# Short Systematic Codes for Correcting Random Edit Errors in DNA Storage

Serge Kas Hanna
I3S laboratory, Côte d'Azur University and CNRS, Sophia Antipolis, France
Email: serge.kas-hanna@{univ-cotedazur.fr, cnrs.fr}

***Abstract*—DNA storage faces challenges in ensuring data reliability in the presence of edit errors—deletions, insertions, and substitutions—that occur randomly during various phases of the storage process. Current limitations in DNA synthesis technology also require the use of short DNA sequences, highlighting the particular need for short edit-correcting codes. Motivated by these factors, we introduce a systematic code designed to correct random edits while adhering to typical length constraints in DNA storage. We evaluate the performance of the code through simulations and assess its effectiveness within a DNA storage framework, revealing promising results.**

## I. Introduction

DNA storage has emerged as a promising medium for next-generation storage systems due to its high density ($10^{15}$-$10^{20}$ bytes per gram of DNA [1]) and long-term durability (thousands of years [2]). One of the main challenges in DNA storage is ensuring data reliability in the presence of *edit* errors, i.e., deletions, insertions, and substitutions, which may occur during various phases of the storage process. A potential source of edit errors is sequencing and synthesis noise, with the actual error rate influenced by factors such as the technologies used and the length of the DNA sequence. Due to the limitations of current DNA synthesis technologies, a binary file is typically encoded in the form of several short DNA sequences, called oligos, usually a few hundred nucleotides long, to minimize synthesis noise [3]. In this setting, a common approach to enhance reliability involves using a combination of an inner and outer error-correcting code. Ideally, the inner code should be a short code capable of correcting edit errors within the oligos, while the outer code can be a longer erasure/substitution code designed to recover lost oligos or correct residual errors from the inner code.

Designing codes that correct edit errors is a fundamental problem in coding theory, dating back to the 1960s [4], [5]. This problem has gained increased interest in recent years, with numerous works dedicated to constructing codes for correcting: only deletions *or* insertions, e.g., [6]–[9], deletions *and* substitutions [10]–[13], and sticky insertions/deletions [14], [15]. However, much less is known about codes correcting all three types of edits simultaneously [16], [17]. The aforementioned works focus on correcting adversarial errors with zero-error decoding, often relying on asymptotics that apply to scenarios with a small number of errors and large code lengths. While such assumptions are typical in coding theory, they do not naturally extend to DNA storage systems, which require different considerations. Specifically, in addition to the need for short codes due to synthesis limitations, edit errors in DNA storage are known to be of random nature and potentially of large quantity. Some earlier studies have proposed concatenated coding schemes for correcting random edit errors, e.g., [18], [19]; however, the code lengths in these constructions are also large, typically in the order of several thousands.

Due to the lack of suitable edit-correcting codes, standard approaches in the literature use off-the-shelf substitution/erasure correcting codes as inner/outer codes for DNA storage. To overcome the challenge of correcting deletions and insertions, existing methods often rely on deep sequencing, which generates many reads per oligo. This injects sequencing redundancy analogous to repetition coding, typically leveraged through sequence alignment algorithms to correct edits via majority voting. While alignment techniques have proven to enhance the reliability of the system in several studies [2], [20]–[27], deep sequencing also has certain drawbacks and limitations. Namely, it is a resource-intensive process incurring high read costs, and the redundancy it provides is beneficial for correcting only sequencing errors. Thus, designing edit-correcting codes that are practical as inner codes in DNA storage remains an intriguing area for exploration.

Motivated by the need to ensure reliability and reduce read costs in DNA storage, we introduce a binary code designed to correct random edit errors, while also being practical for the typical short lengths in DNA storage. The code construction is inspired by the Guess & Check (GC) code, initially introduced in [28], for correcting only deletions. We improve and generalize the previous code design by integrating novel encoding and decoding strategies to simultaneously correct deletions, insertions, and substitutions. Our main contributions and the structure of this paper are summarized as follows. We introduce notation in Section II. In Section III, we present the GC+ code, a systematic binary code capable of correcting edit errors at short lengths suitable for DNA storage applications. We detail the encoding and decoding procedures, discuss code properties, and elaborate on various construction components. In Section IV, we explain how the binary GC+ code can be integrated as an inner code in a typical DNA storage framework. We implement and evaluate the decoding performance of the GC+ code; simulation results are provided in Section V. These results show that the code can correct i.i.d. edits with error rates of up to $1\%$, with a code rate $R > 0.5$, message length $k = 133$, and a low probability of decoding

error. Furthermore, our analysis demonstrates that the code is particularly efficient for localized/burst edits, which were shown to be prevalent in DNA storage according to a statistical study on real experimental data [17]. For these types of edit errors, the code can efficiently handle edit error rates of up to $10\%$, with fast and almost error-free decoding. Additionally, we assess the performance of our code within a DNA storage framework and observe that error-free file retrieval can be achieved by combining an inner GC+ code with a high-rate outer Reed-Solomon code, while using only one read per oligo.

