A SNIA Community

DNA Data Storage Codecs - Examples, Requirements, and Metrics

Version 1.0

30-June-2025

Technical White Paper

ABSTRACT: This white paper provides an overview of the state of codecs for DNA data storage, the metrics they should be concerned with, and the important technical attributes which they should incorporate.

USAGE

DISCLAIMER

The information contained in this publication is subject to change without notice. SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Suggestions for revisions should be directed to https://www.snia.org/feedback/.

Table of Contents

1 Overview

DNA is emerging as a promising media type for storing data that has the potential to address fundamental challenges in data storage around retention, longevity, durability, power consumption, and cooling requirements. Alongside such promise, multiple challenges exist to adequately integrate this radically different type of media.

In modern media storage, data is generally created in binary form and laid out in a manner that is conducive to the substrate, for example magnetic disk, magnetic tape, or NAND flash. Each of the aforementioned media types have been wrapped with a layer of intelligence to ensure that issues encountered in one part of the media can be confined or otherwise replaced with information contained in another to ensure data resiliency.

The process of *encoding* and *decoding* data occurs inside of a *coder-decoder*, also known as a CODEC. The CODEC encapsulates the functions of encoding and decoding, and in so doing, data that is to be written to the media is *encoded* with pieces of information that allow errors to be detected and potentially corrected. This process often involves cyclic redundancy checks (CRC), inclusion of parity bits, and inclusion of redundant chunks of the information (for instance, through erasure coding or other mechanisms). Following encoding, the encoded material is written to the media after any media-specific or format-specific processing that must occur.

On the reverse side, requests to retrieve data are often handled by a controller that is aware of the way data has been written to the media and how the data has been encoded prior to writing to the media. Essentially, reading data requires a process that reverses the process found in the writing process. As data is retrieved from the media, it is decoded, returning it to the original binary form, assuming the data passes the redundancy, error, and parity checks during the decoding process. Should the retrieval of that data fail due to the error detection, the controller is able to (1) localize the errors using the redundancy, (2) infer the error to recover the original data using the redundancy, (3) determine consensus on what the contents should be by examining replica strands, or in the worst case, and (4) alert the system or user retrieving the data of an unrecoverable error.

With DNA's primary use case, as of today, of providing long-term archival storage spanning decades, centuries, and beyond, the CODEC is of paramount importance to ensure data durability and availability.

In order to store data into DNA (See Figure 1 for the flow for data storage in DNA), we need to make small binary chunks of a source file which is then converted into DNA oligos (quaternary format) using an encoding such that errors can be corrected. These DNA oligos are synthesized and stored. When we need the data, these DNA oligos are subject to PCR and then clustering is done and noisy reads are corrected for errors and then finally converted into actual data after decoding. Usually, a 1 GB file gives rise to billions of DNA oligos after encoding so reconstruction and decoding plays an important role for the CODEC. The reader is referred to [10] for a layered

model of DNA storage as it fits into the Open System Interconnection (OSI) model of a modern data storage medium and storage interface.

A recent monograph [18] presents a probabilistic channel model specifically designed to reflect the distinct features of DNA-based data storage systems. This model addresses three key characteristics: (1) information is encoded into a large collection of short DNA strands stored in a disordered manner; (2) these strands are prone to errors arising from synthesis, sequencing, and storage; and (3) retrieval of information is performed by randomly sampling strands from the DNA pool. The work also establishes theoretical bounds on the limits of storing and retrieving data within such systems.

The analysis spans several channel models, such as shuffling and noisy shuffling-sampling channels, to evaluate their influence on achievable storage capacity. Furthermore, it investigates the trade-offs between data density and sequencing complexity and examines the effects of strand degradation and sequence length variability. This in-depth exploration offers critical insights for the development and enhancement of DNA data storage technologies, grounded in principles of information theory.

In Section 2, we describe the state of the art of current CODECs. Section 3 deals with the analysis of requirements such as scalability, performance, compatibility etc. for CODECs and summarizes the basic criteria for DNA codecs.

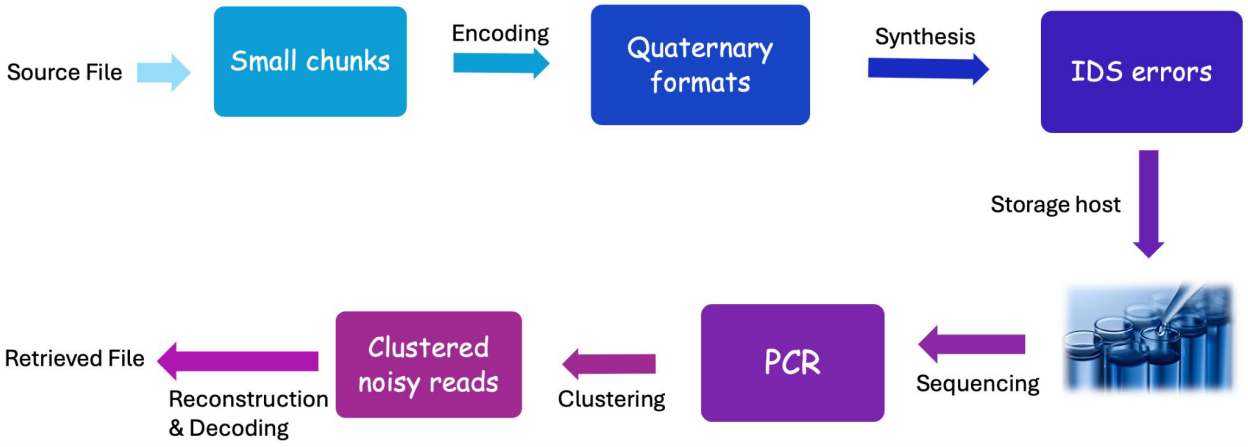


Figure 1: A flow of the DNA Storage Process

2 State of the Art of DNA Codecs

2.1 Types of Codes

There are a number of general information coding fundamentals applied in DNA codes; the following sections outline some of them.

2.1.1 Linear Codes

Codes are useful for error detection and correction for most of the data storage and communication systems. These are of two kinds: linear, where XOR bitwise sum of codewords will lead to another codeword from the set, and nonlinear, where this may not be the case. Often, linear codes have efficient encoding and decoding mechanisms. Thus, linear codes are the most extensively studied type of redundant codes, with several optimal codes known. Consequently, they are frequently utilized in DNA-based data storage systems, often serving as the inner or outer code in dual-layer coding schemes, and occasionally both. However, no efficient linear codes have been discovered that can correct insertions or deletions.

Now we describe two linear codes used in DNA codecs; for more general background on error codes, the reader is referred to the DNA Alliance whitepaper, *DNA Data Storage Technology Review v1.0*.

2.1.1.1 Hamming Codes

A class of binary codes with m parity bits, and a block length of n=2^m-1. Hamming codes guarantee a Hamming distance of 3, which is sufficient to detect and correct a single substitution error. With the addition of a single extra parody bit, Hamming codes can be extended to detect two bit errors. When implementing Hamming codes in DNA one must take care to avoid a single substitution creating multiple bit errors (e.g. if A->T causes 00->11). To prevent this, Hamming codes may be generalized to a 4-character alphabet or a 1 bit per base encoding may be used.

In 2019, [13] Microsoft demonstrated the first prototype of an end to end DNA storage system by encoding and decoding the “hello” word which took ~21 hours and 1 mg of DNA. The codec used in this system was [31, 26] Hamming code over the ring of integers modulo 4 (Z_4) described by the 31 x 5 parity check matrix, as shown here.

$$H = \begin{bmatrix} A \\ I_5 \end{bmatrix},$$

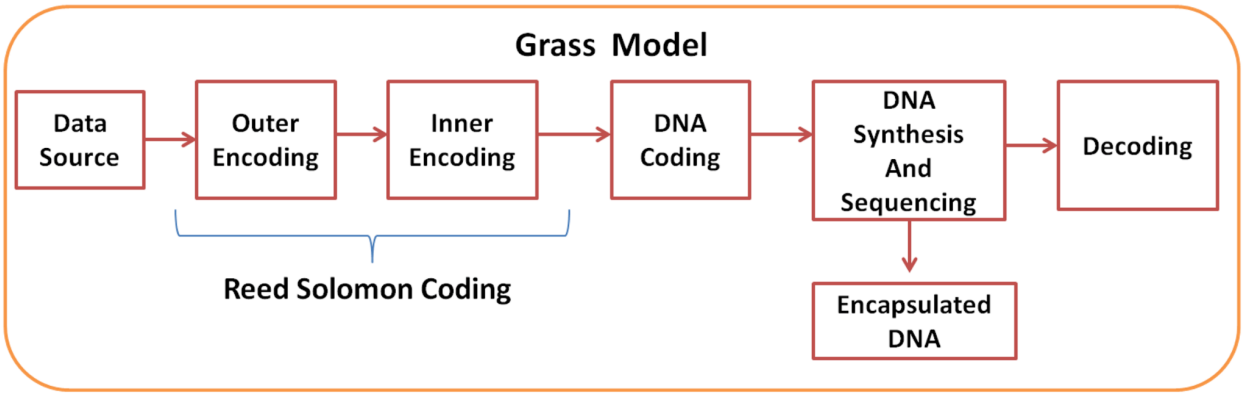
$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

2.1.1.2 Reed Solomon Codes

Reed-Solomon is an optimal linear block code family defined over Galois Fields where the level of redundancy may be tuned for the specific application [47]. RS codes with a block length of n and message length of k may locate and correct up to (n-k)/2 errors or (n-k) erasures (at known locations). Its ability to correct erasures at lower cost than substitution errors make RS especially suited to be applied across strands, where the loss of a single strand would manifest as an erasure rather than a deletion. RS also requires an alphabet of size q=p^m≥n where p is prime and m a

positive integer, meaning that rather than using the alphabet {A,C,G,T} a larger set comprised of groupings of bases is typically used. This constraint does have the benefit that short runs of errors may only be counted as a single error from the point of view of the decoder.

