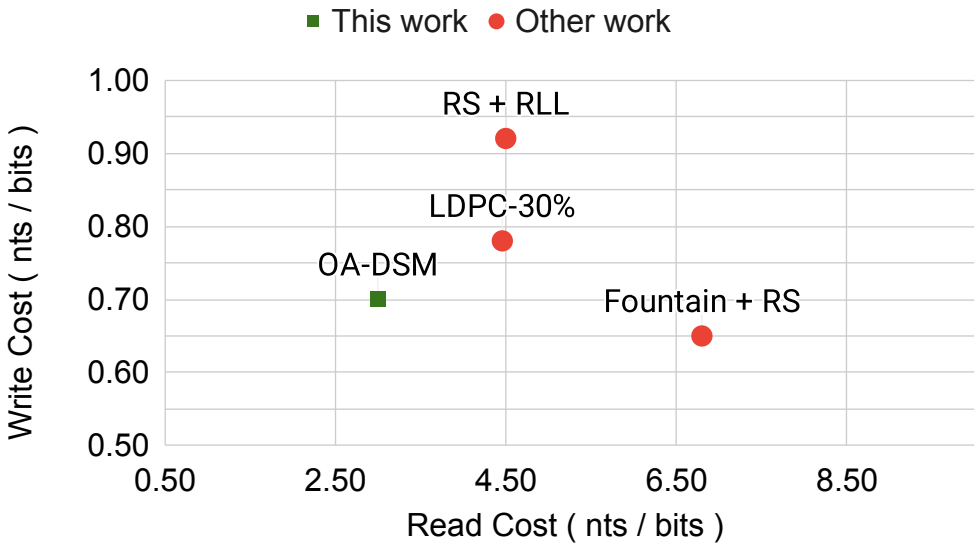

Figure 4.10: OA-DSM vs. SOTA rd/wt costs: RS-RL (Organick et al. [62]), LDPC (Chandak et al. [36]), Fountain+RS (Erlich and Zielinski [69]).

authors also added additional redundancy in each oligo in the form of markers which they used in their decoder. OA-DSM 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.

Comparing OA-DSM with the other two efficient encoders (large-block Reed-Solomon coding by Organick et al. [62] and fountain codes by Erlich et al. [69]), we see that OA-DSM provides substantially better read cost, but slightly worse write cost than fountain coding approach. As we mentioned earlier, we can further improve the write cost for OA-DSM using several approaches. First, the OA-DSM results in Table 6.3 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 4.12). Second, as mentioned in Section 4.3, 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 open these optimizations to future work.

Finally, Lin et al. [63] recently presented the Gini architecture which interleaves nucleotides across oligos in order to minimize the impact of consensus errors. We also tried to compare OA-DSM with Gini, but we could not derive the read/write cost for Gini, which was also not reported, due to lack of

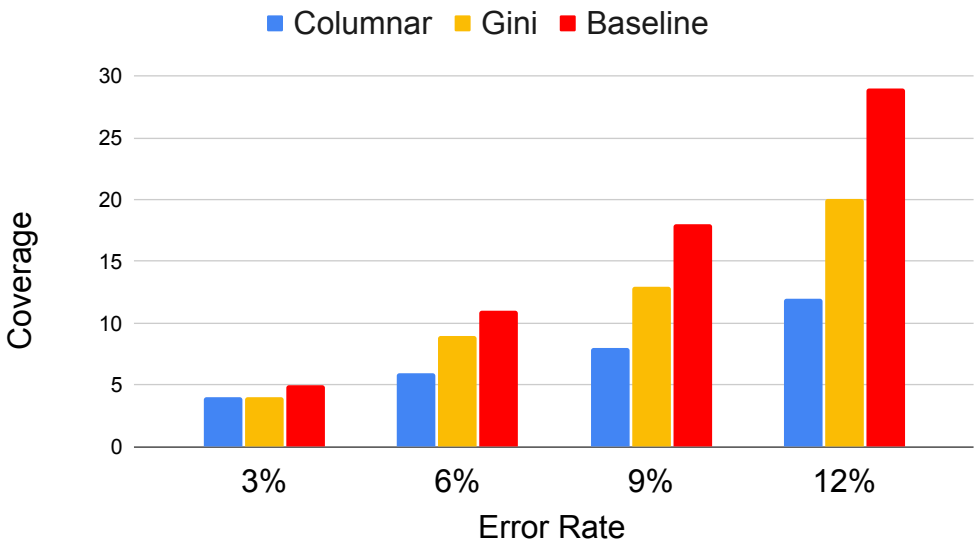

Figure 4.11: Comparison to Gini.

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 4.11 shows the minimum coverage required by OA-DSM, Gini, and a baseline without Gini reported by Lin et al. [63], to perfectly recover data at various error rates. At 18.4% redundancy based on Reed-Solomon coding, the reported baseline needed 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. OA-DSM configured at 30% redundancy with LDPC encoding provides a 40% improvement over Gini, it requires only 12× coverage. Comparing Figure 4.11 with Figure 4.12, we see that OA-DSM provides 25% less coverage (15×) even at 10% redundancy compared to Gini. Thus, OA-DSM has a much lower read cost, thanks to the integrated consensus and decoding enabled by vertical organization.

4.4.3 Benefits of Vertical Design

In order to ensure that the benefits of OA-DSM are due to the vertical design and not other parameters, we have developed a horizontal version of the pipeline shown in Figure 4.4, 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 OA-DSM encoder with horizontal encoder that maps one LDPC block to multiple oligos, (ii) perform consensus to infer entire oligos first, and then decode separately.

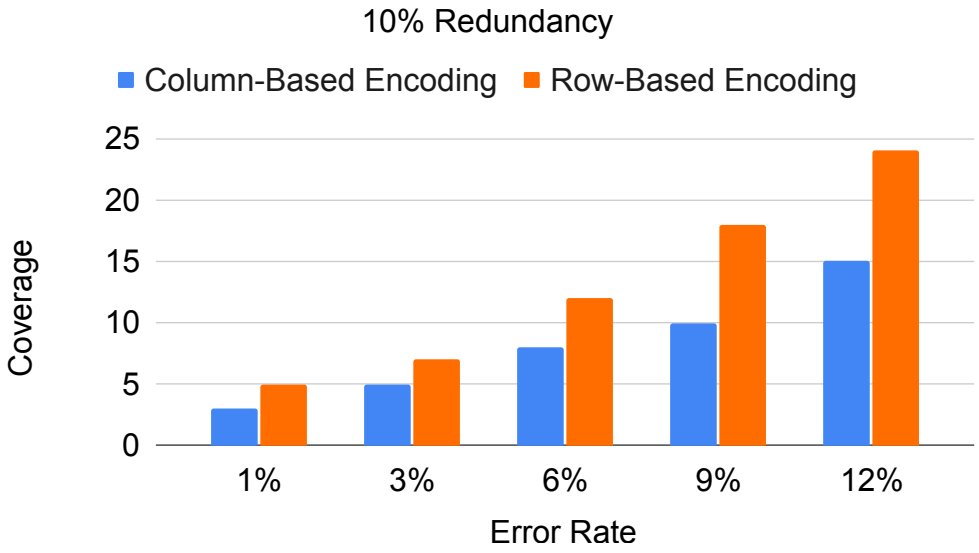

Figure 4.12: Min. cov. at 10% redundancy.

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%). We conduct the experiment similar to SOTA [36], [63] as follows. For each error rate, and for each of the four oligo sets, we generate read datasets at various coverage levels (1× to 25×). In order to generate reads, we first duplicate each oligo a certain number of times according to the configured coverage level. Then we inject random errors at random positions in each read. We inject insertion, deletion and substitution with an equal probability, and 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.

Figure 4.12 shows the minimum coverage for data encoded with 10%

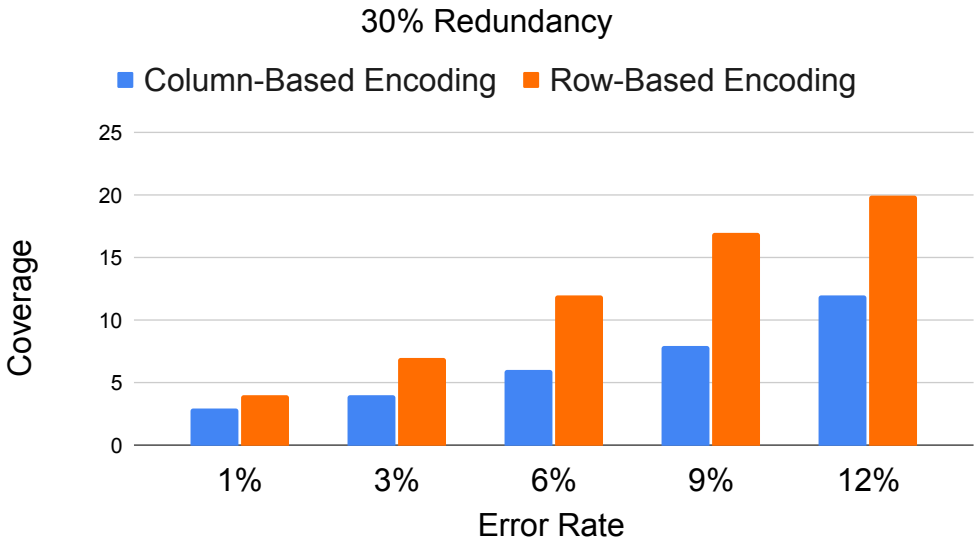

Figure 4.13: Min. cov. at 30% redundancy.

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 $\times$ 30 bits per motif). As explained in Section 4.3, 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 OA-DSM 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 4.13. Notice that in the 30% case, both horizontal and OA-DSM pipelines have a minimum coverage lower than the 10% redundancy case. This is expected, as a higher redundancy implies a higher tolerance to errors.

4.5 Conclusion

All SOTA approaches for DNA data archival use a "row-based" approach (horizontal layout) for mapping input bits onto oligos. In this chapter, we showed how this approach results in a strict separation of consensus calling

and decoding, and how this separation, in turn, results in lost opportunity for improving read/write cost. We presented OA-DSM, an end-to-end pipeline for DNA data archival that uses a novel, database-inspired, columnar data organization. We showed how such an approach enables the integration of consensus and decoding stages so that errors fixed by decoding can improve consensus and vice versa. Using a full system evaluation, we highlighted the benefit of our design and showed that OA-DSM can substantially reduce read-write costs compared to SOTA approaches. As we mention in Section 4.3, internally OA-DSM rely on LDCP for error correction. It has been proved that providing the LDCP decoder with Loglikelihood Ratio (LLR) we can improve its error-correction capabilites. In the next chapter we propose three heuristics used to compute the LLR in the context of a motif-based DNA storage system.

Chapter 5

LLR Estimation for Motif-Based DNA Storage Systems

5.1 Introduction

Despite the advances, high read/write costs and low throughput represent a significant obstacle to DNA's adoption as a storage medium. Over the past few years, researchers have proposed new motif-based approaches to DNA data storage [71], [72] as a potential solution. Instead of using nts (A,C,G,T) as building blocks, these solutions use motifs, which are short oligonucleotide sequences that are drawn from a fixed library, as building blocks for assembling longer oligos. Using motifs as building blocks, one can scale logical density (the number of bits written per synthesis cycle) by storing $log_2(M)$ data bits per synthesis cycle [72]. The use of a fixed library of motifs similar to a typesetting press can simplify miniaturization and automation [73]. Further, new chemical synthesis methods (enzymatic assembly/ligation [20]) can be used to assemble oligos from motifs. While all these aspects are expected to contribute to reduce write cost and improving throughput, they do so at the expense of increased error rate in the synthesized oligos. Thus, effective error management is important for motif-based DNA storage to realize its potential.