## II. Notation

Let $[n] \triangleq \{1, 2, \ldots, n\}$ be the set of integers from 1 to $n$, and $[i, j] \triangleq \{i, i+1, \ldots, j\}$ denote the set of integers from $i$ to $j \geq i$. Let $\mathbb{F}_q$ be the Galois field of size $q$. Bold letters represent vectors, where lowercase $\boldsymbol{x}$ denotes a binary vector and uppercase $\boldsymbol{X}$ denotes a vector in a larger field. We use superscripts to index vectors as $\boldsymbol{x}^i$ and subscripts to index elements within a vector as $x_i$. For a vector $\boldsymbol{x}$, $\boldsymbol{x}_{[i,j]} = (x_i, x_{i+1}, \ldots, x_j)$ represents the substring containing the consecutive bits indexed by $[i, j]$. We use $\langle \boldsymbol{x}^1, \boldsymbol{x}^2 \rangle$ to refer to the concatenation of two vectors $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$. Let $\mathbf{1}^i$ and $\mathbf{0}^j$ denote strings of $i$ consecutive ones and $j$ consecutive zeros, respectively. The $p$-norm of a vector is denoted by $\|\boldsymbol{x}\|_p$, with $p \geq 1$. Additionally, we use $\|\boldsymbol{x}\|_0$ to refer to the number of non-zero elements in $\boldsymbol{x}$. We define $\mathrm{sgn}(\alpha)$ as the function that returns the sign of a real number $\alpha$, with $\mathrm{sgn}(0) = +1$ by convention. All logarithms in this paper are of base 2. Throughout the paper, we use the term *indel* to refer to a single deletion or insertion, and *edit* to refer to a single deletion, insertion, or substitution.

## III. Binary GC+ Code

### A. Encoding

Consider a binary information message $\boldsymbol{u} \in \mathbb{F}_2^k$ of length $k$. Let $\mathsf{Enc} : \mathbb{F}_2^k \to \mathbb{F}_2^n$ be the encoding function that maps the message $\boldsymbol{u}$ to its corresponding codeword $\boldsymbol{x} \in \mathbb{F}_2^n$ of length $n$. The encoding process $\boldsymbol{x} = \mathsf{Enc}(\boldsymbol{u})$ involves the following steps:

1) The message $\boldsymbol{u}$ is segmented into $K \triangleq \lceil k/\ell \rceil$ adjacent substrings of length $\ell$ each, denoted by $\boldsymbol{u}^i \in \mathbb{F}_2^\ell$, where $i \in [K]$ and $\boldsymbol{u} = \langle \boldsymbol{u}^1, \boldsymbol{u}^2, \ldots, \boldsymbol{u}^K \rangle$. Let $U_i \in \mathbb{F}_q$ be the $q$-ary representation of $\boldsymbol{u}^i \in \mathbb{F}_2^\ell$ in $\mathbb{F}_q$, with $q \triangleq 2^\ell$, and let $\boldsymbol{U} \triangleq (U_1, U_2, \ldots, U_K) \in \mathbb{F}_q^K$.[1]
2) The string $\boldsymbol{U} \in \mathbb{F}_q^K$ is encoded using an $(N, K)$ systematic Reed-Solomon (RS) code over $\mathbb{F}_q$, with $N = K + c \leq q$, where $c$ is a code parameter representing the number of redundant parity symbols. The resulting string is denoted by $\boldsymbol{X} = (X_1, X_2, \ldots, X_N) \in \mathbb{F}_q^N$, where $X_{K+1}, X_{K+2} \ldots, X_N$ are the parity symbols.
3) Let $\boldsymbol{p} = \langle \boldsymbol{p}^1, \boldsymbol{p}^2, \ldots, \boldsymbol{p}^c \rangle \in \mathbb{F}_2^{c\ell}$ represent the concatenated binary representation of the $c$ RS parity symbols $X_{K+1}, X_{K+2}, \ldots, X_N$. These parity bits undergo additional encoding using a function $f : \mathbb{F}_2^{c\ell} \to \mathbb{F}_2^{c\ell + r_f}$, wherein $f$ introduces a redundancy $r_f$ that enables the detection and/or correction of edit errors in some or all of the parity bits. Specific choices for the function $f$ are discussed in Section III-E. The encoded parity bits $f(\boldsymbol{p})$ are appended to $\boldsymbol{u}$ to form $\boldsymbol{x}$. Therefore, the codeword $x \in \mathbb{F}_2^n$ is given by
$$\boldsymbol{x} = \mathsf{Enc}(\boldsymbol{u}) = \langle \boldsymbol{u}, f(\boldsymbol{p}) \rangle,$$
with $n = k + c\ell + r_f$.