In many earlier DNA Codecs, several variations of Reed Solomon Codes were used. For example, a Reed Solomon code (255,251) over a field with 256 (since we have 256 ASCII symbols) number of elements can be used to correct 2 errors. Figure 2 describes the use of Reed Solomon Code in an early [6] version of DNA based data storage as an outer and inner coding.



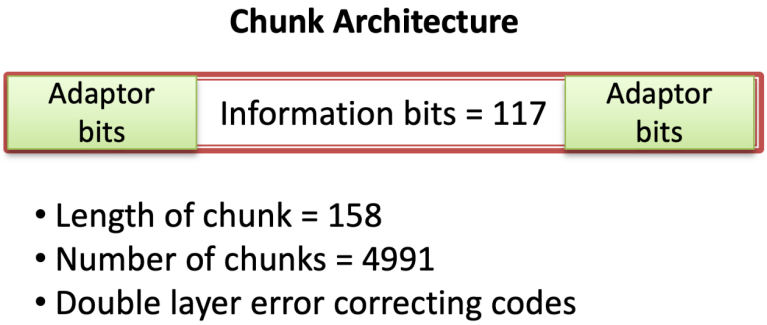


Figure 2: Use of Reed Solomon Codes in the Grass Model [6] of DNA based Storage

There is a similar approach to data storage codecs, closely inspired by conventional archival storage technologies such as magnetic disk and tape systems (i.e., LTO), where Reed-Solomon codes have long been employed in a product code configuration. In these systems, data is protected across two dimensions — rows and columns — allowing robust correction of correlated errors or burst errors that may affect large sections of the medium. This kind of encoding and decoding strategy is the cornerstone of "super data protection" modes, enabling exceptionally low user-visible error rates, often on the order of 10^{-19} [45]. Such reliability standards are critical for long-term archival use cases, and their adaptation into DNA storage reflects the convergence of biological and electronic error correction paradigms. For more background on this method as related to LTO tape, see Annex 1.

2.1.2 DNA Fountain Codes

The DNA Fountain paper by Yaniv Erlich and Dina Zielinski [19] introduces an approach to DNA-based data storage, focusing on maximizing data density and robustness while also introducing error correction.

DNA Fountain employs fountain codes, specifically the Luby Transform (LT), to manage data encoding and dropouts. The Luby Transform is a type of fountain code that enables the creation of numerous short messages (droplets) from the data. Each droplet contains a combination of data segments selected randomly and combined bitwise, ensuring that any sufficiently large subset of droplets can reconstruct the original data.

The encoding process (See Figure 2.1.2) involves several key steps:

1. **Preprocessing:** The input binary file is split into non-overlapping segments.
2. **Luby Transform:** Each segment is used to form droplets. The Luby Transform packages data into any desired number of droplets by selecting a random subset of segments from the file and adding them using a bitwise-XOR operation. Each droplet includes a data payload and a short, fixed-length seed that represents the state of the random number generator during droplet creation.
3. **Droplet Generation:** The Luby Transform iterates over two computational steps: the creation of a droplet and its subsequent screening. The algorithm generates droplets until the required number of valid oligos is achieved.
4. **Screening:** Generated DNA sequences are screened for biochemical constraints, such as acceptable GC content and the absence of long homopolymer runs. Only valid sequences are retained, ensuring robust synthesis and sequencing.

To counteract oligo dropouts and sequencing errors, the architecture introduces 5-10% more oligos than necessary, providing redundancy.

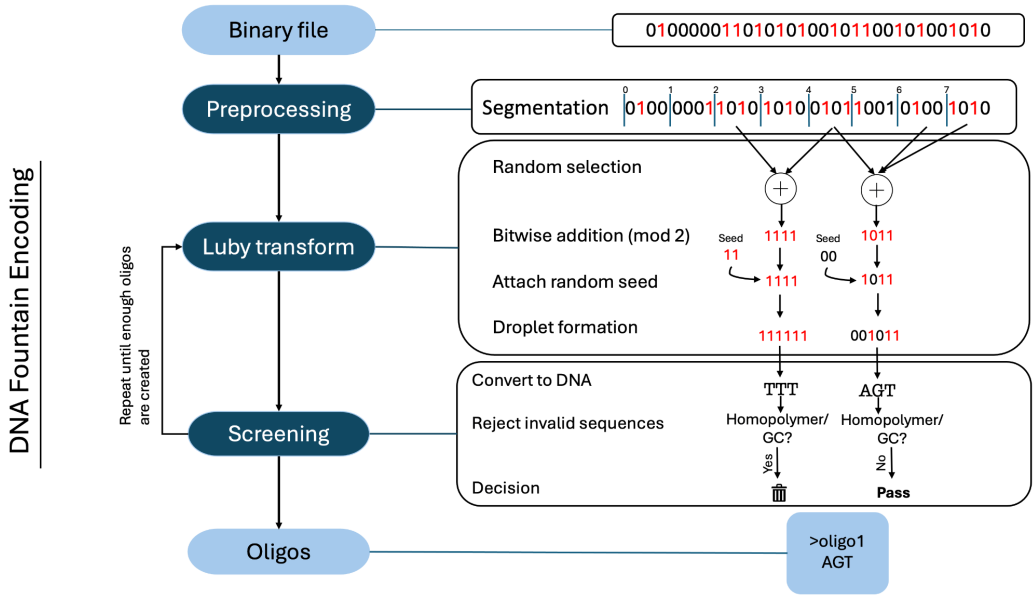


Figure 2.1.2 DNA Fountain Code Encoding [19]

2.1.2.1 Error Correction in DNA Fountain Codes

DNA Fountain incorporates Reed-Solomon (RS) error correction codes within the droplets. RS codes are effective in correcting substitution errors, which are common in DNA sequencing. Each droplet consists of a data payload and a seed, with the RS code adding redundancy to the payload, enabling error detection and correction.

However, DNA Fountain's error correction scheme is primarily designed to handle substitution errors. While it can correct substitution errors efficiently, it can only detect errors from insertions and deletions (indel errors). This limitation means that while the architecture provides robust protection against substitution errors, it may not fully address all types of errors encountered in DNA storage.

The decoding process uses a message-passing algorithm to reverse the Luby Transform. The decoder collects droplets, uses the seeds to identify the segments, and reconstructs the original data. Droplets with substitution errors are identified and discarded using the RS code, ensuring only error-free data contributes to the reconstruction. This efficient decoding process requires a subset of droplets slightly larger than the original file size, reducing the amount of sequencing needed for accurate data retrieval.

Different codec implementations for fountain codes are given in the literature, some of which focus on reducing repair bandwidth and complexity for distributed data storage applications. An example of an open source implementation of this is Founsure [1]. On the other hand, in broadcast/multicast file delivery and fast data streaming applications, minimizing the coding overhead might be of interest. In that case, variations of different low-complexity decoding algorithms (e.g., inactivation decoding in RaptorQ [17]) along with suitable code constructions are employed to timely deliver data in delay-sensitive applications such as multimedia streaming. Moreover, online codes can be adapted for object data storage to efficiently and reliably store large data sets (e.g., Amplidata [14]).

2.2 Overview of some DNA CODECs

The following sections describe some current DNA Codecs.

2.2.1 Ying Yang

Inspired by the rotating coding method of Goldman and the DNA Fountain approach, Ping et al. [12] introduced a novel DNA data storage encoding framework known as the Yin-Yang Codec (YYC). This method involves three primary phases. Initially, as illustrated in Figure 1.1.1.4 B, byte-level data is divided into uniform segments. In the next phase, two binary segments are chosen at random and fused on a bitwise level—first using the Yang rule and then the Yin rule—to produce the corresponding nucleotide, as depicted in Figure 1.1.1.4 A.