In this chapter, we focus on the use of soft information for improving error correction capacity of our OA-DSM. OA-DSM uses large-block-length, low-density parity-check (LDPC) codes to provide resilience against DNA storage errors. Prior work [36] in non-motif-based DNA storage has demonstrated that providing soft information, such as Log-Likelihood Ratio (LLR), as input to LDPC decoders can substantially improve their error-correction capability. However, as we describe later, the use of motifs as building blocks and the

columnar organization in OA-DSM requires a completely different approach to calculate soft information. In this work, we present a few estimation techniques, empirically validate them using real-world wet-lab experiments.

5.2 System model

As mentioned in previous chapters, SOTA pipelines encode data on DNA following the same basic steps, although they differ in their implementation. Data is written to DNA by encoding bits into sequences of nts. This encoding is internally a two-step process. First, data is grouped into blocks of bits that are fed as input to an error-control coding module to produce parity bits. The data and parity bits together form a unit of recovery. The bits belonging to the block are mapped to nts. Due to limitations in synthesizing DNA, the length of a single DNA strand is limited to a few hundred nts at best. Thus, each block of data and parity is converted into a set of DNA sequences. These sequences are synthesized chemically to form actual DNA strands, the oligos.

Reading the data back involves sequencing the DNA to produce reads. In the ideal case, sequencing would produce one read per oligo created during the writing phase. However, due to errors in sequencing and synthesis, reads are noisy duplicates of the starting sequences as they can contain substitution, insertion, and deletion errors. The number of reads corresponding to each oligo (also called *coverage*) can also vary by an order of magnitude, with some oligos being covered by thousands of reads, and others completely missing. In order to recover the original data from this noisy dataset, a consensus step is performed first to infer original sequences from noisy reads. The inferred sequences produced by the consensus step are then passed to the error-control decoder for conversion back from nts into bits. It is important to note here that these consensus sequences need not to be error-free, accurate reproductions of original oligos. It is the job of the error-control decoder to use the additional parity bits to recover the original input data despite these errors. Thus, in all SOTA techniques, the decoding and consensus stages are independent of each other. This is an inherent consequence of how oligos are encoded. When a block of data is mapped to a group of oligos, that group becomes a unit of recovery that must be reassembled entirely (through consensus) before decoding.

As we show in Chapter 4, OA-DSM differs from the SOTA approaches by using a library of motifs as building blocks during encoding rather than individual nts. In OA-DSM, we exploit this motif design and the columnar layout of oligos to iteratively perform consensus and decoding in an integrated fashion. Unlike other approaches, OA-DSM processes the reads one column

at a time. Thus, the first step is columnar consensus using a bit-wise majority algorithm [53], which takes as input the set of reads and produces one column of motifs. These motifs are fed to the *oligo-decoder* which maps the motifs into their 30-bit values using the same array of motifs used during encoding, which does not need to be transmitted but can be generated prior to decoding. Note here that despite consensus, the inferred motifs can still have errors. These wrong motifs will result in wrong 30-bit values. These errors are fixed by the standard BP *LDPC-decoder*, which takes as input the 256,000 bits corresponding to one LDPC block and produces as output the error-corrected input bits. OA-DSM uses the motif library to perform encoding and decoding of data column-by-column instead of oligo by oligo.

5.2.1 Improving OA-DSM with Soft Information

The error correction capabilities of LDPC code can be improved by feeding the LDPC decoder with soft information such as the LLR. However, the calculation of LLR depends on the channel model, which presents a challenge for DNA storage channels due to the lack of an exact channel model with a specific distribution. To address this issue, Chandak et al. [36] proposed a simplified method for LLR computation that is defined under the assumption that DNA storage mimics a binary symmetric channel with error probability epsilon and an ideal Poisson random sampling model. Under this assumption, the transition probability for the channel is given as:

$$p\left((k_0,k_1)\,|0\right) = \frac{e^{-\lambda}\lambda^{k_0,k_1}}{(k_0,k_1)!}\begin{pmatrix} k_0+k_1 \\ k_0 \end{pmatrix}(1-\epsilon)^{k_0}\,\epsilon^{k_1} \tag{5.1}$$

$$p\left((k_0,k_1)\,|1\right) = \frac{e^{-\lambda}\lambda^{k_0,k_1}}{(k_0,k_1)!}\begin{pmatrix} k_0+k_1 \\ k_1 \end{pmatrix}(1-\epsilon)^{k_1}\,\epsilon^{k_0} \tag{5.2}$$

where λ is the ratio between reading cost and writing cost, the input of the channel is a bit, the output is a tuple (k_0, k_1) where k_b is the number of times that the bit is read as b. From this, the LLR is computed as:

$$\log\frac{p\left((k_0,k_1)\,|0\right)}{p\left((k_0,k_1)\,|1\right)} = (k_0-k_1)\log\frac{1-\epsilon}{\epsilon} \tag{5.3}$$

Chandak et al. [36] use nts as the building blocks of encoding with a direct mapping between two bits and four nts (A-00, C-01, G-10, T-11). During the consensus stage of the read pipeline, they perform read clustering to group similar reads into buckets such that each bucket corresponds to an original sequence. Then, for each position, they count the number of occurrences of

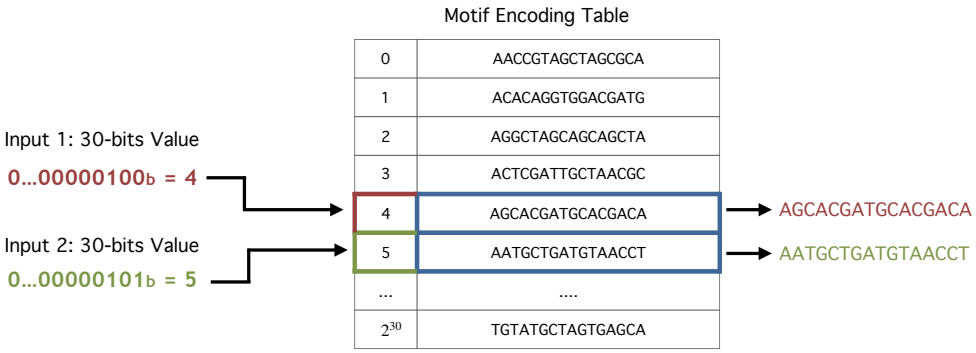

Figure 5.1: Mapping on bits to nucleotides in OA-DSM.

nts, and use it to directly determine k_0 and k_1 values for the consensus LLR. For example, to determine the LLR for the first bit, the count of As and Cs is used as k_0 as they would result in first bit being 0, and the count Gs and Ts is used as k_1 as they would result in first bit being 1. Chandak et al. [36] showed that such an LLR computation improves the read/write cost of DNA data storage, as it allowed LDPC decoders to tolerate more errors while using fewer parity bits.

Unfortunately, this approach cannot be directly applied to OA-DSM; instead of mapping two bits to 1 nt like Chandak et al. [36], OA-DSM maps 30 bits to 16 nts using an associative array. Thus, using a per-position nt count to compute per-bit k_0 and k_1 is not correct for OA-DSM. Figure 5.1 illustrates an example of motif-based bits-to-nucleotides mapping in OA-DSM. The process of mapping bits to nucleotides involves using an associative array that represents the entire space of all possible nucleotide combinations for a given motif length, filtered to exclude sequences that don't respect certain biological constraints. This results in two entries in the array being completely different, thereby losing the direct relationship between bits and nucleotides. In the example, we aim to encode two values expressed in 30 bits, corresponding to 4 and 5 in base 10. As shown in the table, although the two values differ by only one bit, the resulting motifs differ significantly in many nucleotides. Furthermore, there is no correspondence between a pair of bits in the 30-bit values and the nucleotides of the corresponding motif. The straightforward extension of Chandak et al. [36]'s approach to motif design is to extract motifs from the reads, identify which motifs contribute to zero and one bits for each position, and use their count for k_0 and k_1. Unfortunately,

this approach does not work well due to two problems. First, it causes error amplification. When nts are used as building blocks, an error in a single nt will result in the corresponding two bits being wrong. However, with OA-DSM, a single error can result in a completely different 30-bit pattern compared to the correct motif. Thus, using the wrong motif to derive a count of zeroes and ones will result in poor LLR. Second, this approach is slow, as it requires tens of billions of array lookups to convert each motif in each read into its bit sequence.

Given these issues, we implemented three heuristic approaches in OA-DSM for approximating LLR in a scalable fashion. Our first heuristic exploits the fact that we apply a consensus procedure to infer each motif in the read pipeline. When each inferred motif is converted back into bits, we apply Equation 5.3 for each bit, where the difference $\alpha = k_0 - k_1$, is set to 1 or -1. The intuition behind this is that we view the majority consensus output as the only bits emerging out of the DNA storage channel; a zero bit results in k_0 being 1 and k_1 being 0, and a one bit results in k_1 being 0 and k_0 being 1. Thus, α is ± 1. Although this definition works in practice (as we will show in Section 5.3), it does not provide any additional information about the reliability of the bits given by consensus.

Our second heuristic extends the first one by taking each consensus motif, counting the number of reads that exactly with it, and using the count as α. The intuition behind this heuristic is that in the general case, with sufficient coverage, the count of "correct" motifs will be higher than the count of wrong motifs. In such a case, the number of correct motifs can be used as k_0 for a zero bit, and k_1 for a one bit. As majority consensus will identify the correct motif, we can get this count by comparing the consensus motif with every read. Thus, α is $\pm$(exact match count). The drawback of this method is that it requires motifs in reads to exactly match the consensus motif. However, in cases of high error rates, it is possible that the consensus produces the correct motif, but the consensus result does not match even a single read.

Our third heuristic extends the second one by exploiting the fact that OA-DSM uses a bitwise majority consensus algorithm that internally aligns motifs to compute consensus motifs. Alignment is done to deal with insertion and deletion errors that can make motifs in a read longer or shorter than the original motif. Such an alignment also produces the edit distance, or Levenshtein distance, between motifs. Thus, our third heuristic uses this information by changing the exact match count to the count of motifs having an edit distance lower or equal to a configurable threshold. We set this threshold to 15% to tolerate a sufficiently high error rate.

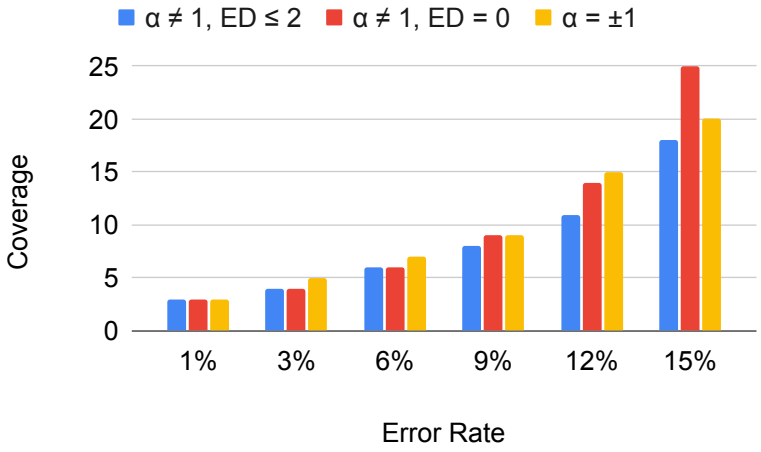

Figure 5.2: Figure shows the minimum coverage required to guarantee 100% data recovery of various LLR methods under a range of error rates.