[1] If the last substring $\boldsymbol{u}^K$ has length $\ell' < \ell$, the computation of $U_K$ assumes padding with zeros in the $\ell - \ell'$ most significant bit positions.

### B. Decoding

Suppose $\boldsymbol{x} \in \mathbb{F}_2^n$ is affected by edit errors, resulting in a string of length $n'$ denoted by $\boldsymbol{y} \in \mathbb{F}_2^{n'}$. Let $\Delta \triangleq n - n'$ be the number of *net* indels in $\boldsymbol{y}$. Define $\mathsf{Dec} : \mathbb{F}_2^{n'} \to \mathbb{F}_2^k \cup \{\text{Fail}\}$ as the decoding function, which either outputs an estimate of the message $\hat{\boldsymbol{u}} \in \mathbb{F}_2^k$ or declares a decoding failure.[2]

The decoding process $\hat{\boldsymbol{u}} = \mathsf{Dec}(\boldsymbol{y})$ employs a Guess & Check mechanism. In this process, a portion of the RS parities is used to generate guesses on $\boldsymbol{u}$, while the remaining parities are used to check the validity of these guesses. Each of the two parts of the parities serves a different function, denoted as $\boldsymbol{p}_G \triangleq \langle \boldsymbol{p}^1, \ldots, \boldsymbol{p}^{c_1} \rangle$ for the first $c_1$ parities used to generate the guesses, and $\boldsymbol{p}_C \triangleq \langle \boldsymbol{p}^{c_1+1}, \ldots, \boldsymbol{p}^c \rangle$ for the remaining $c_2 = c - c_1$ parities used for checking the validity of the guesses. The first step of the GC+ decoder is to leverage the redundancy introduced by the parity encoding function $f$ to either: *(i)* Detect that the information bits are error-free and conclude decoding; or *(ii)* Retrieve the check parities $\boldsymbol{p}_C$ and initiate the Guess & Check process, as described next. Further elaboration on strategies to detect errors or retrieve $\boldsymbol{p}_C$ is deferred to Section III-E. The subsequent discussion assumes the successful recovery of the check parities $\boldsymbol{p}_C$.

The following offers a high-level overview of the Guess & Check process. Let $\boldsymbol{y}' \triangleq \boldsymbol{y}_{[1:n'']}$ denote the first $n'' \triangleq k + c_1\ell + \Delta$ bits of $\boldsymbol{y}$. Based on the value of $\Delta$, the decoder makes a guess about the locations and number of *net* indels within the $N' \triangleq K + c_1$ segments corresponding to $\boldsymbol{y}'$. Subsequently, $\boldsymbol{y}'$ is segmented according to this guess. Specifically, each substring $i \in [N']$ is segmented to a length of $\ell + \delta_i$, where the value of $\delta_i$ follows from the guess. The outcome of this segmentation is decoded using the $(N', K)$ RS code punctured at the first $N'$ positions, with the decoder taking the $q$-ary representations of segments presumed to have zero *net* indels as input while treating the remaining segments as hypothetical symbol erasures over $\mathbb{F}_q$. If the output of the RS decoder is consistent with the $c_2$ check parities corresponding to $\boldsymbol{p}_C$, the guess is validated and the decoder outputs a final estimate $\hat{\boldsymbol{u}}$; otherwise, it proceeds to the next guess. If no valid estimate is obtained after processing all intended guesses, the decoder declares a decoding failure. A more rigorous description of this process is provided below.