Following the encoding, the resulting DNA sequences undergo a rigorous screening process to ensure they meet predefined biochemical criteria: a GC content ranging between 40% and 60%,

homopolymers no longer than four bases, and a minimum free energy threshold of −30 kcal/mol to avoid secondary structures. If a candidate fails to satisfy these conditions, an alternative binary segment is randomly selected, and the process is repeated.

Notably, this encoding approach offers 1536 distinct ways to map binary sequences into DNA, with the potential to reach a theoretical maximum information density of 1.965 bits per nucleotide under the imposed constraints. Performance benchmarks were conducted using a custom-built software tool named *Chamaeleo*, where the YYC method was evaluated against established schemes, particularly the DNA Fountain strategy.

To assess the resilience of the YYC framework, both random and systematic errors were introduced. Simulations revealed that data recovery remained robust—up to 98% success—when sequence loss was kept below 2%. The method was further validated through experimental storage of two file types, implemented both synthetically as 200-nucleotide oligos and biologically as a 54,240-base pair DNA fragment in yeast cells. These trials indicated a projected physical storage capacity of approximately 200 exabytes per gram of DNA, marking a significant leap over earlier methodologies.

A flow diagram of the encoder is given in Figure 1.1.1.4 B. The Ying Yang rules are shown in Figure 1.1.1.4 A.

Yin Rule

Previous Nucleotide	Binary Digits: 0	Binary Digits: 1
A	A or G	C or T
T	C or T	A or G
G	A or G	C or T
C	C or T	A or G

Yang Rule

Binary Digits: 0	Binary Digits: 1
A or T	C or G

Figure 1.1.1.4 A: The Yin and Yang Rule [12]

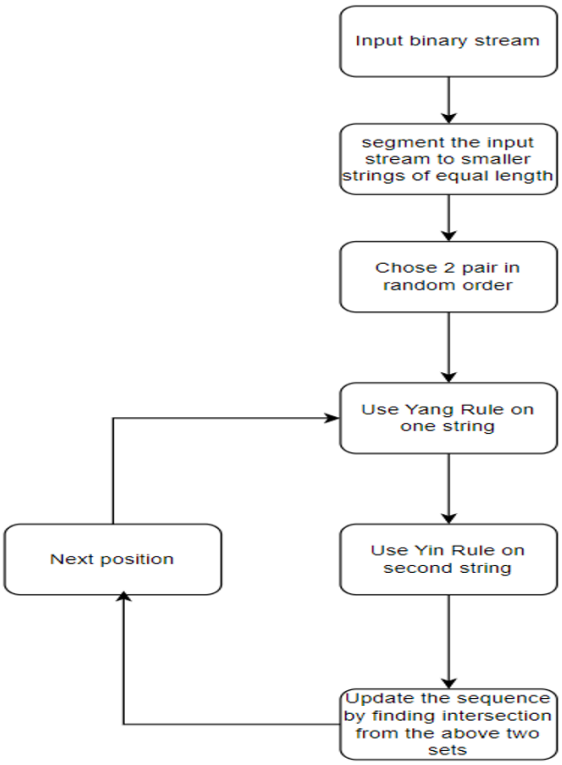


Figure 1.1.1.4 B: A flow diagram of Yin Yang Encoder

Example: Consider two binary segments as a=00110101 and b=11010110. Assuming the predefined virtual nucleotide as 'A', we will encode a1=0=[A,T] (using Yang rule) and b1=1=[C,T] (using Yin rule) which will give us next nucleotide after A as [A,T] ∩ [C,T]=T so the overall encoded sequence will be 'AT'. Continuing in this way with all other bits in both the strings the final output of the encoder will be 'ATAGCTGTC'.

2.2.2 HEDGES

HEDGES (Hash Encoded, Decoded by Greedy Exhaustive Search) [23] is an infinite constraint length convolutional code (Figure 1.2.5.1) with features specific to DNA storage channels. The basic plan is based on the classical autokey cipher. The HEDGES encoder consists of two parts, the inner codec based on HEDGES and the outer codec based on Reed Solomon (RS) for correcting the errors (See Figure 1.2.5.2)

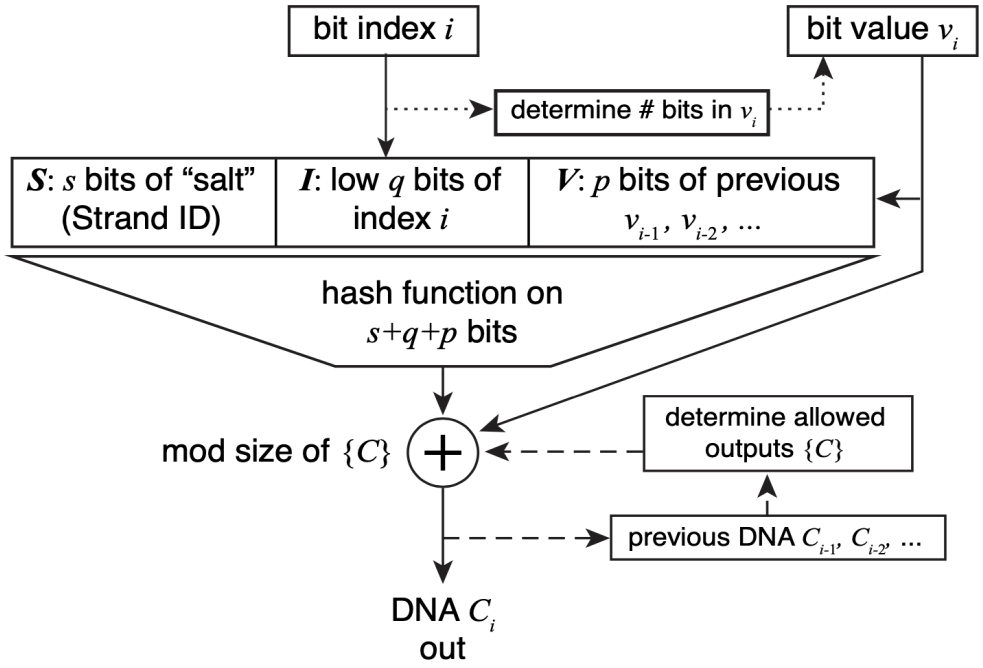


Figure 1.2.5.1: Full HEDGE Algorithm Flow [23]

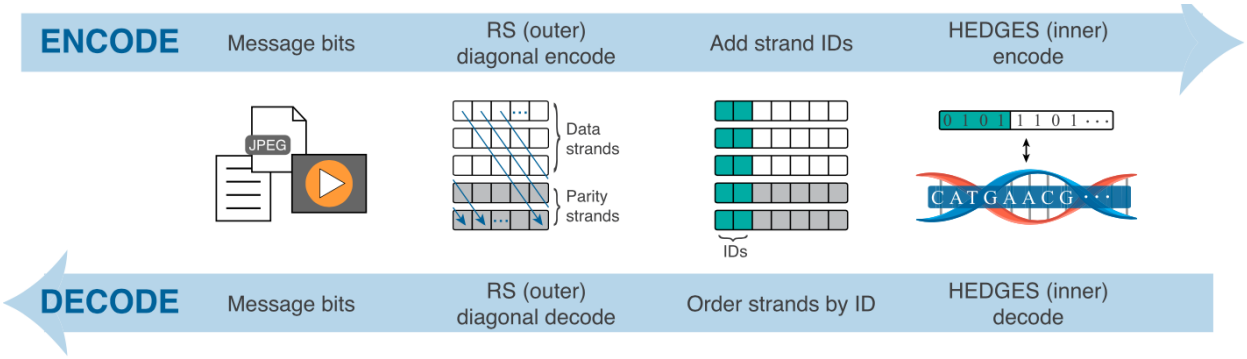


Figure 1.2.5.2: An Overview of HEDGES Codec [23]

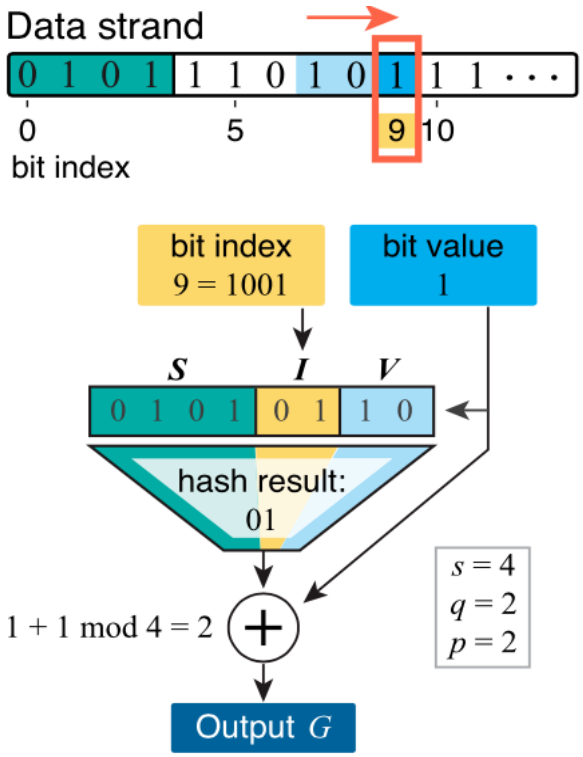


Figure 1.2.5.3: An Example of HEDGES Encoding bit 9 [23]

The RS correcting code is applied diagonally across multiple strands. In the example, 1 bit of input generates 2 bits of output. Hashing each bit value with its strand ID, bit index, and a few previous bits positions bad decoding hypotheses, allowing for correction of indels.