5.3 Evaluation

In this section, we present the experimental results to highlight the effectiveness of using soft information in OA-DSM. First, we compare the performance of our columnar decoding with respect to various LLR heuristics. Then, we validate the end-to-end encoding and decoding pipeline by presenting the results of our wet-lab experiment. Finally, we compare OA-DSM to other SOTA approaches with respect to read/write costs. All experiments were conducted on a server equipped with a 12-core Intel (R) CPU and 128GB of RAM.

To compare the heuristically defined LLRs in Section 5.2.1, we encoded a binary file of size 609KB (a relational database generated by using TPC-H DBGEN utility) into 22,188 oligos of 160 nts each. Using a custom simulator, we constructed from this set of oligos multiple simulated reads datasets, by uniformly injecting errors with an error rate ranging from 1% to 15% and varying coverage levels. For each simulated read dataset and heuristic LLR, we conducted multiple runs to determine which LLR approach requires the smallest amount of coverage to decode the entire file. Figure 5.2 reports the minimum coverage that our OA-DSM decoder needs to recover data for different types of errors, depending on the LLR it uses. As can be seen, LLR having $\alpha = \pm 1$ is capable of decoding data. At low error rates, it performs as well as other methods. But at high error rates, it is able to recover data only at a much higher coverage. The exact match LLR (shown as $\alpha \neq 1$ ED=0) marginally outperforms the $\alpha = \pm 1$ LLR for 9% and 12% error rate, but underperforms at 15%. This proves our intuition that at high error rates exact matches might not be found; at 15% error rate, there is an error in

every single motif. While consensus can still recover the original motif, the lack of matches would drive the LLR to zero, resulting in poor performance. On the other hand, the LLR using alignment-derived edit distance (shown as $\alpha \neq 1$, ED ≤ 2) outperforms other LLRs.

Having identified the LLR that allows for the lowest coverage in decoding, we validated OA-DSM (with $\alpha \neq 1$, ED ≤ 2 LLR) using a real wet-lab experiment. We propose here the same wet-lab experiment we showed in Chapter 4, as it was already relying on the best LLR heuristic. In this wet-lab experiment, we encoded a 1.2MB binary file representing a compressed relational database archive with 30% LDPC redundancy to generate 44376 oligos, with each oligo having a length of 160 nts. Twist Biosciences synthesized the oligos, which we sequenced using the Oxford Nanopore PromethION platform, generating approximately 43 million noisy reads. We ran the pipeline with all 43 million reads, corresponding to an average coverage of 951×, and were able to fully reconstruct the original data. In order to prove the OA-DSM's capability to handle lower coverage, we subsampled 200K reads from the original dataset, generating a new dataset with an average coverage of 4×. We found that OA-DSM was able to perform full recovery of the original data despite nearly 3500 oligos being completely missing in the subsampled dataset, that is, not covered by any read. With our wetlab experiment, we validated 4× as the least coverage OA-DSM can manage, as further reduction in coverage resulted in data loss.

We conclude this section by presenting the comparison between OA-DSM and SOTA methods, including LDPC coding by S. Chandak et al. [36], large-block Reed-Solomon coding by Organick et al. [62], and fountain codes by Erlich et al. [69], in terms of reading and writing costs. Writing cost is defined as $\frac{\#nts-in-oligos}{\#bits}$, where the numerator is the product of the number of encoding oligos by their length and the denominator is the input file size expressed in bits. Similarly, reading cost is defined as $\frac{\#nts-in-reads}{\#bits}$, i.e., as the ratio between the sum total of all read lengths and the input size in bits. The higher the redundancy and encoding overhead, the higher the write cost, while the higher the coverage required, the higher the read cost. Table 5.1 compares the read/write cost for OA-DSM and other SOTA approaches. We computed the costs for OA-DSM using the values from the wet-lab experiment. Table 5.1 shows that OA-DSM outperforms other SOTA methods substantially in terms of read cost. The write cost of OA-DSM is slightly higher than that of fountain codes but significantly lower than large-block Reed-Solomon coding. The focus of this work was on soft information for improving decoding performance, and hence the read cost. OA-DSM can achieve further reductions in write cost by reducing redundancy and scaling the motif set to use more motifs. We leave open these optimizations to future work. These results suggest that

	OA-DSM	LDPC-30%	RS+RLL	Fountain+RS
Read Cost	2.82	4.46	4.57	6.43
Write Cost	0.70	0.78	0.91	0.62

Table 5.1: OA-DSM vs. SOTA rd/wt costs: RS-RLL (Organick et al. [62]), LDPC (S. Chandak et al. [36]), Fountain+RS (Erlich and Zielinski [69]).

OA-DSM can achieve a good balance between read and write costs, making it a promising solution for efficient DNA data storage.

5.4 Conclusion

In this work, we provided more details on our work with OA-DSM, a motif-based DNA storage system. Considering the mismatch between the motif-based design used by OA-DSM and the nucleotide-based LLR computation proposed by state-of-the-art methods, we proposed three strategies for computing LLR based on various design aspects of OA-DSM. Using results from simulation studies and real-world wet lab experiments, we demonstrated the ability of soft information to reduce read/write costs in OA-DSM.

In the next chapter, we will present the final component of our DNA storage system — the capability to enable random access to a subset of data extracted from a larger DNA pool.

Chapter 6

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

6.1 Introduction

As traditional storage technologies struggle to keep pace with the exponential growth of data, particularly in the context of archival data, DNA has emerged as a promising medium for long-term storage. DNA presents several advantages over the traditional storage media. It is seven orders of magnitude denser than tape [14] and can store up to 1 Exabyte of data in a cubic millimeter [28]. It is extremely durable and can last several millenia when stored under proper conditions. DNA is read by a process called sequencing, and the sequencing technology used to DNA is decoupled from DNA, the storage medium, itself. Thus, DNA will not suffer from obsolescence issues 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.

Given the high density of DNA, an archive stored in DNA can contain millions to billions of files. However, real-world scenarios often demand access to only a fraction of the information. For instance, clients may request a single table from a database or extract a specific image from a collection. Sequencing the entire stored information in response to these requests is needless, expensive, and time consuming, as decoding the information stored in DNA-based storage involves applying a computationally-intensive consensus calling procedure to aggregate reads originating from the same oligo. Thus, the implementation of reliable random access is crucial in making large-scale,

cost-efficient DNA-based storage feasible.

In this chapter, we present *Columnar MOlecular Storage Systems (CMOSS)*, an end-to-end pipeline for DNA storage that provides substantially lower read/write costs than SOTA approaches. Our work is orthogonal to the growing literature on designing optimal error-correction codes for DNA storage in that CMOSS can be used with any error-control code. Built on top of OligoArchive-DSM, that we presented in Chapter 4, its key aspects that distinguish it from SOTA approaches are: (i) a motif-based, vertical, cross-oligo layout for DNA storage in contrast to the nucleotide-based, horizontal layout used by SOTA, and (ii) an integrated consensus and decoding technique that exploits the novel layout to incrementally recover data at very low sequencing coverage; (iii) a reliable, fixed-size, block-based random access organization for DNA storage instead of a variable-sized, object-based access used by SOTA. While (i) and (ii) are inherited by the OA-DSM design, whose benefits have been widely discussed in Chapter 4, (iii) is novel in CMOSS and enable our storage system with random access on a subset of files stored in a pool of DNA molecules. In developing CMOSS, we make the following contributions.

- Using real data from wet-lab experiments, we perform a quantification of random-access errors in DNA storage. Prior studies have done error quantification in terms of substitution, deletion, and insertion errors present in post-sequenced reads. However, there has been little focus on performing a systematic quantification of the effect of coverage bias introduced by Polymerase Chain Reaction (PCR)–the fundamental procedure used to achieve random access in DNA storage–while amplifying a complex DNA pool storing files of various sizes. We bridge this gap by presenting such an analysis (Section 6.2.1)
- Using CMOSS, we perform simulation studies and one, large-scale wet-lab experiment where we encode MB-sized dataset to generate complex oligo pool. We use these experiments to (i) validate the CMOSS design by ensuring successful data recovery and (ii) perform a novel, systematically 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 the fixed-size, block-based random access organization of CMOSS makes it resilient to errors caused by PCR bias.

6.2 Background

In Chapter 4, we introduced the errors that need to be taken into account by every DNA storage pipeline. We showed that, in order to deal with errors

introduced during synthesis and sequencing, all these pipelines rely on high percentages of redundancy, high sequencing coverage, and consensus calling.

Not every pipeline developed to store digital data on DNA enables random access. When they do, they have to contend with additional errors. In this section, we will provide an overview of the errors that are specific to pipelines supporting random access (Section 6.2.1).

6.2.1 Errors due to Random Access

Having described in Chapter 4 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[62], [74]. The central idea is to associate a distinct pair of short DNA sequence, 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 primer corresponding to the object that is requested.

Prior studies have quantified the nature and frequency of substitution and indel errors introduced by different sequencing technologies and used such quantification to configure the amount of redundancy introduced during encoding/decoding[70]. Studies have also looked at oligo drop outs caused by coverage bias[75]. Coverage bias refers to the fact that after sequencing, original oligonucleotide sequences are covered at very different rates, with some sequences being covered by multiple sequenced reads an others completely missing. Coverage bias is a well-known issue in DNA storage, with both synthesis and PCR contributing to it. During the synthesis, multiple copies of each oligo are created, with the distribution of copies being non uniform. During sequencing, a sample of synthesized DNA is extracted and PCR is used to amplify the DNA. Both samping and the inherent stochasticity of PCR can amplify the pre-existing synthesis bias leading to some oligos being copied in abundance, with others being dropped out. The issue with uneven coverage distribution lies in the inadequately represented sequences, which may not be recoverable during the decoding process because decoders typically require multiple copies of sequences to overcome randomly introduced substitution, insertion, and deletion errors. When certain sequences lack sufficient copies, it can result in the loss of information. However, there has been limited work on systematically quantifying the errors introduced by PCR-based random access [74], [75]. In practical archival scenarios, objects stored tend to have

widely varying sizes. Thus, the impact of PCR issues in the context of a more realistic complex pool requires further study.

In order to understand the impact of PCR induced errors in random access over DNA storage, 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[1], 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 as listed in Table 6.1. 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.

Table 6.1: Table shows the number of oligos and the corresponding database and table primers for each table in SSB, TPCH, and SYN databases.

Table#	Table Primer	Database (DB 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

Figure 6.1: The oligo structure for object based abstraction. UFP: universal forward primer, DBP: database primer, TBLP: table primer, URP: universal reverse primer.

[1]https://www.tpc.org/tpch/

The databases were converted using Goldman et al[67]'s rotational encoding approach to generate 5258 oligonucleotide sequences of 110 base pair (bp) each. The sequence consists of the payload in the middle, flanked by the database primer (DBP), table primer (TBLP), universal forward primer (UFP), and universal reverse primer (URP) on both sides (Figure 6.1). Given the knowledge that the oligo length is constrained to a few hundred nucleotides due to the limitations of SOTA synthesis technology, a longer primer length translates to a reduction in payload length. To enhance the payload weight and information capacity, we utilized short primers of five nucleotides, derived from the Illumina adapter sequences. Table 6.1 shows the table and database primers used and the number of oligos generated for each table.