A guess involves assuming a specific distribution of the $\Delta$ *net* indels among the $N' = K + c_1$ segments corresponding to $\boldsymbol{y}'$. More precisely, each guess corresponds

[2] Here, the term "decoding failure" denotes a detectable decoding error, indicating that the decoder acknowledges its inability to decode $\boldsymbol{y}$.

to a segmented indel pattern $\boldsymbol{\delta}$, represented as a vector $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{N'}) \in \mathbb{Z}^{N'}$, where $\delta_i$ indicates the number of *net* indels in segment $i \in [N']$ and $\sum_{i=1}^{N'} \delta_i = \Delta$. Given a pattern $\boldsymbol{\delta}$, $\boldsymbol{y}'$ is segmented into $N'$ adjacent binary substrings $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^{N'}$, where the length of $\boldsymbol{y}^i$ is $\ell + \delta_i$, for all $i \in [N']$. For a given $\boldsymbol{\delta}$, let $\boldsymbol{Y} = (Y_1, \ldots, Y_{N'}) \in \mathbb{F}_q^{N'}$ with

$$Y_i = \begin{cases} (\boldsymbol{y}^i)_{\mathbb{F}_q}, & \text{if } \delta_i = 0, \\ \mathcal{E}, & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathcal{E}$ denotes an erasure and $(\boldsymbol{y}^i)_{\mathbb{F}_q}$ is the $q$-ary representation of $\boldsymbol{y}^i$ in $\mathbb{F}_q$. The string $\boldsymbol{Y}$ is decoded using the punctured $(N', K)$ RS code to obtain $\hat{\boldsymbol{U}} \in \mathbb{F}_q^K$. Here, we consider syndrome-based RS decoder implementations capable of correcting all combinations of $e$ erasures and $s$ substitutions, provided that $e + 2s \leq N' - K = c_1$ [29], [30]. The decoder then checks if $\hat{\boldsymbol{U}}$ is consistent with the $c_2$ parity symbols corresponding to $\boldsymbol{p}_C$. If the guess is valid, $\hat{\boldsymbol{u}}$ (the binary equivalent of $\hat{\boldsymbol{U}}$) is returned, and decoding is terminated; otherwise, the decoder proceeds to make another guess by considering a different pattern $\boldsymbol{\delta}$.

Next, we outline the sequential steps taken by the GC+ decoder and explicitly define the patterns considered in these steps. If $\Delta = 0$, the decoder initiates a *fast check*, examining the singular pattern $\boldsymbol{\delta} = \boldsymbol{0}^{N'}$, as per the previously explained procedure. If $\Delta \neq 0$, the decoder runs a *primary check*, where it investigates patterns associated with scenarios where all edit errors are localized within $c_1$ consecutive segments. Namely, the decoder considers the set of patterns $\mathcal{P}_1$ defined by

$$\mathcal{P}_1(\Delta, N', c_1) \triangleq \bigcup_{j=1}^{N'-c_1+1} \mathcal{P}_1^j(\Delta, N', c_1),$$

$$\mathcal{P}_1^j(\Delta, N', c_1) \triangleq \left\{ \boldsymbol{\delta} \in \mathbb{Z}^{N'} : \sum_{i \in \mathcal{I}_j} \delta_i = \Delta, \delta_i = 0 \ \forall i \notin \mathcal{I}_j \right\},$$

where $\mathcal{I}_j \triangleq \{j, j+1, \ldots, j+c_1-1\}$. Notice that due to the localized nature of errors examined in this phase, for a given $j$, processing any single pattern in $\mathcal{P}_1^j$ suffices, as all patterns in $\mathcal{P}_1^j$ lead to the same guess. Consequently, in the *primary check*, the decoder processes a total of $N' - c_1 + 1 = K + 1$ patterns in an arbitrary order. If the decoding is not terminated during the *fast* or *primary check*, the decoder proceeds to an optional *secondary check*, examining a more exhaustive set of patterns $\mathcal{P}_2$ defined by

$$\mathcal{P}_2(\Delta, N', c_1, \lambda) \triangleq \Bigg\{ \boldsymbol{\delta} \in \mathbb{Z}^{N'} : \|\boldsymbol{\delta}\|_0 \leq \min\{|\Delta| + 2\lambda, c_1\},$$
$$\sum_{i=1}^{N'} \delta_i = \Delta, \ \ -\lambda \leq \operatorname{sgn}(\Delta)\delta_i \leq |\Delta| + \lambda \ \forall i \in [N'] \Bigg\},$$

where $\lambda \geq 0$ is a configurable decoding parameter that we call *decoding depth*. Note that $\mathcal{P}_2$ may contain patterns that were already examined in the phases preceding the *secondary check*; these patterns are thus omitted. Furthermore, in this phase, the patterns are processed in increasing order of $\|\boldsymbol{\delta}\|_1$ norm. In summary, the decoder sequentially executes the following steps, with each step performed only if the preceding steps did not yield a valid estimate $\hat{\boldsymbol{u}}$:

1) The parity encoding function $f$ is utilized to either detect that the information bits are error-free or retrieve the check parities $\boldsymbol{p}_C$ and start the Guess & Check process.
2) If $\Delta = 0$, the *fast check* is employed; else, if $\Delta \neq 0$, the *primary check* is initiated.
3) If the option to run a *secondary check* is active, it is executed.
4) If no valid estimate $\hat{\boldsymbol{u}}$ is obtained in the preceding steps, a decoding failure is declared.

### C. Code Properties

*1) Code Rate:* The redundancy is $n - k = (c_1 + c_2)\ell + r_f$, and hence the code rate is

$$R = \frac{k}{n} = 1 - \frac{(c_1 + c_2)\ell}{n} - \frac{r_f}{n}.$$

*2) Time Complexity:* We begin by discussing the encoding complexity of the $q$-ary $(N, K)$ RS code and the decoding complexity of the punctured $(N', K)$ RS code. The encoding complexity of the $(N, K)$ code, utilizing basic polynomial multiplication, is $\mathcal{O}(K(N-K)) = \mathcal{O}((c_1+c_2)K)$, and the decoding complexity of the $(N', K)$ code, employing syndrome-based decoding, is $\mathcal{O}(N'(N'-K)) = \mathcal{O}(c_1 K + c_1^2)$ [30]. For these encoding and decoding approaches, the constants hidden by the $\mathcal{O}$-notation are small, making them efficient for short codes. For a comprehensive non-asymptotic analysis of these complexities, we refer interested readers to [31]. Furthermore, these complexities are primarily influenced by the number of multiplications in $\mathbb{F}_{2^\ell}$; thus, the bit complexities for encoding and decoding include an additional factor of $\mathcal{O}(\log^2(2^\ell)) = \ell^2$. Recall that $K = \lceil k/\ell \rceil$, resulting in bit encoding and decoding complexities of $\mathcal{O}((c_1 + c_2)\ell k)$ and $\mathcal{O}(c_1 \ell k + c_1^2 \ell^2)$, respectively.

Next, we analyze the encoding and decoding complexities of the GC+ code. We assume that the order of these complexities remains unaffected by the operations related to the parity encoding function $f$. This assumption holds true for all practical purposes, as we later discuss in Section III-E. The encoding complexity is dominated by the generation of the $c = c_1 + c_2$ RS parities, resulting in $\mathcal{O}((c_1 + c_2)\ell k)$. On the decoding side, the complexity is dominated by the process of generating guesses, computed as the product of the total number of guesses and the complexity of the $(N', K)$ RS decoder. Thus, the worst-case complexity of the second decoding step, involving either the *fast check* or *primary check*, is $(K+1)\mathcal{O}(c_1 \ell k + c_1^2 \ell^2) = \mathcal{O}(c_1 k^2 + c_1^2 \ell k)$. If the option to run a *secondary check* is active, an upper bound on the worst-case decoding complexity is given by $|\mathcal{P}_2(\Delta, N', c_1, \lambda)| \cdot \mathcal{O}(c_1 \ell k + c_1^2 \ell^2)$, where $\mathcal{P}_2$ is the set of patterns defined in Section III-B. In Claim 1, we provide an expression that determines the exact value of $|\mathcal{P}_2|$ in terms of $\Delta, N', c_1$, and $\lambda$, based on the inclusion-exclusion principle. The proof of this claim is omitted.