2.2.3 ADS Codec

The ADS Codec or ACOMA (Adaptive Codec for Organic Molecular Archives) [24] attempts to address GC and homopolymer constraints to ensure that the resulting oligos can be synthesized, as well as error correction to address errors in synthesis, amplification (through PCR) and sequencing (but not storage).

The codec works on three levels:

1) **Bit packing:** Converts bits to nucleotides.
2) **Single oligo layout:** Defines the structure of an oligo, i.e., location and size of index/address (also referred to as oligo identification) as well as of data/payload. Also entails metadata for error correction.
3) **Error correction:** Error correction across multiple oligos

For **bit packing** (level 0), all oligos of length 17 nucleotides are enumerated and the ones that cannot be used due to homopolymer length (fixed, max of 4 for A, T, C and max of 2 for G) are discarded. GC content restrictions are handled on the next level. The remaining oligos are mapped to bit patterns, resulting in 32 data bits + 1 parity bit.

Single oligo layout (level 1) defines how single oligos are structured. As can be seen in the figure below, an oligo is a collection of blocks (length 32 bits for data/payload and one parity bit to detect errors) as well as metadata (high GC flag - GC content handled on this level rather than level 0, erasure flag for level 2 as well as address/ID of the data block in the file). The metadata is split into N-1 blocks and placed between the data blocks where N is 33 (the size of the data block in bits + parity bit).

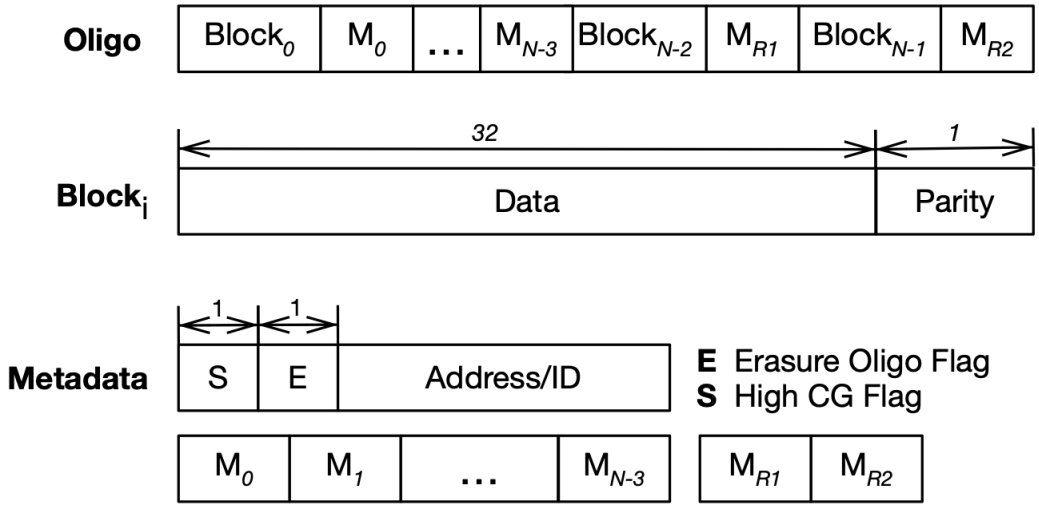


Figure 2.1.6 A: ADS Codec [24]

Error correction (level 2) defines the error correction approach. The figure illustrates error correction based on Reed-Solomon erasure coding. Specifically, a number of oligos are grouped together and erasure codes for data blocks from different oligos are encoded. Columns consist of blocks from different positions (to correct for spatial bias of errors).

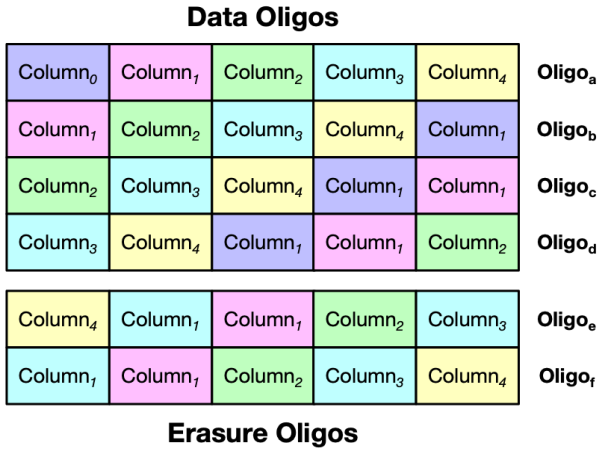


Figure 2.1.6 B: ADX Codec Error Correction [24]

2.2.4 DNA Aeon

The DNA Aeon codec [20] consists of an outer foundation code (Raptor fountain code) and an inner code resembling an arithmetic code (it is a lossless entropy encoding technique used as a basis for many common video standards). See Figure 2.1.7.1 for an overview.

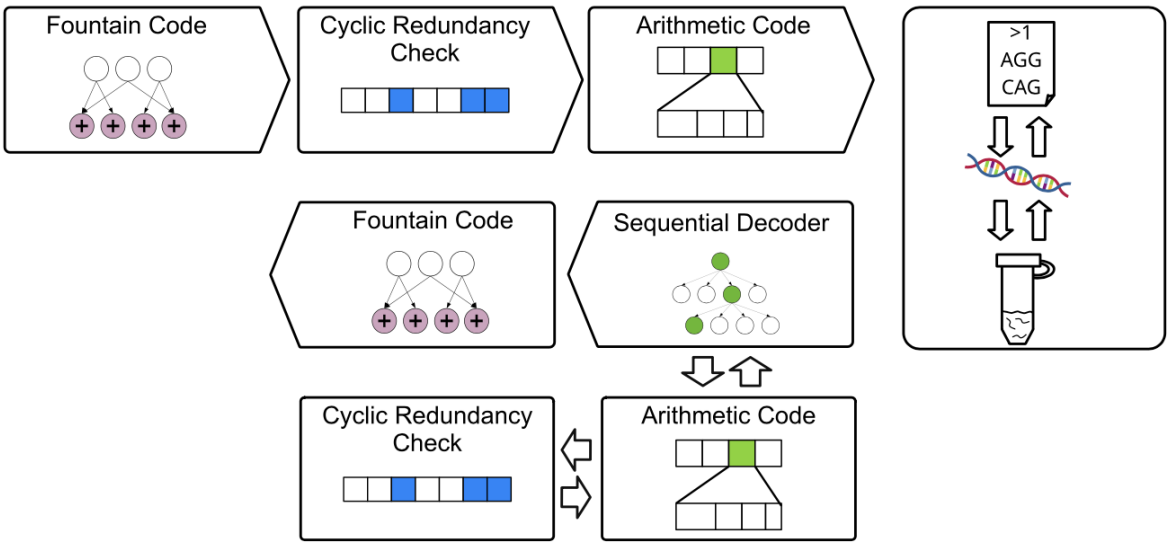


Figure 2.1.7.1 An overview of the DNA Aeon codec: Input data is encoded and packetized using the Raptor fountain code, followed by periodic insertion of an 8-bit CRC checksum including a final CRC. The resultant packets are then encoded in parallel using the arithmetic code [20].

The arithmetic code compresses the data by iteratively partitioning the interval (0,1) into smaller subintervals. For a simple example of arithmetic coding, see Figure 2.1.7.2 [20].