The oligos designed in **Exp. 1** were synthesized by Twist Bioscience and sequenced using Illumina NovaSeq. Due to the synthesis and sequencing errors, we observed an average error rate of 0.0033 for substitutions, 0.0003 for deletions, and 0.0003 for insertions when comparing the sequenced reads to the original oligos. With such low error rates, we were able to reliably study the coverage bias of PCR-based random access.

Coverage bias refers to the fact that after PCR-amplification and sequencing, original oligonucleotide sequences are covered at very different rates, with some sequences being covered by multiple sequenced reads an others completely missing. The issue with uneven coverage distribution lies in the inadequately represented sequences, which may not be recoverable during the decoding process. Because decoders typically require multiple copies of sequences to overcome randomly introduced substitution, insertion, and deletion errors. When certain sequences lack sufficient copies, it can result in the loss of information and consequently lead to a reduced recall rate of random access.

Quantitative definition of coverage bias. To quantitatively measure the representation of a subset relative to the whole, we use the term "population fraction" [75], referring to the proportion of the data that belongs to a specific object (database or table) within the entire archival dataset. Given n objects, the population fraction 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 a specific object i. Given the original oligonucleotide sequences per table, we can compute the raw population fraction for each object. We refer to the raw population fraction of object i as p_i^r. Similarly, after PCR amplification and sequencing, we can determine the population fraction of the sequenced reads. We refer to this as p_i^s for object i. The ratio of the population fraction after sequencing to the raw population fraction is defined as *population fraction change*. More formally, the population fraction change of object i, denoted as c_i, is computed as $c_i = p_i^r / p_i^s$. Coverage is considered unbiased if and only

if all sequences are equally present as the original objects' distribution. It implies that the mathematical expectation of the population fraction change for all sequences is expected to be one, indicating the absence of coverage bias.

Coverage bias observation in real wet-lab experiment. To investigate whether the data belonging to each database is uniformly distributed after the synthesis and PCR, we employed the universal forward primer to extract all the reads. Next, we aligned the reads to the oligos, using them as a reference to determine the reads' original databases with the sequence aligner tool Accel-Align [64], [65]. The numbers of original oligos and the numbers of sequenced reads belonging to each database are presented in Table 6.2. It illustrates that coverage is uneven and biased, as some databases becoming overrepresented while others become underrepresented after sequencing. To study coverage bias under random access, we employed the universal forward primer in conjunction with a database primer to extract oligos belonging to one particular database. Then, we mapped the reads to oligos using Accel-Align to determine their original tables. Using this alignment, we investigated the population fraction change per table. Figure 6.2 shows the number of oligos and the population fraction change of each table across the three databases. We can clearly see that databases with tables of varying sizes, and hence a varying number of oligos, exhibit a huge variation in population fraction change. For instance, for the SSB database, the tables vary in size from 14 oligos to 2594 oligos. The average population fraction change, which ideally should be 1, is 0.71, with smaller tables being significantly under represented (minimum population fraction change of 0.01), and some being over represented (maximum population fraction change of 1.72). The standard deviation, which ideally should be 0, is 0.73. For the TPCH database, it is even worse, with a minimum population fraction change of 0.42, maximum of 32.56 (significant over representation), average of 6.20, and standard deviation of 10.86. In contrast to these two databases, the simulated SYN database with its uniform table sizes exhibits a more uniform distribution of popular fraction change, with a minimum value of 0.86, maximum of 1.5, average of 1.03, and a standard deviation of 0.19. 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 coverage 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 retieve without substantially higher redundancy to prevent data loss. In contrast, the simulated SYN databases shows that using uniformly-sized units of storage can substantially minimize this bias and ensure a more uniform representation

of oligos.

Table 6.2: The number of oligos (#oligos), raw population fraction (raw pop frac), number of sequenced reads (#reads), population fraction (pop frac) and fraction change (frac change) for SSB, TPCH and SYN databases.

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**

Coverage bias impact factor. To gain a deeper understanding of PCR-induced coverage bias on random access, we employed the universal forward primer in conjunction with a database primer to extract data belonging to a particular database. Then, we determined the original tables of the reads through mapping using Accel-Align, and investigated the population fraction change per table as depicted in Figure 6.2. The results illustrate that smaller-sized objects are more susceptible to coverage bias, showing greater vulnerability to its effects. Conversely, larger tables with uniform sizes demonstrate reduced variation in population fraction change. As an illustration, consider the SYN database, comprising 2422 oligos, and characterized by a uniform distribution across its tables. The mathematical expectation of population fraction change for SYN stands at 1.03, displaying a significantly closer proximity to 1 in contrast to TPCH (0.72) and SSB (10.86).

These observations expose a natural limitation of naively storing objects on DNA and using PCR-based random access without taking into account their sizes. Consequently, such uncertainty can introduce more coverage bias, ultimately affecting the efficiency of random access.

6.3 Design

Having described the coverage bias issue of DNA storage, we now present CMOSS. CMOSS is based on OA-DSM design and extend it to enable random access. For sake of completeness, in this section we present CMOSS design starting from the same concepts introduced in Chapter 4 (i.e., integrated consensus calling and decoding, columnar-based layout and motif-based bits to nucleotides mapping). 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

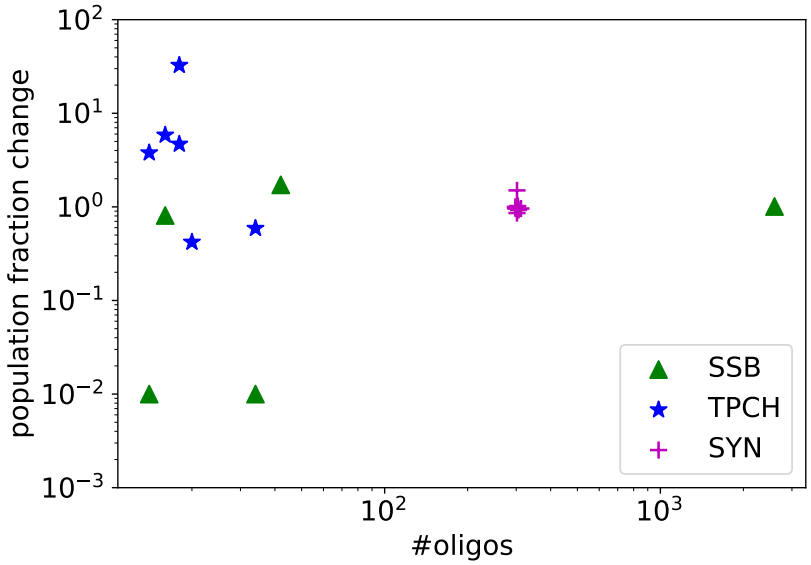

Figure 6.2: The population fraction change (y-axis) and the number of oligos (x-axis) of each table in SSB, TPCH and SYN databases. The '+' purple points are overlapping together because SYN database has 8 tables of uniform size and their population fraction changes are all close to 1.

must be reassembled by consensus, albeit with errors. The key idea in our system is to change the layout from the horizontal, row-style SOTA layout (Figure 6.3(c)) to a vertical, column-style cross-oligo layout (Figure 6.3(d)). Our DNA storage system encodes and decodes data vertically across several oligos instead of horizontally. The key benefits of this are the fact that (i) it 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 low-coverage-bias random access organization where each unit of random access is a fixed size extent instead of a variable-sized object. In the rest of this section, we will explain the design of our system and these advantages in more detail by presenting its read and write pipelines.

6.3.1 Write Pipeline

The top half of the Figure 6.4 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 it into chunks of size 256,000 bits. Each chunk of input is then

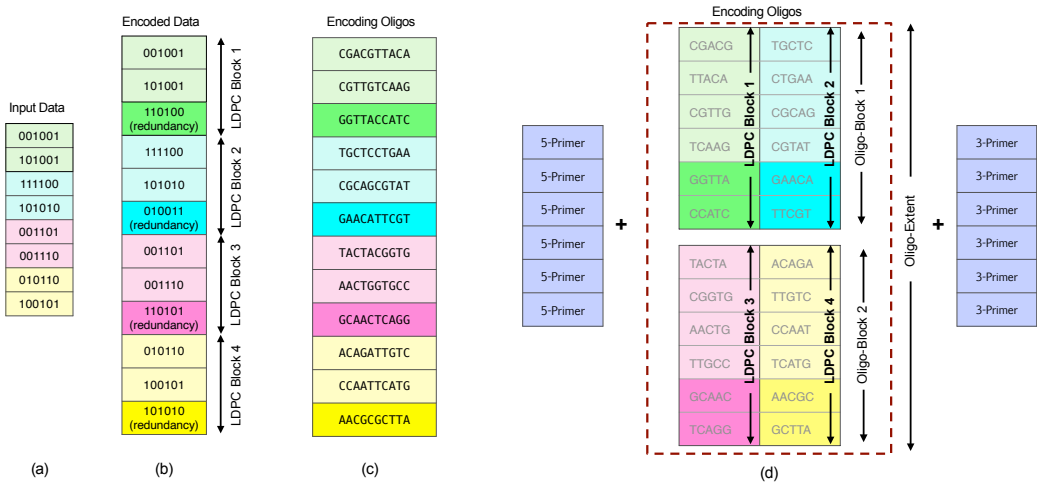

Figure 6.3: Example comparing the mapping layout of SOTA approach to our CMOSS. (a) and (b) are common to both SOTA and CMOSS. (a) The raw input data are grouped into 4 blocks as highlighted with different colors. (b) Each block in the example contains 12 bits and is encoded using LDPC error correcting code, which add 6 parity bits to each block; the resulting block is then split in 3 smaller chunks (2 containing 12 bits of data and 1 containing the parity bits).(c) SOTA encodes each chunk with one encoding oligo. As a result, each LDPC block is mapped to 3 oligos (with the same color in the picture). (d) On the contrary, CMOSS maps each block using a motif-based approach. Every group of 3 bits maps to a motif (short oligo) of 5 nucleotides. As a result, every chunk of 12 bits maps to two motifs, that are disposed vertically to form a column until the full LDPC block is encoded. Every LDPC block mapped into a column of motif is appended to the previous one: for example the blue column mapping the blue LDPC block is appended to the green column mapping the green LDPC block. Once the desired length for the oligo is reached (2 columns in the example), a new group of columns is started (in the picture, the pink and yellow columns). We refer to a column group as Oligo-Block. We call the set of Oligo-Blocks as Oligo-Extent. This organization facilitates the indexing, as every extent is identified by a pair of primers while oligos across oligo-blocks are identified with indexes. Notice that for sake of simplicity the numbers reported in the figure are limited to this specific example. They are customizable, and in the actual design, we use a LDPC blocks containing 256000 bits, a motif length of 16-nts and groups of 30 bits mapping to a motif.

randomized. We use pseudo-randomly generated values in XOR with the bits of each block. The objective is to make the encoding oligos as far as possible