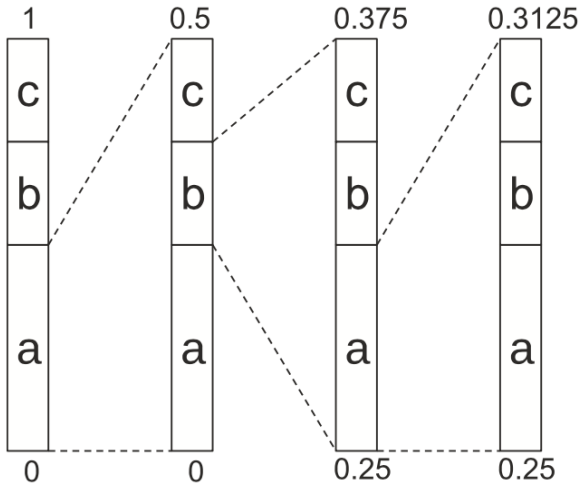


*Figure 2.1.7.2 A Simple Example of Arithmetic Code: The encoding of the string "abc" with probabilities a:0.5; b:0.25; c:0.25 after encoding the final symbol c is [0.296857,0.3125]. In the first iteration, the interval [0, 1) is divided into three subintervals: [0, 0.5) for the symbol **a** as the first character of the string, [0.5, 0.75) for **b**, and [0.75, 1) for **c**. Since **a** is the first symbol in the given example, the current subinterval becomes [0, 0.5) after encoding the first symbol. In the next iteration, this subinterval is further subdivided into [0, 0.25), [0.25, 0.375), and [0.375, 0.5). After complete encoding we get [0.296857,0.3125]. Source [20]*

2.2.5 Deep DNA Storage

By leveraging advanced neural network architectures, deep learning provides significant improvements in tasks ranging from base calling and synthesis prediction to sequence reconstruction and database searches. Depending on the application, different technologies can be applied: Transformers for tasks requiring long-range dependencies and global context understanding; Convolutional Neural Networks (CNNs) for capturing local patterns and motifs in DNA sequences; Recurrent Neural Networks (RNNs) for modeling sequential dependencies and temporal patterns; hybrid models to leverage the strengths of multiple architectures. The main applications of deep learning in the DNA data storage field are:

- **Nanopore Base Calling**: Deep learning models are used to convert raw signal data from nanopore sequencers into nucleotide sequences (base calling) [32]. CNNs and RNNs are commonly used for this task. These models can handle the noise and variability inherent in nanopore sequencing data.

- **Synthesis success factors**: Predicting the success of oligonucleotide synthesis based on various input parameters, such as sequence composition, secondary structure, and chemical properties [7]. They help optimize synthesis protocols by predicting the likelihood of synthesis failures or low yields. This leads to more efficient and cost-effective synthesis processes.

- **Predicting DNA hybridization**: Predicting the binding efficiency and stability of DNA hybridization, based on the minimum free energy (MFE) of secondary structures that indicate potentially undesirable behaviors [33]. Deep learning models can predict melting temperatures and hybridization affinities based on sequence data and help design primers with optimal binding properties for PCR.

- **Database search applications**: Content-based similarity search involves finding sequences that are similar based on their content rather than their exact sequence [3]. In the context of hybridization, the task consists of identifying nucleic acid sequences that are likely to hybridize with a given target sequence. Deep learning models, particularly hybrid models combining CNNs, RNNs, and Transformers, are well-suited for this task because they can capture both local and global patterns in sequences. For instance, convolutional layers can detect motifs and short patterns that are important for hybridization, while certain sequences might form stable hairpins or loops that affect binding.

- **Sequence reconstruction**: Reconstructing data from single-read sequences can be challenging due to errors introduced during sequencing. Deep learning techniques help denoise and correct errors in single-read sequences. The use of specific tokens helps the model understand the structure and syntax of the sequences, improving its ability to correct errors and denoise the data. A transformer model is trained using a self-

supervised learning approach, which means the model learns to predict parts of the sequence based on other parts, without needing labeled data. [34] [2]

2.3 Summary

The following Table summarizes characteristics of the codecs discussed above, plus others in the literature, noting the achievable information densities for different DNA data packet formats

Reference	Packet Architecture	Year	Data Size	Strand Length (nt)	ID (bit/nt)	Coverage	Random access	ECC
Church et al. [25]	Indices + Payload	2012	0.65MB	159	0.6/0.83	3000x	No	1 bit per base to resolve DNA constraints
Goldman et al. [22]	Payload	2013	0.63MB	117	0.19/0.29	520X	No	Base 3 to resolve DNA constraints
Grass et al. [6]	Address + indices + payload	2015	0.08MB	158	0.86/1.16	371X	No	Constrained codes + RS
Bornholt et al. [26]	Payload + index	2016	0.15MB	120	0.57/0.85	40X	Yes	Repetition
Erlich & Zielinski [19]	Payload + index + RS	2017	2.11MB	200	1.19/1.57	10.5X	No	RS + Fountain codes + removed bad strands
Yazdi et al. [21]	Payload + Address + Index	2017	0.003MB	1000	1.7/1.74	200X	Yes	Constrained codes + alignment + deletion correction
Blawat et al. [27]	Payload + index	2016	22MB	230	0.89/1.08	160X	No	RLL + BCH (address) + RS + CRC
Organick et al. [28]	Payload + Indices	2018	200MB	150-154	0.81/1.1	5X (Illumina), 36X/80X (Nanopore)	Yes	RS
Chandak et al. [29]	Index + Payload + BCH	2019	1.69MB	150	0.55-0.75	5X	No	BCH + LDPC
Wang et al. [30]	Address + Payload + CRC	2019	0.4MB	190	1.56/1.77	12-545X	No	CRC (single primer)
Anavy et al. [31]	Payload + RS + Barcode + Identifier	2019	6.4MB	194	1.57-1.96	~10X	No	RS + Fountain codes (composite base - 6)
Ying-Yang [12]	Payload + RS + index	2022		200	1.965		No	RS
Cao et al. [36]	Adaptive coding	2022	698 KB	162	1.29/1.22	35	Yes	Fountain codes + RS
El-Shaikh et al. [37]	High-scale random access	2022		133, 189			Yes	Fountain codes
Ping et al. [38]	practical and robust DNA-based data archiving	2022	52 KB	200	1.95		Yes	yin–yang codec + RS
Song et al. [39]	Robust data storage	2022	6.8 MB	200	1.3	variable	Yes	de Bruijn graph-based
Welzel et al. [40]	DNA-Aeon	2023	91.4 KB	10			Yes	Inner AC based, outer fountain (Raptor)
Zan et al. [41]	Modulation Encoding and Decoding	2023		120 + primers	1.0	100	Yes	pairwise sequence alignment
Gomes et al. [42]		2024	38 B	146	1.6			RS + LDPC

Reference	Packet Architecture	Year	Data Size	Strand Length (nt)	ID (bit/nt)	Coverage	Random access	ECC
Preuss et al. [43]	shortmer combinatorial encoding	2024		220		12		2D RS
Zhao et al. [44]	Composite Hedges	2024	4 KB	243	1.17	4-8		RS

Table 2.1: Examples of end-to-end DNA data storage demonstrations. Bit density is denoted as (encoded data with overhead) / (encoded data only). Coverage denotes the number of times a given sequence was read during sequencing and, unless otherwise noted, sequencing was done using Illumina sequencers. ECC denotes the error correction codes applied.

3 DNA Codec Requirements and Criteria

This section discusses the requirements for DNA codecs and the criteria to consider about their capabilities and performance. We first briefly summarize the DNA error model and then move to requirements and criteria, which ultimately flow from this model.

3.1 DNA Errors

DNA synthesis and sequencing introduce errors, so encoding algorithms must include error-correcting codes. DNA sequences are subject to multiple error modes including substitutions, insertions, deletions, and erasures. These errors may be present in isolation or in bursts of any combination. Below is a summary of these error types:

- **Substitutions**: Sometimes referred to as Single Nucleotide Variants (SNVs), or Single Nucleotide Polymorphisms (SNPs, primarily in genomics literature), substitution errors occur when an incorrect/unintended base take the place of a correct/intended base in a DNA sequence. These are analogous to bit errors in digital systems.
- **Insertions**: Insertion errors occur when a base has been erroneously inserted into a DNA sequence. There is no analogous common error type in electronic systems.
- **Deletions**: Deletion errors occur when a base has been erroneously deleted from a DNA sequence. There is no analogous common error type in electronic systems.
- **Erasures**: Erasures occur when a base or sequence cannot be read but it is known to exist. Erasures may occur when sequencing technology reads a base with low confidence, encounters an abasic site (a location in the DNA sequence where the base is missing, resulting in a site without a nucleotide base), or when an entire sequence is absent from a series of reads. Some sequencing technologies may insert a default rather than reporting an erasure.

3.2 CODEC Scalability

The scalability of the algorithms used to encode and decode data in DNA storage systems is a key factor in the practicality of DNA data storage. Several algorithmic challenges affect the scalability of these CODECs. Here we examine the algorithms most commonly used in the DNA data storage process (see Figure 1), which may differ from one CODEC to another (see Section

2). These include the clustering of similar strand copies containing errors, multi-strand alignment and error correction.

3.2.1 Clustering

In traditional data storage and transmission channels, CODECs use Hamming distance for error correction and detection. For a DNA-based data storage channel, however, Levenshtein distance, the minimum number of insertions, deletions and substitutions required to change one sequence to another, is the typical metric. This metric is usually applied to regroup strands. Unlike Hamming distance (an $O(n)$ algorithm), calculating Levenshtein distance is a dynamic algorithm with time complexity $O(n1 \cdot n2)$, where n1 and n2 are the lengths of the DNA molecules being compared (Wagner & Fischer, 1974). If both are the same length, n, then Levenshtein distance is an $O(n^2)$ algorithm. If there are *k* strands to compare, the total complexity depends on the number of pairwise comparisons to be made. In the simplest case, where all the strands have approximately length *n*, the complexity is $O(k^2 \cdot n^2)$. This reflects the quadratic growth in both the number of strand comparisons and the time required for each calculation. The algorithm is therefore very computationally expensive as *k* increases.

Based on the Levenshtein distance, it is possible to cluster all molecule copies together, even if there are errors, which ensures that the correct consensus payload strand can be derived from each group, which in turn ensures that information can be successfully retrieved and decoded from an archive. The more copies there are, the lower the probability of error at the outcome of the consensus.

3.2.2 Multistrand alignment

In DNA decoding, it is generally assumed that strands belonging to the same original strand, but which differ due to indel and substitution errors, must be aligned to improve the chances of finding the original strand. There are two main approaches to strand alignment:

1) **Exact algorithms** based on a dynamic programming approach involving the building of a multi-dimensional matrix. The time complexity of aligning *n* sequences of length *m* is $O(m^n)$. This exponential complexity makes it impractical for large *n* or *m* sequences. This type of algorithm is therefore mainly used for small datasets.

2) **Heuristic algorithms** like ClustalW (Chenna, 2003) or MAFFT (Katoh & Toh, 2008) are often used to manage the exponential time complexity of exact methods. These provide good, but not necessarily optimal, alignments with significantly lower complexity of the order of $O(n \cdot m^2)$. Tools like T-Coffee (Korak et al., 2021) improve alignment accuracy by incorporating consistency information. However, they come with additional computational cost, often with a time complexity of about $O(n^3 \cdot m^2)$.

3.2.3 Error correction mechanisms

Several error-correcting codes (ECC) can be applied to DNA data storage. Although effective, these codes increase both the complexity and computational load of the CODECs, as they introduce overhead by adding redundancy to the data, reducing net storage capacity and slowing down encoding and decoding operations. At a minimum, a CODEC suitable for data storage in DNA should have:

1) A level of parameterization that allows for the matching of error correction to the error rate of the underlying physical system (the composition of synthesis, storage, and sequencing)
2) Some consideration for indels. (Even if it doesn't handle them elegantly. it should at least recognize reads are the wrong length and discard them, rather than crashing or corrupting normally recoverable data.)

Multiple error correcting codes can be combined to utilize their respective strengths. In a process of concatenation, concatenated codes form a class of error-correcting codes that are derived by combining an inner code and an outer code.

Generally, the inner code operates on a smaller block of data, usually referred to as a "subblock" or "inner codeword". The primary responsibility of the inner code is to correct errors within this smaller block. The inner code generally offers stronger error correction capabilities and is capable of handling more significant error rates.

The outer code operates on a larger block comprising multiple subblocks generated by the inner code. This larger block is often referred to as a "superblock" or "outer codeword." The outer code is responsible for correcting residual errors that might not have been corrected by the inner code as well as any new errors that occurred during storage.

The following is a brief complexity comparison of some of the most common ECC schemes:

- **Parity codes** work by adding an extra bit to a block of data so that the total number of 1's in the block is always even or odd, depending on the type of parity used.
    - Can detect but not correct single bit errors.
    - Does not work well with large blocks of data.
    - For N data bits, linear complexity $O(N)$ for generating and checking.
- **CRC (Cyclic Redundancy Check) codes** generate a checksum based on the data being transmitted, and appends that checksum to the data.
    - Can detect single- and multi-bit errors but cannot correct errors.
    - Based on an optimization scheme, complexity can be linear for generating and checking.
- **Hamming codes** work by adding extra parity bits to the data being transmitted. The number of parity bits added depends on the number of data bits being transmitted.
    - Can detect 1-bit and 2-bit errors. Can correct 1-bit errors.
    - Complexity is superlinear $O(NlogN)$ for generating and checking.

- **LDPC (Low Density Parity Check) code** works by solving a set of linear equations to correct errors in the data. LDPC is considered the state of the art when error rates are significantly high.
    - Can detect and correct single- and multi-bit errors.
    - Linear complexity for generating, and superlinear $O(NlogN)$ for decoding.
- **Erasure codes** work by adding redundant information to a datum that allows the receiver to reconstruct the datum even if parts of it are missing.
    - Can detect and correct errors, including the loss of entire data blocks.
    - No closed form for complexity (either for generating or checking). For the frequently used Reed-Solomon ECC, parity symbols are calculated using polynomial evaluation in a Galois Field (GF). Given an n-symbol codeword and k-data symbols, where $n = k + 2t$ and t is the number of correctable errors, the encoding process requires $O(n \cdot k)$ multiplications and additions in the GF. Decoding is more complex as it involves identifying and correcting errors, which needs to first compute syndromes, then finds and locates errors based on a locator polynomial, with an overall complexity of $O(n \cdot t + t^2)$. Thus, in practical terms, the complexity of Reed-Solomon increases with the number of symbols *n* and the number of correctable errors *t*.
- **Fountain codes** work by using a randomized algorithm to generate an infinite stream of encoded packets from a single source packet. The receiver can reconstruct the original data by collecting enough of these encoded packets.
    - Can detect and correct errors, including the loss of entire data blocks.
    - Generating and decoding is linear for Raptor.
    - Generating is superlinear $O(NlogN)$ and decoding superlinear $O(NlogN)$ to quadratic $O(n^2)$ for Luby Transform.
    - Less efficient than other error correction codes requiring more storage to achieve the same level of error correction.
- **Viterbi codes** work by generating a series of encoded bits from the original data stream using a shift register and feedback. The receiver uses a technique called maximum likelihood decoding, which compares the received signal with all possible transmitted sequences to determine the most likely sequence.
    - Can detect and correct errors.
    - Complexity is O(N·2k), where N is the length of the input sequence and k is the constraint length of the convolutional code (defines the memory depth of the encoder)

3.3 Performance (throughput)

Given the nature of the DNA archive medium, and that archives are built to survive decades, new technologies will likely be introduced, making designating absolute CODEC performance numbers a challenge. Instead, we provide guidelines on optimization directions and considerations when designing a CODEC, to be used as a framework to evaluate CODEC performance.

1. The CODEC being a computational entity, should be faster than the slowest part of a larger system, and provide a level of performance that appears to be subjectively reasonable (measured in bytes/bits over time). Specifically, it should encode faster than the synthesizer and decode faster than the sequencer on a desktop workstation class of computer.

2. The CODEC should have a high efficiency level both computationally and energetically (throughput per watt). Throughput should take into account valid archives (the archive must conform to the expected format and structure defined by the CODEC), as well as different error-handling scenarios within the envelope defined in the Error Correction section.

3. The CODEC should be able to be implemented as pure software. Hardware may be employed as an acceleration mechanism, but implementations should not rely on the existence of such hardware.

3.4 Compatibility

DNA archives are built to survive for decades. An underlying assumption is that the technology used to recover the archive will likely not be the same as the one used to encode the archive. This raises the question of what is needed to ensure that future archive readers will be able to read the archive.

A CODEC should provide a detailed Codec Format Specification of the archive format it produces. This document is provided when registering the CODEC. The Codec Format Specification should contain sufficient details to enable the decoding of the archive, including the structures, layout, and error correction paradigm. This enables encoder and decoder implementations to vary widely based on technologies existing at the time of use, and in particular, regarding decoding, it avoids a reliance on decoder technologies that were used at the time of archive creation.

3.4.1 Codec Format Specification

The DNA Data Storage Alliance Sector 0 and Sector 1 specifications [45], the so-called DNA Archive Rosetta Stone specifications, are a proposed instance of a Codec Format Specification. The decoder identification is defined by the Sector 0 specification. The type and version are used to look up the remainder of the codec specification in Sector 1, which contains:

1. Physical media description and parameters - bases, oligo lengths, prefixes/suffixes
2. Media constraints and tolerances that decoder must be aware of
3. Complete description of the error correction algorithms in use and how they are used, this includes layout diagrams as needed
4. Specifics of references and details on how to specify formats in the file, for example such as defined given by the PRONOM register [46]

5. Logical definitions of the file or filesystem format including metadata, logical structures, and data layout

At a minimum, the Codec Format Specification should document the error correction algorithms and media parameters and add references to the media standards being used in the encoding. While it may not be specific to the media at hand, the Codec Format Specification must have enough information to enable the selection of a decoder that is capable of processing the DNA archive.

3.5 Summary of Criteria of DNA Codecs

In the above sections, we have reviewed DNA Codec requirements and criteria. The following are considered the general criteria a CODEC must meet in order to provide the experience desired from using DNA as a storage media type.