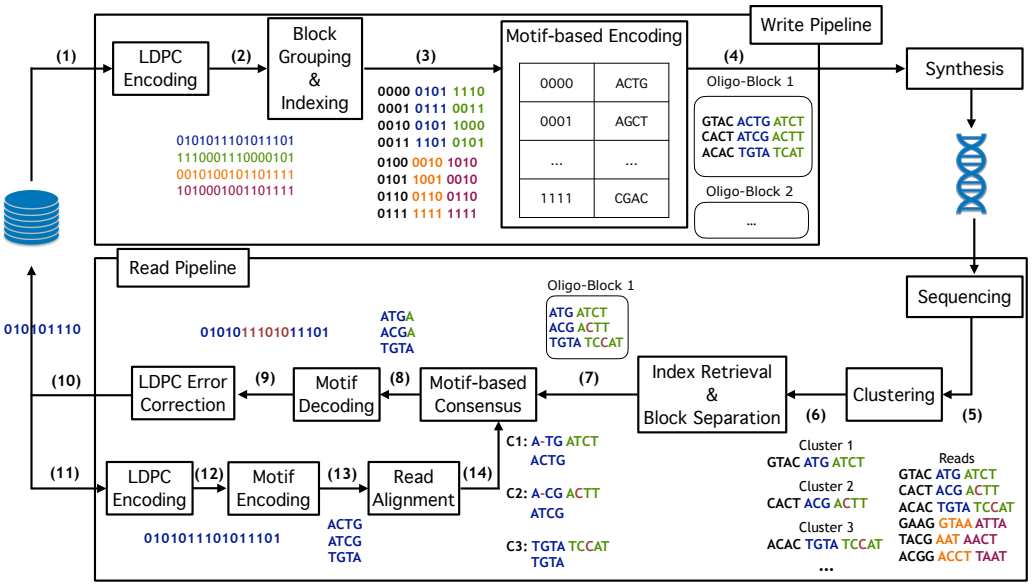

Figure 6.4: Our CMOSS data writing pipeline (top) shows the binary to DNA encoding pathway, and long-term DNA storage in an encapsulated container like Imagene DNAShell™. Our CMOSS reading pipeline (bottom) shows DNA to the binary decoding pathway. For sake of simplicity, the example shown in decoding pipeline refers to one extent only and assumes sequencing coverage 1x.

from each other. This will improve the accuracy of read clustering in the data decoding stage. After randomization, error correction encoding is applied to each chunk to protect the data against errors. Because of the encoding, a chunk will function as 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 as used by Organick et al. [62](65,536 symbols with 16 bits per symbol) in their work on random access in DNA storage. We paramaterize the LDPC code with a chunk size of 256,000 bits, similar to prior work from Chandak et. al. [36], 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 [36]. For the rest of this section, we will only use the LDPC code to discuss the rest of the stages.

The LDPC encoded bit sequence is fed as input to the *oligo-encoder* which converts bits into oligos. While SOTA approaches design each oligo as a random collection of nucleotides, our oligo-encoder designs oligos using composable building blocks called *motifs*. Each motif itself 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 nucleotides as building blocks because, as we will see later in Section 6.3.2, integration of decoding and consensus relies on alignment which cannot be done over single nucleotide.

In order to perform the conversion of bits into motifs, the oligo-encoder maintains an associative array with a 30-bit integer key and a 16 nucleotide-length (nt) 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 0.25 to 0.75. With these constraints, using 16nt motifs, out of 4^{16} possible motifs, we end up with 1,405,798,178 that are valid. By mapping each motif to an integer in the range 0 to $2^{30} - 1$, we can encode 30-bits of data per motif. Thus, at the motif level, the encoding density is 1.875 bits/nt. Under the same biological constraints, we can increase this density by increasing motif size. However, we limited ourselves to this configuration due to two reasons: (i) memory limitation of our current 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 6.3(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 that orders oligos during encoding, and hence enables reordering during decoding.

The second major difference of our approach to SOTA is the layout of motifs that 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 shown in Figure 6.3(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 again from the first column. We refer to such a batch of oligos as a *oligo-block (OB)*. OB is the minimum granularity of decoding in our MOSS pipeline, as all columns (i.e., LDPC blocks) belonging to an OB must be encoded before starting a new OB, and 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 we will see in Section 6.3.2.

In order to scale to large datasets, we designed CMOSS as a flexible,

hierarchical DNA storage system. For this purpose, we group one or more OB into a higher-level structure, termed *oligo-extent (OE)*. If an OB is the unit of encoding and decoding in our MOSS, an OE is the unit of random access. Each OE is designed to be a self-contained, fixed-size (configurable during encoding), and addressable DNA storage partition. Hence, each OE is made randomly addressable by adding a unique pair of primers to the beginning and end of every oligo in all OBs within that OE. All oligos within the OE are addressed using the indexing strategy already 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 OE, which, in turn, can be reduced by simply grouping more OB into a single OE. Second, it makes it possible to pick random access granularity during design time ranging from one OB to multiple OB. For the former, one has to set OE to be the same size as one OB. This would make the number of OE and number of OB identical. Thus, one could divide the list of primers into an equal combinatorial left–right set, and use the left primer to identify an OE and the right primer to identify an OB within each OE. For the latter, one would set the OE to span multiple OB. In this case, the left and right primers would be used to randomly access a full OE. Third, the use of OE as the granularity of random access is effectively equivalent to partitioning the DNA into fixed-sized storage units. As described in Section 6.2.1, the use of fixed-sized, extent-based random access will reduce the impact of PCR coverage bias.

6.3.2 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 insertion, deletion, or substitution errors. Due to the hierarchical structure, decoding begins by first grouping the reads based on their OE. To accomplish 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 OE, considerably 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, but the same procedures are simultaneously applied to all extents. Recall that an OB consists of multiple oligos organized as several columns of motifs. As each oligo can be covered by multiple reads, the first step in decoding is clustering to group related reads together. In prior work, we developed a string clustering solution for this purpose [53]. Our solution uses the CGK randomized embedding algorithm [76] to map

reads into embedded reads such that the hamming distance of embedded reads closely approximates the edit distance of the original reads. Next, our algorithm uses Locality Sensitive Hashing [41] to separate embedded reads into clusters based on hamming distance. Due to randomization during encoding, reads corresponding to the same original oligo are "close" to each other in edit distance despite errors and very "far" from the reads related to other oligos. Thus, the embedded hamming distances will also reflect this disparity. Our algorithm exploits this to avoid computing pair-wise edit distance which is computationally expensive. The output of this algorithm is a set of clusters, each corresponding to some unknown original oligo. While we use our solution, we would like to mention that any other read clustering solution [77], [78] can also be used, and is orthogonal to the work presented here. After the clustering stage, other SOTA methods apply consensus methods in each cluster to infer consensus oligos from reads. This is then followed by decoding using the consensus oligos. During decoding, SOTA methods use error-correction codes to recover from any residual errors that might be present after consensus. Thus, decoding 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 explain this with an example, let us consider the first three oligos (in green) in Figure 6.3(c). Those oligos encode the LDPC-block-1 shown in Figure 6.3(b). In order to decode and correct this block of bits, SOTA approaches need to first perform consensus calling to infer the first three full oligos; then, they can convert the inferred oligos 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 will be only used as part of the final output, that is the reconstructed original input file. In CMOSS, we exploit the motif design and columnar layout of oligos to progressively perform consensus and decoding in an integrated fashion as shown in the bottom part of Figure 6.4. Unlike other approaches, our system processes the reads one column at a time. Thus, the first step is motif-based consensus which takes as input the set of reads and produces the first column of motifs. The choice of consensus algorithm is orthogonal to CMOSS design. We use an alignment-based algorithm that we developed previously for motif-based consensus calling [53]. The algorithm works by taking a motif-length portion of each read belonging to a cluster, aligning them, and taking a position-wise majority of aligned motifs to determine a consensus motif. 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. Note here that despite consensus, the inferred motifs can still have errors. These wrong motifs will result in wrong 30-bit values. These errors are fixed by the *LDPC-decoder*, which takes as input the 30-bit values corresponding to one LDPC block and produces as output the error-corrected, randomized input bits. These input bits are then derandomized to produce the original input bits for that block. Different from SOTA, in CMOSS, these decoded bits are reencoded again by passing them through the LDPC-encoder and the oligo-encoder. This once again produces the correct first column of motifs as it would have been done during encoding in the write pipeline. The correct motifs are then used to realign reads within each cluster so that the next round of decoding for the second column start at the correct offset. This whole process is repeated for all subsequent columns.

The intuition behind this realignment is as follows. An insertion or deletion error in the consensus motifs will not only affect that motif, but also all downstream motifs also due to a variation in length. For instance, if we look at the reads between sequencing stage and block separation stage in Figure 6.4, we see a deletion error in the first read $A-TGATCT$ which should have been $A\mathbf{C}TGATCT$. This results in the first motif being incorrectly interpreted as $ATGA$ (instead of $ACTG$, and second motif as CTG (instead of $ATCT$). As SOTA approaches separate consensus calling and decoding, it is impossible to know the correct motif at the consensus stage. Without a knowledge of the correct motif, there is no way to fix these errors. Thus, an error early in consensus keeps propagating leading to the reliability bias as explained in Chapter 4.2.

On the contrary, thanks to the motif-based vertical layout in CMOSS, there is a way to acquire the knowledge of the right motif at consensus stage. To recap, every column stores a LDPC block, which contains error-correcting bits. Thus, we can get these error-corrected bits as soon as we decode one column of motifs. We can then use these decoded bits to generate the correct column of motifs by rencoding them during decoding. We can use the correct motifs to fix these errors by realigning them against reads. This realignment will determine the position where the 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.

Note here that 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 nucleotides, 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 small refinement to the decoding procedure we have described so far is the special handling required to deal with indexing information for the case when an OE has more than one OB. Recall that OE is the unit of random access, while OB is the unit of decoding. Also recall that we store indexing information in the first column of motifs across the whole OE. Thus, in the case where an OE has more than OB, primers are used to identify OE, while this index is used to indirectly identify the OB of each oligo within an OE. So the decoding of the first column of motifs is special in that it produces this indexing information across all OBs within an OE. The index information is used to separate reads into constitutent OBs and starting from the second column, we switch to per-OB processing. This whole process is illustrated with a simple example in Figure 6.4.

6.4 Evaluation

In this section, we present a thorough evaluation of CMOSS. First, we present results from two wet-lab experiments to study the ability to achieve full data recovery (see Sec. 6.4.1 and Sec.6.4.2). Second, we present a large-scale simulation to validate our end-to-end CMOSS pipeline on an upscaled dataset (see Sec.6.4.3). Third, we demonstrate that the CMOSS design is orthogonal to the adopted error-correcting code, and its benefits lie in the vertical layout combined with integrated consensus calling and decoding, rather than in the specific ECC algorithm. To illustrate this, we implemented a version of CMOSS where we replaced LDPC codes with Reed-Solomon codes (see Sec.6.4.4). Finally, we conclude by providing a comprehensive overview of CMOSS's performance compared to other state-of-the-art methods (see Sec.6.4.5). 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 of the pipeline shown in Figure 6.4 have been implemented in C++17.

6.4.1 Small-Scale Wet-lab Validation

As the first prototype test, we used the TPC-H DBGEN utility to generate a compressed database of 1.2MB which was subsequently encoded by CMOSS, configured with 30% LDPC redundancy, into 44376 oligos of length 160nt partitioned into sixteen OEs. Primers of 20nts where added to each oligos, for a total of 200nt in each oligo. The oligos were synthesized by Twist Biosciences. Subsequently, we sequenced the oligos using the Oxford Nanopore

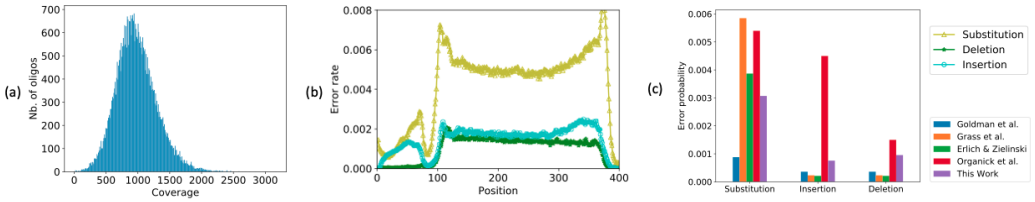

Figure 6.5: Statistics about **Exp. 2** where the input was encoded by CMOSS with 30% LDPC redundancy, the oligos were synthesized by Twist Biosciences and sequenced by the Oxford Nanopore PromethION. (a) The histogram of coverage across oligos. The x-axis is the sequence coverage, y-axis is the number of oligos having that sequence coverage. (b) The substitution, deletion and insertion error rates of each position of the read. (c) Comparison of the overall average substitution, deletion and insertion errors rates with previous work.