Criterion	The Codec should …
Scalability	• use sub-quadratic algorithmic complexity as a function of sequence length and the number of strands
Performance	• be faster than the slowest parts of the system, such as synthesis or sequencing • have a high efficiency level both computationally and energetically (throughput per watt) • be able to be implemented as pure software, even if hardware may be employed as an acceleration mechanism
Error detection and correction	• have an acceptable probability of successful decoding for a given amount of data, while this probability should take all 3 types of error into account
Compatibility	• provide a detailed specification of the file format it produces, whereas the specification must contain sufficient details to enable reading the archive
Biological constraints	• verify the GC ratio, avoid homopolymers and hybridization, among other factors

DNA Codecs are an integral part of the DNA data storage pipeline. We believe that standard open source codecs will emerge over time, but this will require that the DNA data storage ecosystem matures and the technical coupling between the protocol embodied by the codec and the chemistry used in the physical parts of the pipeline become more routine [10]. Until then, we hope this whitepaper helps to guide codec developers, thereby facilitating the emergence of an interoperable DNA data storage ecosystem.

4 Annex 1 – Codecs for Linear Tape Open (LTO)

In tape data storage, Error Correction Coding (ECC) is internal to the hardware design (implemented at the Register-Transfer level) to achieve rapid data processing and very high data integrity. In many standards including the optical discs and LTO standard, a product code (named after the concept of a Cartesian product) is used as the ECC mechanism of the read channel for providing highly reliable data protection. A 2-D product code (see Figure 2.1.3) can be constructed by concatenating two (conventionally linear) block codes: a code C1 with parameters [n1, k1, d1] and a code C2 with parameters [n2, k2, d2], where ni, ki, and di (i = 1, 2) stand for codeword length, number of information symbols, and minimum Hamming distance of the code, respectively.

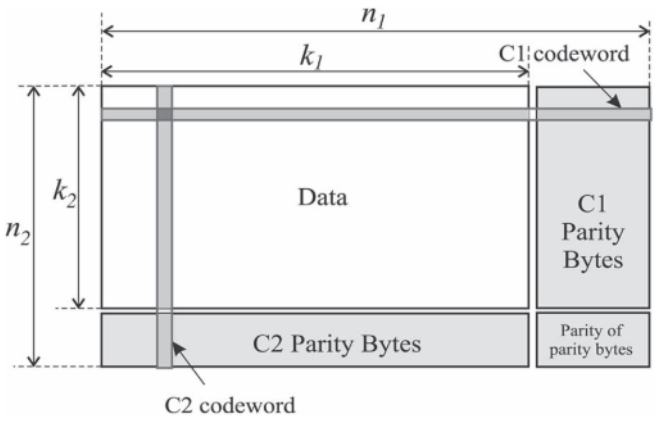


Fig 2.1.3: A 2D Product code representation with two constituent codes: C1 and C2.

A typical construction of the product code P1 = C1 × C2 is shown in Fig. **2.1.3**. using two systematic block codes. A k1 × k2 data array (data blob or frame) is first encoded vertically using code C2 (coding k1 columns using code C2), and the encoded data are then horizontally coded using code C1 (coding n2 rows using code C1). The order of encoding operations does not matter as long as the constituent codes are linear. In LTO, constituent linear codes are selected to be Reed-Solomon codes. In the later versions of LTO (beginning with LTO7), the constituent codes started to encode metadata (such as header information) as well alongside with the user data.

The decoding is typically achieved by Bounded Distance Decoding - a low complexity decoding algorithm that algebraically relies on “decoding spheres” around the codewords determined by a codebook. An example is shown in Fig. 2.1.4. The first constituent code (typically C1) is decoded in error correction mode - any word within the sphere decoded to the corresponding codeword in the center. In other words, all available redundancy is used to correct for error detection and correction. For instance, if there are n (=2 in Fig. 2.1.4) redundant symbols floor(n/2) (=1 in Fig. 2.1.4) errors can be decoded successfully. If there are more than floor(n/2) errors, depending on the number and their distribution along the codeword, the decoder can detect the error but cannot correct in which case, all symbols could be labeled/flagged as “erased” (before vertical decoding, see 2.1.4). Later in decoding, depending on the decoder failure flags, some of the codewords could be labeled as erasures and passed on to the next constituent code decoding (typically C2)

now in erasure correction mode (can correct 2 erasures for the code given in Fig. 2.1.4). Furthermore, to leverage the full potential of error and erasure correction capabilities of the constituent codes (C1 and C2), various techniques are developed such as AI-based mode selections and iterations [reference https://patents.google.com/patent/US11990920B2/] between the decoders to make sure that the decoding operation comes close to the optimal data retrieval (in maximum likelihood sense) performance with only minor additional complexity/hardware. As part of QoS guarantees, standard LTO ECC parameters (power of error correction) are selected such that the bit error rate at the user level is guaranteed to be 10^{-20} or less i.e. an average of 1 bit error in 10^{20} bits.

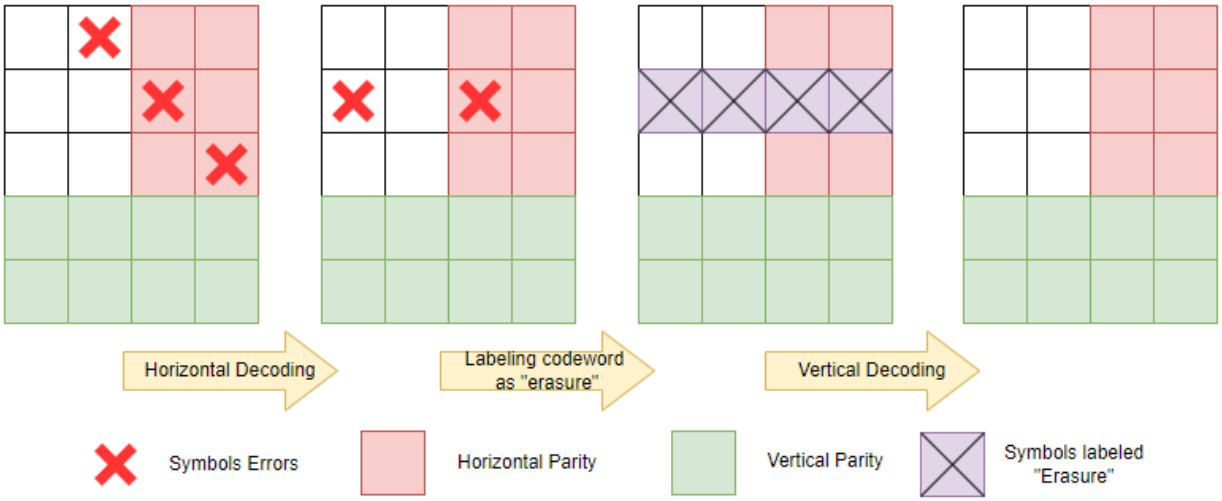


Fig 2.1.4: An example decoding process where C1 decoder (horizontal decoding) is operated in error correction and C2 decoder (vertical decoding) in erasure correction mode.

5 Acknowledgements

We thank the following individuals for their contributions to this effort.

Manish K. Gupta (Kaushalya: The Skill University)
Pierre-Yves Burgi (University of Geneva)
Suayb S. Arslan (Boğaziçi University, MIT)
David Landsman (Western Digital)
Chris Takahashi (University of Washington)
Omer Sabry (Technion)
Daniella Bar-Lev (Technion)
Don Doerner (Quantum Technologies)
Joel Christner (Dell)
Udi Shemer (Dell)
Turguy Goker (Quantum)
Natalio Krasnogor (Newcastle, UK)
Eva Gil San Antonio (I3S University Côte d'Azur and CNRS)
Alessia Marelli (Avaneidi)
John M. Hoffman ()
Takashi Kobayashi (Fujitsu)
Bill Efcavitch (Molecular Assemblies)
Gerardo Bertero (Western Digital)
Gemma Mendonsa (Seagate)
James Banal (CacheDNA)
Marthe Colotte (Imagene)
Stephane Lemaire (Sorbonne University, Biomemory)
Esther Singer (Twist Bioscience)
Thomas Heinis (Imperial College London)
Andre Martins (University of São Paulo)
Thiago Aoyagi (Instituto de Pesquisas Tecnologicas)
Emanuele Viterbo (Monash University)

6 References

[1] Şuayb Ş. Arslan, Founsure 1.0: An erasure code library with efficient repair and update features, SoftwareX, Volume 13, 2021, 100662, https://doi.org/10.1016/j.softx.2021.100662, Codec Implementation: https://github.com/suaybarslan/founsure

[2] Bar-Lev, D., Orr, I., Sabary, O., Etzion, T., Yaakobi, E. (2021) Deep DNA Storage: Scalable and Robust DNA Storage via Coding Theory and Deep Learning. arXiv:2109.00031

[3] Bee, C., Chen, YJ., Queen, M. et al. (2021) Molecular-level similarity search brings computing to DNA data storage. NatCommun 12, 4764.DOI: 10.1038/s41467-021-24991-z

[4] Buterez, D. (2021) Scaling up DNA digital data storage by efficiently predicting DNA hybridisation using deep learning. Sci Rep 11, 20517.DOI: 10.1038/s41598-021-97238-y

[5] Chenna, R. (2003). Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Research, 31(13), 3497–3500. https://doi.org/10.1093/nar/gkg500

[6] Grass, Robert N., et al. "Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes." Angewandte Chemie International Edition 54.8 (2015): 2552-2555.