PromethION 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 [64] sequence aligner. 99.9999% reads were aligned to a reference oligo, indicating a very high quality of the generated read set. Figure 6.5.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 [66]. The error rate per position is illustrated in Figure 6.5.b. Note that while the data-carrying payload had a length of 160nts, our reads are longer as they include the primers that were appended at both ends of the oligo for sequencing. As these primers 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 6.5.c compares our error rates with those reported in prior work on DNA storage [62], [67]–[70]. We calculated these statistics using raw reads without any quality-based filtering. As can be seen, our reported error rates are lower than those

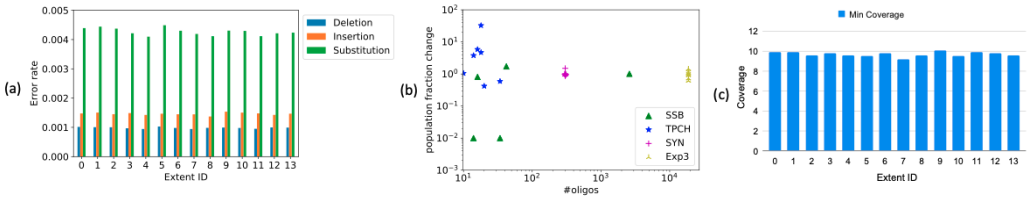

Figure 6.6: Statistics about **Exp. 3** where the input was encoded by CMOSS with 10% LDPC redundancy, the oligos were synthesized by Twist Biosciences and sequenced by the Oxford Nanopore PromethION. (a) The error rate of the reads selected by each extent's primers. (b) The population fraction change (y-axis) and the number of oligos (x-axis) of each extent. The yellow points are overlapping together because **Exp. 3** has 14 extents of uniform size and their population fraction changes are all close to 1. We also plotted those metrics about **Exp. 1** mentioned in Figure 6.2 for comparison. (c) The minimum coverage required for full data reconstruction per extent.

reported in literature. Both array synthesis and Nanopore sequencing have improved in accuracy over the past few years, with Nanopore PromethION platform offering single-read accuracy of over 99% with LSK114 kit [79]. 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 pipline. 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 readsets 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 column-based 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 wetlab 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.

6.4.2 Large-Scale Wet-lab Validation

As a large-scale wet-lab 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 by with just 10% LDPC redundancy, lower than the one adopted in **Exp. 2**. The reason of a lower coverage 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) which were synthesized with Twist Biosciences. To evaluate data recovery per extent, we conducted 14 independent wetlab experiments. Each wetlab 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 6.6.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 6.5.b.

Coverage Bias

As we explained in Section 6.2.1, 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 6.6.b is an extension of Figure 6.2 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 0.7 and 10.8. 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 6.6.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.

6.4.3 Simulation of Large Scale Random Access

In this section we present the performance of our system in accurately retrieving only a small portion a large file of 100GB. We encoded this file using LDPC codes with 30% redundancy. Then, we mapped the resulting bits into a set oligos of 1024 nts each. Based on these parameters, every OB is approximately 4MB in size, resulting in about 26,421 total OB. Every OB comprises 22,188 oligos, and we assume for this experiment that the number of OE matches the number of OB. In this way, a random access to one OE will correspond to a random access one OB, or 4MB. Every OE is then tagged with unique left and right primer sequences, extending the total nucleotide count to 1064 per oligo. The entire file is encoded using nearly 586 million oligos. To simulate real-world conditions, we developed a sampling algorithm that generate simulated reads based on the oligos dataset and corresponding primers of the target extent. The simulation included a similar rate of improper binding as found in earlier wet-lab experiments. In

order to simulate natural errors like substitutions, insertions, and deletions that characterize every sequencing process, we used BBMap tool to introduce random nucleotide variations. We then generated read datasets of different sizes to identify the minimum coverage required to accurately retrieve an OE. As a result, we found that at least 155,052 reads were needed for complete reconstruction. However, nearly 26% of the simulated reads contained errors — either in the primers or the payload—making them unusable for data retrieval. The remaining 74% of reads were successfully used to retrieve the target extent. Subsequent analysis revealed that this corresponds to a 5x coverage rate, which we confirmed as the minimum coverage necessary for accurate extent retrieval. Thus, this experiment validates the capability of CMOSS in isolating and decoding specific OB within larger files.

6.4.4 Code Adaptability Across Diverse ECCs

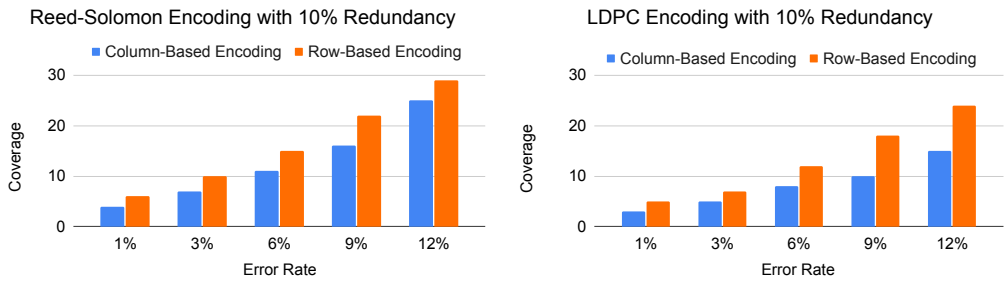

Figure 6.7: Minimum coverage required by our column-based and row-based implementations for decoding a 3MB archive file encoded with RS code (on the left) and LDPC code (on the right) at 10% redundancy. Lower is better.

In the design section we mentioned that this work is orthogonal to current efforts in designing optimal codes. Our core contributions include the columnar 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 implemented RS code with the same block length and symbol size as used in Organick et al. [62](65,536 symbols with 16 bits per symbol) in both columnar-based and row-based 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.[62], 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 (column/row based), we generated read datasets at various coverage levels (from 1× to 25×). As shown in Figure 6.7, the trend of minimum coverage for various error rates is similar to the experiment conducted using LDPC; column-based encoding with its integrated consensus outperforms the row-based 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 columnar layout and integrated consensus aspects of CMOSS design are orthogonal to the choice of error-control codes.

6.4.5 Extended comparison with SOTA

Table 6.3: Comparison of this work with SOTA DNA Storage Methods. The table summarize the information about the wet lab experiments when are available in their publication, such as size of file encoded, number of oligos generated, oligo length, minimum coverage they reported in their publications to fully reconstruct the encoded data. From this information, we computed two metrics for comparisons: write cost, defined as $\frac{\#nts-in-oligos}{\#bits}$ and read cost $\frac{\#nts-in-reads}{\#bits}$.For both the metrics, lower is better.

Reference	Binary Size	Nb of Oligos	Oligo Length (nt)	Recovery Coverage	Write Cost (nt/bit)	Read Cost (nt/bit)
Church et al. [80] (2012)	658 KB	54,898	115	3000	1.17	3513.66
Goldman et al. [67] (2013)	650 KB	153,335	117	51	3.37	171.83
Grass et al. [68] (2015)	85 KB	4,991	117	372	0.84	311.97
Bornholt et al. [81] (2016)	150 KB	16,994	80	40	1.11	44.26
Yazdi et al. [82] (2017)	3.55 KB	17	1000	200	0.58	116.91
Erlich and Zielinski [69] (2017)	2.11 MB	72,000	152	10.4	0.62	6.43
Organick et al. [62] (2018)	200 MB	13,448,372	110/114	5	0.91	4.57
Anavy et al. [83] (2019)	22.5 B	1	42	100	0.23	23.33
Choi et al. [84] (2019)	135.4 KB	4,503	111	250	0.45	112.66
S. Chandak et al. [85] (2019)	192 KB	11,892	n.a.	5	0.78	4.46
THIS WORK (Exp2)	1.2 MB	44376	200	4	0.70	2.82
THIS WORK (Exp3)	13 MB	262,836	280	9.5	0.57	5.49

We conclude this chapter by presenting a broader overview of a comparison of our work with SOTA in terms of reading and writing cost [36], [62], [63]. 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 6.3 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 6.3, 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.

First, comparing CMOSS with row-based SOTA approach that also uses LDPC (by S. Chandak et al. [36]). we can see that even by using the same error-correcting strategy (both use the same LDPC encoder configured with 30% redundancy) the CMOSS approach has both a lower write and read cost. The cost reported here is for around 1% error rate in both cases. The difference in write cost can be explained due to the fact that in the row-based 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.

Comparing CMOSS with large-block RS coding by Organick et al. [62] and fountain codes by Erlich et al. [69], 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 6.3 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 4.12). Second, as mentioned in Section 6.3, 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.

Finally, we would like to mention that there are other SOTA approaches that tried to add to Table 6.3 [47], [61], [86]–[89]. 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 6.3.

6.5 Conclusion

All SOTA approaches for DNA data archival utilize an object-based interface for storing data and a horizontal layout approach for mapping input bits onto oligos. In this chapter, we demonstrated how these assumptions exacerbate PCR coverage bias in a complex pool with files of multiple sizes, leading to a rigid separation of consensus calling and decoding. This separation, in turn, results in a missed opportunity for improving read/write costs. We introduced CMOSS, an end-to-end pipeline for DNA data archival that employs a novel, vertical oligo layout using motifs as building blocks, and a fixed-size, block/extent-based random access over DNA storage. We showed how this approach facilitates random access in a DNA-based storage systems, using both large-scale wet-lab and simulated experiments.

Chapter 7

Future Works and Conclusion

As data keep growing at exponential rate, scaling the archival practices is becoming increasingly important. In this work, we argued that the current tape-driven, migration-based archival method suffers from fundamental problems, which will soon render cost-efficient data archival infeasible. Given recent advances in the design of obsolescence-free storage media, we believe that the time is ripe to investigate the impact of such media on the archival tier of data lakes. In this work, we considered synthetic DNA as a possible solution for achieving migration-free, long-term data archival.

We provided an overview of the collaboration between Project OligoArchive and the Danish National Archive, focusing on the use of DNA to preserve culturally significant digital data. Building on prior work in molecular information storage and digital preservation, we presented a holistic, end-to-end pipeline for preserving both the data and its meaning on DNA. This pipeline employs SIARD-DK to combat format obsolescence, as validated through simulation studies.

Starting with this pipeline implementation, we investigated several optimizations to the encoding and consensus algorithms to support alternative synthesis and sequencing technologies with potentially higher error rates. As a result, we identified room for improvement in SOTA approaches for DNA data archival, which use a 'row-based' approach for mapping input bits onto oligos. We demonstrated how this approach results in a strict separation of consensus calling and decoding, leading to lost opportunities for improving read/write costs. We presented OA-DSM, an end-to-end pipeline for DNA data archival that uses a novel, database-inspired columnar data organization. This approach enables the integration of consensus and decoding stages, allowing errors fixed by decoding to improve consensus, and vice versa.

OA-DSM relies on (i) a first-level encoding based on Low-Density Parity Check (LDPC) to enable error correction during decoding, and (ii) a motif-

based decoding to map bits to nucleotides. LDPC decoding relies on LLR to improve decoding capabilities. Considering the mismatch of the motif-based design used by OA-DSM with the nucleotide-based LLR computation proposed by SOTA methods, we proposed three strategies for computing LLR, based on various design aspects of OA-DSM. Using results from simulation studies and real-world wet lab experiments, we demonstrated the ability of soft information to reduce read/write costs in OA-DSM. Using a full system evaluation, we highlighted the benefits of our design and showed that OA-DSM can substantially reduce read-write costs compared to SOTA approaches. Finally, we addressed the problems related to random access in a DNA-based storage system. All SOTA approaches for DNA data archival use an object-based interface. In this thesis, we showed how these assumptions amplify PCR coverage bias in a complex pool with files of multiple sizes. Thus, we extended OA-DSM to CMOSS, an end-to-end pipeline for DNA data archival that uses a vertical oligo layout based on motifs as building blocks, and a fixed-size, block/extent-based random access over DNA storage. We demonstrated how the fixed-size, block-based random access organization of CMOSS makes it resilient to errors caused by PCR bias while benefiting from an integrated consensus and decoding stage developed in OA-DSM. Using a full system evaluation, we highlighted the benefits of our design and showed that CMOSS can substantially reduce read-write costs compared to SOTA approaches. While our DNA storage system solves the challenge related to archiving data on DNA, there are other challenges that we do not address in this work.

High cost in synthesis and sequencing techniques Given the benefits highlighted in Chapter 1, synthetic DNA is clearly a promising candidate as an archival medium. The main obstacle to the mainstream adoption of DNA as storage medium lies in the high read and write cost and the high access latency that it inherits from the chemical processes used to create DNA strands and to readout information embedded in those molecules. Although the latency to read out data (minutes-hours) from DNA does not represent a major issue if we consider infrequently accessed data (in a long term archival scenarios), we cannot say the same for the read/write cost. The predominant cost in DNA data storage is related to synthesis, the chemical process used to manufacture DNA strands. Industry analyst Robert Carlson has estimated the cost of array synthesis [1] – under the pessimistic assumption that we can store 1 bit per nucleotide – at approximately US$0.0001 per base, amounting to an astonishing US$800 million per terabyte. This figure is significantly higher,

[1] technique that enables the synthesis of many DNA sequences in parallel

by about 7–8 orders of magnitude, compared to the cost of tape storage, which was around US$16 per terabyte in 2016 and has been decreasing by approximately 10% annually [90]. This substantial cost gap underscores the economic challenges of making DNA data storage a viable alternative for large-scale archival purposes. However, the synthesis process – that DNA data storage borrows from life sciences – has different requirements when applied to DNA data storage, as it allows for sacrificing accuracy for a lower cost. For this reason, this gap is expected to narrow due to technological advancements and economies of scale in DNA synthesis, coupled with improvements in efficiency through parallel synthesis and automation. Slightly lower is the cost to read DNA, i.e., the cost associated with sequencing process. According to the National Human Genome Research Institute (NHGRI) [91], the cost of sequencing 1 million of nucleotides in 2022 is around US$0.01, i.e., US$$10^{-8}$ per base. While these costs are expected to decrease independently of their adoption in DNA data storage, since they are fundamental to other research fields, there remain several challenges within the data management community that need addressing to enable DNA as a viable storage medium: (ii) metadata archival and (iii) format obsolescence.

Metadata archival. In order to provide reliable data storage on DNA despite such errors, encoders need to add additional redundancy to data in the form of parity bits generated by using error control coding techniques. Error-control logic often uses additional metadata, like parity check matrices, in order to derive these parity bits from data. As this metadata is required to decode the data back from DNA, it cannot be stored on DNA itself. Typically, with contemporary media like tape, such error-control metadata is stored in the tape reader where the encoding/decoding logic is also implemented. However, as we mentioned earlier, one of the key benefits of DNA–its obsolescence free nature–is due to the separation of the reader (sequencers) from the media itself (DNA). This raises the question of how and where should this auxiliary metadata be stored. One possibility is to store metadata and data separately, with the former on tape and latter on DNA. However, decades of experience from the digital preservation domain argues against this separation and favors self-contained information storage that physically and logically groups together related data and metadata [92]. Thus, it is necessary to develop tiered storage strategies for passive archival of data and metadata.

A large and highly collaborative “DNA storage alliance” has developed around this topic and rapid advances in automation are being investigated for scaling throughput and latency. While there is not a significant data management component involved in scaling throughput, there are several other

research directions that are particularly relevant to the data management community. We outline a few that we are pursuing here.

Format obsolescence. As mentioned several time in this manuscript, DNA solves the media obsolescence problem but not format obsolescence. Museums and archives have long suffered from this problem as culturally significant digital data that is stored in databases cannot be preserved over long duration due to proprietary file formats. To circumvent this issue, the SOTA approach for long-term database preservation is to extract data from databases, convert it into a textual representation based on CSV and XML [34], and archive the text file. Unfortunately, the switch from binary to text leads to severe data bloat and is not suitable for large cloud data lakes. In Chapter 3 we used as solution to archive culturally significant data in the context of the collaboration with the Danish National Archive SIARD-DK, a Danish version of the SIARD open format. However, more work is required to understand the applicability of other open-source, binary file formats like Parquet, Arrow, Deltalake, and Iceberg, as the basis for long-term archival in terms of forward compatibility, conformance to SQL standards, and their ability to archive application logic expressed in stored procedures, SQL queries, and views which provide the context in which data is accessed.

New synthesis techniques and consensus calling. The key bottleneck when it comes to DNA storage cost is the phosphoramidite synthesis chemistry that is used for manufacturing DNA. Recently, innovative solutions have emerged in order to dramatically reduce the writing cost based on enzymatic DNA synthesis [93], [94]. While these techniques have the potential to reduce cost by several orders of magnitude, they are highly error prone. As a result, novel consensus algorithms that can decode oligos from highly noisy reads are required.

Uncertain data management over DNA storage. Unique to DNA storage is the fact that sequencing DNA not only provides reads but also quality scores, also called Phred scores, that represent the probability of each nt in the read being correct. Current research primarily focuses on using DNA as a precise storage media. An interesting avenue of research is developing techniques that can map phred scores back to higher-level constructs, like attributes or tuples, and use them with uncertain data models [95] for answering queries with error estimates. Doing so will transform DNA into an approximate storage medium [96], [97].

Author's Publications

Conferences

[1] E. Marinelli, E. Ghabach, T. Bolbroe, O. S. Sella, T. Heinis, and R. Appuswamy, "DNA4DNA: preserving culturally significant digital data with synthetic DNA," in *Proceedings of the 17th International Conference on Digital Preservation, iPRES 2021, Beijing, China, October 19-22, 2021*, Z. Chen, Ed., 2021.

[2] E. Marinelli, V. Magnone, M.-C. Dumargne, P. Barbry, and R. Appuswamy, "Using soft information to improve error tolerance of motif-based dna storage systems," in *2023 24th International Conference on Digital Signal Processing, (DSP)*, 2023, pp. 1–5.

Journals

[3] E. Marinelli, E. Ghabach, Y. Yan, *et al.*, "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*, 2022, pp. 119–135.

[4] E. Marinelli, Y. Yan, V. Magnone, *et al.*, "Towards migration-free just-in-case data archival for future cloud data lakes," in *Proceedings of the VLDB Endowment*, vol. 16, 2023, pp. 1923–1929.

References

[5] J. G. David Reinsel and J. Rydning, *Data age 2025: the digitization of the world from edge to core.* 2018.

[6] Intel, *Cold Storage in the Cloud: Trends, Challenges, and Solutions*, White Paper.

[7] H. I. Strategies, *Tiered storage takes center stage*, Report, 2015.

[8] M. Perlmutter, *The lost picture show*, `https://tinyurl.com/y9woh4e3`, 2017.

[9] R. Fontana, R. Biskeborn, M. Lantz, and G. Decad, "Tape in the cloud—technology developments and roadmaps supporting 80 tb cartridge capacities," *AIP Advances*, vol. 9, p. 125 222, Dec. 2019.

[10] D. M. Caudle, C. M. Schmitz, and E. J. Weisbrod, "Microform not extinct yet: Results of a long-term microform use study in the digital age," *Library Collections, Acquisitions, and Technical Services*, vol. 37, no. 1, 2013.

[11] PIQL, *Unicef deposits child convention in awa*, `https://www.piql.com/unicef-deposits-child-convention-in-awa/`, 2020.

[12] K. Patiejunas, *Freezing exabytes of data at facebook's cold storage*, 2014.

[13] P. Anderson, R. Black, A. Cerkauskaite, *et al.*, "Glass: A new media for a new era?" In *HotStorage*, 2018.

[14] S. R. Corporation, *2018 semiconductor synthetic biology roadmap*, `https://www.src.org/program/grc/semisynbio/ssb-roadmap-2018-1st-edition_e1004.pdf`, 2018.

[15] D. Clermont, S. Santoni, S. Saker, M. Gomard, E. Gardais, and C. Bizet, "Assessment of dna encapsulation, a new room-temperature dna storage method," *Biopreservation and Biobanking*, vol. 12, no. 3, pp. 176–183, 2014.

[16] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer, "Substantial biases in ultra-short read data sets from high-throughput DNA sequencing," *Nucleic Acids Research*, vol. 36, no. 16, 2008.

[17] Y.-J. Chen, C. N. Takahashi, L. Organick, *et al.*, "Quantifying molecular bias in dna data storage," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[18] *U.S. National Library of Medicine.*

[19] S. L. Beaucage and M. H. Caruthers, "Deoxynucleoside phosphoramidites—a new class of key intermediates for deoxypolynucleotide synthesis," *Tetrahedron Letters*, vol. 22, pp. 1859–1862, 1981.

[20] H. H. Lee, R. Kalhor, N. Goela, J. Bolot, and G. M. Church, "Terminator-free template-independent enzymatic dna synthesis for digital information storage," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[21] S. A. Borodina TA Lehrach H, "Ligation-based synthesis of oligonucleotides with block structure.," *Analitical Biochemistry*, 2003.

[22] J. Clarke, H. Wu, L. Jayasinghe, A. Patel, S. Reid, and H. Bayley, "Continuous base identification for single-molecule nanopore dna sequencing," *Nature Nanotechnology*, vol. 4, no. 4, pp. 265–270, 2009.

[23] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: Ten years of next-generation sequencing technologies," *Nature Reviews Genetics*, vol. 17, pp. 333–351, 2016.

[24] F. Rang, W. Kloosterman, and J. De Ridder, "From squiggle to basepair: Computational approaches for improving nanopore sequencing read accuracy," *Genome Biology*, vol. 19, Jul. 2018.

[25] T. Batu, S. Kannan, S. Khanna, and A. McGregor, "Reconstructing strings from random traces," in *SODA*, 2004.

[26] R. E. Fontana and G. M. Decad, "Mooreâs law realities for recording systems and memory storage components: Hdd, tape, nand, and optical," *AIP Advances*, vol. 8, no. 5, 2018.

[27] SNIA, *100 year archive requirements survey 10 years later*, `https://tinyurl.com/yytsbvmb`, 2017.

[28] G. M. Church, Y. Gao, and S. Kosuri, "Next-Generation Digital Information Storage in DNA," *Science*, vol. 337, no. 6102, 2012.