[7] Halper, S.M., Hossain, A., Salis, H.M. (2020) Synthesis Success Calculator: Predicting the Rapid Synthesis of DNA Fragments with Machine Learning. ACS Synthetic Biology 9 (7), 1563-1571. DOI: 10.1021/acssynbio.9b00460

[8] Katoh, K., & Toh, H. (2008). Recent developments in the MAFFT multiple sequence alignment program. Briefings in Bioinformatics, 9(4), 286–298. https://doi.org/10.1093/bib/bbn013

[9] Korak, T., Aşir, F., Işik, E., & CengiZ, N. (2021). Multiple sequence alignment quality comparison in T-Coffee, MUSCLE and M-Coffee based on different benchmarks. Cumhuriyet Science Journal, 42(3), 526–535. https://doi.org/10.17776/csj.842265

[10] Landsman, D. and Strauss, K., "The DNA Data Storage Model" in Computer, vol. 56, no. 07, pp. 78-85, July 2023, doi: 10.1109/MC.2023.3272188.

[11] Nahum, Y., Ben-Tolila, E., Anavy, L. (2021) Single-Read Reconstruction for DNA Data Storage Using Transformers. arXiv:2109.05478

[12] Ping, Z., Chen, S., Zhou, G. et al. (2022) Towards practical and robust DNA-based data archiving using the yin–yang codec system. Nat Comput Sci 2, 234–242. DOI: 10.1038/s43588-022-00231-2

[13] Song, Z., Yuan L., Jing, Y. (2022) Nanopore Detection Assisted DNA Information Processing. Nanomaterials 12(18) 3135. DOI: 10.3390/nano12183135

[14] K. D. Spiegeleer and R. R. A. Slootmaekers, “Method of Storing a Data Set in a Distributed Storage System, and Computer Program Product for Use with Said Method", U.S. Patent 2011/0113282 A1, May 12, 2011.

[15] Takahashi, C.N., Nguyen, B.H., Strauss, K. et al. Demonstration of End-to-End Automation of DNA Data Storage. Sci Rep 9, 4998 (2019). https://doi.org/10.1038/s41598-019-41228-8

[16] Wagner, R. A., & Fischer, M. J. (1974). The String-to-String Correction Problem. Journal of the ACM, 21(1), 168–173. https://doi.org/10.1145/321796.321811

[17] Qualcomm Incorporated White Paper, “RaptorQTechnicalOverview",2010. Available online: https://www.qualcomm.com/news/releases/2010/10/qualcomm-unveils-raptorq-product-enhanced-digital-content-delivery

[18] Ilan Shomorony and Reinhard Heckel (2022), "Information-Theoretic Foundations of DNA Data Storage", Foundations and Trends® in Communications and Information Theory: Vol. 19: No. 1, pp 1-106. http://dx.doi.org/10.1561/0100000117

[19] Erlich Y, Zielinski D. DNA Fountain enables a robust and efficient storage architecture. Science. 2017 Mar 3;355(6328):950-954. doi: 10.1126/science.aaj2038. PMID: 28254941.

[20] Welzel, M. et al. DNA-Aeon provides flexible arithmetic coding for constraint adherence and error correction in DNA storage. Nat. Commun. 14, 628 (2023).

[21] Yazdi, S. M. H. T., Yuan, Y., Ma, J., Zhao, H. & Milenkovic, O. A Rewritable, Random-Access DNA-Based Storage System. Sci. Rep. 5, 14138 (2015).

[22] Goldman, N. et al. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. Nature 494, 77–80 (2013).

[23] William H. Press, John A. Hawkins, Stephen Knox Jones Jr, Jeffrey M. Schaub, and Ilya J. Finkelstein, HEDGES Error-Correcting Code for DNA Storage Corrects Indels and Allows Sequence Constraints Proc Natl Acad Sci. 117 (31) 18489-18496 (July 16, 2020)

[24] Z. Ping, N. Goldman, and J. Yuan, "ADS Codex: Adaptive Codec for Organic Molecular Archives (ACOMA)," arXiv preprint arXiv:2309.02219, 2023. [Online]. Available: https://arxiv.org/abs/2309.02219.

[25] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," Science, vol. 337, no. 6102, p. 1628, 2012, doi: 10.1126/science.1226355.

[26] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A DNA-based archival storage system," in Proc. 21st Int’l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '16), 2016, pp. 637–649. doi: 10.1145/2872362.2872397.

[27] M. Blawat et al., "Forward error correction for DNA data storage," Procedia Computer Science, vol. 80, pp. 1011–1022, 2016. doi: 10.1016/j.procs.2016.05.445.

[28] L. Organick et al., "Random access in large-scale DNA data storage," Nature Biotechnology, vol. 36, no. 3, pp. 242–248, 2018, doi: 10.1038/nbt.4079.

[29] S. Chandak, S. Kannan, G. Seelig, L. Ceze, and K. Strauss, "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), 2019, pp. 127–133. doi: 10.1109/ALLERTON.2019.8919892.

[30] Y. Wang et al., "High capacity DNA data storage with variable-length oligonucleotides using repeat accumulate code and hybrid mapping," Journal of Biological Engineering, vol. 13, no. 1, p. 89, 2019, doi: 10.1186/s13036-019-0211-2

[31] 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, no. 10, pp. 1229–1236, 2019, doi: 10.1038/s41587-019-0240-x.

[32] W. Song et al., "High information density and low coverage data storage in DNA with efficient channel coding schemes," arXiv preprint arXiv:2410.04886, 2022. [Online]. Available: https://arxiv.org/abs/2410.04886

[33] D. Buterez, "Scaling up DNA digital data storage by efficiently predicting DNA hybridisation using deep learning," Scientific Reports, vol. 11, no. 1, p. 20517, 2021, doi: 10.1038/s41598-021-97238-y.

[34] Y. Nahum, E. Ben-Tolila, and L. Anavy, "Single-read reconstruction for DNA data storage using transformers," arXiv preprint arXiv:2109.05478, 2021. [Online]. Available: https://arxiv.org/abs/2109.05478

[35] Y. Nahum, E. Ben-Tolila, and L. Anavy, "Single-read reconstruction for DNA data storage using transformers," arXiv preprint arXiv:2109.05478, 2021. [Online]. Available: https://arxiv.org/abs/2109.05478

[36] Cao, B. et al. (2022) Adaptive coding for DNA storage with high storage density and low coverage. Systems Biology and Applications, 8(1). DOI 10.1038/s41540-022-00233-w

[37] El-Shaikh, A. et al. (2022) High-scale random access on DNA storage systems. NAR Genomics and Bioinformatics, 4(1). DOI 10.1093/nargab/lqab126

[38] Ping, Z. et al. (2022) Towards practical and robust DNA-based data archiving using the yin–yang codec system. Nature Computational Science, 2(4). DOI 10.1038/s43588-022-00231-2

[39] Song, L. et al. (2022) Robust data storage in DNA by de Bruijn graph-based de novo strand assembly. Nature Communications, 13(1). DOI 10.1038/s41467-022-33046-w

[40] Welzel, M. et al. (2023) DNA-Aeon provides flexible arithmetic coding for constraint adherence and error correction in DNA storage. Nature Communications, 14(1). DOI 10.1038/s41467-023-36297-3

[41] Zan, X. et al. (2023) A Robust and Efficient DNA Storage Architecture Based on Modulation Encoding and Decoding. Journal of Chemical Information and Modeling, 63(12). DOI 10.1021/acs.jcim.3c00629

[42] Gomes, C.P. et al. (2024) Coding, Decoding and Retrieving a Message Using DNA: An Experience from a Brazilian Center Research on DNA Data Storage. Micromachines, 15(4). DOI 10.3390/mi15040474

[43] Preuss, I et al. (2024) Efficient DNA-based data storage using shortmer combinatorial encoding. Scientific Reports, 14(1). DOI 10.1038/s41598-024-58386-z

[44] Zhao, X et al. (2024) Composite Hedges Nanopores codec system for rapid and portable DNA data readout with high INDEL-Correction. Nature Communications, 15(1). DOI 10.1038/s41467-024-53455-3

[45] DNA Data Storage Alliance, Sector 0 & 1: https://www.snia.org/dna/workgroups#DNAArchiveRosetta

[46] The National Archives Technical Registry; https://www.nationalarchives.gov.uk/pronom/

[47] Wicker, Stephen B., and Vijay K. Bhargava, editors. *Reed-Solomon Codes and Their Applications*. Wiley-IEEE Press, 1999.