[29] N. Goldman, P. Bertone, S. Chen, *et al.*, "Toward Practical High-capacity Low-maintenance Storage of Digital Information in Synthesised DNA," *Nature*, vol. 494, 2013.

[30] Y. Erlich and D. Zielinski, "DNA Fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, 2017.

[31] L. Organick, S. D. Ang, Y.-J. Chen, *et al.*, "Random access in large-scale DNA data storage," *Nature Methods*, vol. 11, no. 5, 2014.

[32] B. Shapiro, "Mammoth 2.0: will genome engineering resurrect extinct species?" *Genome Biology*, 2015.

[33] *Digital Preservation Handbook*. Digital Preservation Coalition, 2015.

[34] L. of Congress, *SIARD (Software Independent Archiving of Relational Databases) Version 1.0*, `https://www.loc.gov/preservation/digital/formats/fdd/fdd000426.shtml`, [Online; accessed 28-May-2021], 2015.

[35] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[36] S. Chandak, K. Tatwawadi, B. Lau, *et al.*, "Improved read/write cost tradeoff in dna-based data storage using ldpc codes," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing*, 2019.

[37] E. Marinelli and R. Appuswamy, "Onejoin: Cross-architecture, scalable edit similarity join for dna data storage using oneapi," in *ADMS*, 2021.

[38] E. Ukkonen, "Algorithms for approximate string matching," *Information and Control*, vol. 64, no. 1, pp. 100–118, 1985.

[39] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016, pp. 712–725.

[40] H. Zhang and Q. Zhang, "Embedjoin: Efficient edit similarity joins via embeddings," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 585–594.

[41] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99, 1999, pp. 518–529.

[42] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[43] Y. Yan, N. Chaturvedi, and R. Appuswamy, "Accel-align: A fast sequence mapper and aligner based on the seed-embed-extend method," *BMC Bioinformatics*, Mar. 2021.

[44] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "Cd-hit: Accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[45] R. C. Edgar, "Search and clustering orders of magnitude faster than blast," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[46] E. Bao, T. Jiang, I. Kaloshian, and T. Girke, "Seed: Efficient clustering of next-generation sequences," *Bioinformatics*, vol. 27, no. 18, pp. 2502–2509, 2011.

[47] P. L. Antkowiak, J. Lietard, M. Z. Darestani, *et al.*, "Low cost dna data storage using photolithographic synthesis and advanced information reconstruction and error correction," *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.

[48] E. Zorita, P. Cusco, and G. J. Filion, "Starcode: Sequence clustering based on all-pairs search," *Bioinformatics*, vol. 31, no. 12, pp. 1913–1919, 2015.

[49] B. T. James, B. B. Luczak, and H. Z. Girgis, "Meshclust: An intelligent tool for clustering dna sequences," *Nucleic acids research*, vol. 46, no. 14, e83–e83, 2018.

[50] J. Jeong, S.-J. Park, J.-W. Kim, *et al.*, "Cooperative sequence clustering and decoding for dna storage system with fountain codes," *Bioinformatics*, vol. 37, no. 19, pp. 3136–3143, 2021.

[51] C. Rashtchian, K. Makarychev, M. Z. Rácz, *et al.*, "Clustering billions of reads for dna data storage," in *Advances in Neural Information Processing Systems (NIPS)*, I. Guyon, U. von Luxburg, S. Bengio, *et al.*, Eds., vol. 2017, Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 3360–3371.

[52] G. Qu, Z. Yan, and H. Wu, "Clover: tree structure-based efficient DNA clustering for DNA-based data storage," *Briefings in Bioinformatics*, vol. 23, no. 5, bbac336, Aug. 2022, ISSN: 1477-4054. eprint: `https://academic.oup.com/bib/article-pdf/23/5/bbac336/45937680/bbac336.pdf`.

[53] E. Marinelli, E. Ghabach, Y. Yan, *et al.*, "Digital preservation with synthetic dna," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems*. 2022.

[54] S. Kannan and A. McGregor, "More on reconstructing strings from random traces: Insertions and deletions," in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 2005, pp. 297–301.

[55] K. Viswanathan and R. Swaminathan, "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*, S. Teng, Ed., SIAM, 2008, pp. 399–408.

[56] A. Krishnamurthy, A. Mazumdar, A. McGregor, and S. Pal, "Trace reconstruction: Generalized and parameterized," *CoRR*, vol. abs/1904.09618, 2019. arXiv: `1904.09618`.

[57] A. Magner, J. Duda, W. Szpankowski, and A. Grama, "Fundamental bounds for sequence reconstruction from nanopore sequencers," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 2, no. 1, pp. 92–106, 2016.

[58] P. S. Gopalan, S. Yekhanin, S. D. Ang, *et al.*, "Trace reconstruction from noisy polynucleotide sequencer reads," 2019.

[59] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin, "Trellis bma: Coded trace reconstruction on ids channels for dna storage," in *2021 IEEE International Symposium on Information Theory (ISIT)*, Melbourne, Australia: IEEE Press, 2021, pp. 2453–2458.

[60] O. Sabary, G. Shapira, E. Yaakobi, and A. Yucovich, *Reconstruction algorithms for dna-storage systems*, US Patent App. 17/447,066, Mar. 2023.

[61] D. Bar-Lev, I. Orr, O. Sabary, T. Etzion, and E. Yaakobi, "Deep dna storage: Scalable and robust dna storage via coding theory and deep learning," *arXiv preprint arXiv:2109.00031*, 2021.

[62] L. Organick, S. D. Ang, Y.-J. Chen, *et al.*, "Random access in large-scale dna data storage," *Nature biotechnology*, vol. 36, no. 3, pp. 242–248, 2018.

[63] D. Lin, Y. Tabatabaee, Y. Pote, and D. Jevdjic, "Managing reliability skew in dna storage," in *ISCA*, 2022.

[64] Y. Yan, N. Chaturvedi, and R. Appuswamy, "Accel-align: A fast sequence mapper and aligner based on the seed–embed–extend method," *BMC bioinformatics*, vol. 22, no. 1, pp. 1–20, 2021.

[65] Y. Yan, N. Chaturvedi, and R. Appuswamy, "Optimizing the accuracy of randomized embedding for sequence alignment," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2022, pp. 144–151.

[66] B. Bushnell, "Bbmap: A fast, accurate, splice-aware aligner," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2014.

[67] N. Goldman, P. Bertone, S. Chen, *et al.*, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[68] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust chemical preservation of digital information on dna in silica with error-correcting codes," *Angewandte Chemie International Edition*, vol. 54, no. 8, pp. 2552–2555, 2015.

[69] Y. Erlich and D. Zielinski, "Dna fountain enables a robust and efficient storage architecture," *science*, vol. 355, no. 6328, pp. 950–954, 2017.

[70] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the dna data storage channel," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[71] N. Roquet, S. P. Bhatia, S. A. Flickinger, *et al.*, "Dna-based data storage via combinatorial assembly," *bioRxiv*, 2021.

[72] Y. Yan, N. Pinnamaneni, S. Chalapati, C. Crosbie, and R. Appuswamy, "Scaling logical density of dna storage with enzymatically-ligated composite motifs," *bioRxiv*, 2023.

[73] *Catalog dna*, `https://www.catalogdna.com`, Accessed: 2022-11-20.

[74] C. Winston, L. Organick, D. Ward, L. Ceze, K. Strauss, and Y.-J. Chen, "Combinatorial pcr method for efficient, selective oligo retrieval from complex oligo pools," *ACS Synthetic Biology*, vol. 11, no. 5, pp. 1727–1734, 2022.

[75] Y.-J. Chen, C. N. Takahashi, L. Organick, *et al.*, "Quantifying molecular bias in dna data storage," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[76] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016, pp. 712–725.

[77] C. Rashtchian, K. Makarychev, M. Rácz, *et al.*, "Clustering billions of reads for dna data storage," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA, 2017, pp. 3362–3373.

[78] T. Shinkar, E. Yaakobi, A. Lenz, and A. Wachter-Zeh, "Clustering-correcting codes," *IEEE Transactions on Information Theory*, vol. 68, no. 3, pp. 1560–1580, 2022.

[79] K. A. Wetterstrand, "Ligation sequencing kit v14, sqk-lsk114," 2024, Accessed: 2024-04-06.

[80] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[81] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A dna-based archival storage system," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 637–649.

[82] S. H. Tabatabaei Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A Rewritable, Random-Access DNA-Based Storage System," *Nature Scientific Reports*, vol. 5, no. 14318, 2015.

[83] L. Anavy, I. Vaknin, O. Atar, R. Amit, and Z. Yakhini, "Data storage in dna with fewer synthesis cycles using composite dna letters," *Nature Biotechnology*, vol. 37, Oct. 2019.

[84] Y. Choi, T. Ryu, A. Lee, *et al.*, "High information capacity dna-based data storage with augmented encoding characters using degenerate bases," *Scientific Reports*, vol. 9, Apr. 2019.

[85] S. Chandak, K. Tatwawadi, B. Lau, *et al.*, "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, 2019, pp. 147–156.

[86] W. Press, J. Hawkins, S. Jones, J. Schaub, and I. Finkelstein, "Hedges error-correcting code for dna storage corrects indels and allows sequence constraints," *Proceedings of the National Academy of Sciences*, vol. 117, p. 202 004 821, Jul. 2020.

[87] K. J. Tomek, K. Volkel, A. Simpson, *et al.*, "Driving the scalability of dna-based information storage systems," *ACS synthetic biology*, vol. 8, no. 6, pp. 1241–1248, 2019.

[88] K. Tomek, K. Volkel, E. Indermaur, J. Tuck, and A. Keung, "Promiscuous molecules for smarter file operations in dna-based data storage," *Nature Communications*, vol. 12, p. 3518, Jun. 2021.

[89] K. Lin, K. Volkel, J. Tuck, and A. Keung, "Dynamic and scalable dna-based information storage," *Nature Communications*, vol. 11, Jun. 2020.

[90] L. Ceze, J. Nivala, and K. Strauss, "Molecular digital data storage using dna," *Nature Reviews Genetics*, May 2019.

[91] K. A. Wetterstrand, "Dna sequencing costs: Data from the nhgri genome sequencing program," *https://www.genome.gov/about-genomics/fact-sheets/ DNA-Sequencing-Costs-Data*,

[92] S. Rabinovici-Cohen, M. Baker, R. Cummings, S. Fineberg, and J. Marberg, "Towards sirf: Self-contained information retention format," May 2011, p. 15.

[93] H. Lee, D. J. Wiegand, K. Griswold, *et al.*, "Photon-directed multiplexed enzymatic dna synthesis for molecular digital data storage," *Nature Communications*, vol. 11, no. 1, 2020.

[94] H. H. Lee, R. Kalhor, N. Goela, J. Bolot, and G. M. Church, "Terminator-free template-independent enzymatic dna synthesis for digital information storage," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[95] C. C. Aggarwal and P. S. Yu, "A survey of uncertain data algorithms and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 5, pp. 609–623, 2009.

[96] C. Bee, Y.-J. Chen, M. Queen, *et al.*, "Molecular-level similarity search brings computing to dna data storage," *Nature Communications*, vol. 12, p. 4764, Aug. 2021.

[97] G. Franzese, Y. Yan, G. Serra, I. D'Onofrio, R. Appuswamy, and P. Michiardi, "Generative dna: Representation learning for dna-based approximate image storage," in *International Conference on Visual Communications and Image Processing*, 2021.