*Claim 1:* The size of $\mathcal{P}_2(\Delta, N', c_1, \lambda)$ is given by (2).

$$|\mathcal{P}_2(\Delta, N', c_1, \lambda)| = \sum_{i_1=0}^{I_1} (-1)^{i_1} \binom{N'}{i_1} \sum_{i_2=i_1}^{I_2} \binom{N'-i_1}{N'-i_2} \sum_{i_3=0}^{I_3} (-1)^{i_3} \binom{i_2-i_1}{i_3} \binom{|\Delta| + (i_2-i_3)(\lambda+1) - i_1(|\Delta|+2\lambda+1) - 1}{i_2 - i_3 - 1},$$

$$I_1 = \left\lfloor \frac{|\Delta| + N'\lambda}{|\Delta| + 2\lambda + 1} \right\rfloor, \qquad I_2 = \min\{|\Delta| + 2\lambda, c_1\}, \qquad I_3 = \min\left\{ \left\lfloor \frac{|\Delta| + i_2\lambda - i_1(|\Delta|+2\lambda+1)}{\lambda} \right\rfloor, i_2 - i_1 \right\}. \quad (2)$$

It is important to note that since decoding terminates upon finding a valid guess, depending on the underlying random edit error model, the average-case decoding complexity can be significantly lower than the worst-case scenario. The patterns in the *secondary check* are processed in increasing order of $\|\boldsymbol{\delta}\|_1$ to improve the average-case decoding time, assuming that fewer errors are more probable in the underlying error model.

*3) Error Correction Capability:* The edit error correction capability of the GC+ decoder at the bit level stems from the erasure and substitution correction capability of the RS code at the $q$-ary level. Specifically, for a given decoder input $\boldsymbol{y}'$ and a guess parameterized by a pattern $\boldsymbol{\delta}$, the edit errors in $\boldsymbol{y}'$ can be corrected if $e + 2s \leq c_1$, where $e$ and $s$ denote the number of erasures and substitutions, respectively, in $\boldsymbol{Y}$ (defined in (1)). Two sources contribute to the possibility of a decoding error: *(i)* An *undetectable* decoding error, which results from a spurious guess, producing an incorrect estimate $\hat{\boldsymbol{U}}$ that accidentally aligns with the $c_2$ check parities. *(ii)* A *detectable* decoding error (*decoding failure*), which occurs when none of the guesses yield a valid estimate, indicating either that the bit-level edit error combination exceeds the error correction capability of the $q$-ary RS code, or that the actual *net* indel pattern was not covered during the Guess & Check process. The probability of an *undetectable* decoding error is influenced by the value of $c_2$, while the probability of a *decoding failure* depends on $c_1$, the underlying random error model, and other encoding/decoding parameters.

Next, we elaborate on the nature of edit errors that can be corrected during different phases of the decoding process. The *fast check* is applied when $\Delta = 0$, where the decoder examines only the all-zeros pattern. For $\boldsymbol{\delta} = \mathbf{0}^{N'}$, the string $\boldsymbol{Y}$ obtained from (1) does not contain any marked erasures. Consequently, decoding is successful as long as bit-level edit errors result in $\tau \leq \lfloor c_1/2 \rfloor$ symbol substitutions in $\boldsymbol{Y}$. This condition allows for a wide range of possibilities regarding the positions, types, and total number of correctable edit errors at the bit level. For instance, it covers scenarios with any number of edit errors occurring in $\tau \leq \lfloor c_1/2 \rfloor$ out of the $N' = K + c_1$ binary segments, contingent on the number of *net* indels in each of these $\tau$ segments being zero.

The *primary check* covers all cases with $\Delta \neq 0$ that correspond to scenarios where the edit errors affect at most $c_1$ consecutive segments. Similar to the *fast check*, the *primary check* addresses a variety of bit-level edit error scenarios, but without the constraint of having zero *net* indels in each segment. For example, it covers all types and numbers of edit errors, provided that the locations of these errors are confined to any $(c_1 - 1)\ell$ consecutive bit positions. This includes cases of burst or localized edit errors, with an arbitrary number of errors occurring within a window of size $(c_1 - 1)\ell$. The reasoning extends similarly to the *secondary check*, which covers a broader spectrum of edit error scenarios while leveraging the ability of RS codes to simultaneously correct erasures and substitutions.

### *D. Choice of the code parameters*

The configurable encoding and decoding parameters of the GC+ code are summarized below.

| Parameter | Description |
|---|---|
| $\ell$ | segmentation length for encoding |
| $c_1$ | number of RS parities for guessing |
| $c_2$ | number of RS parities for checking |
| $\lambda$ | decoding depth in *secondary check* |

The choice of the value of $\ell$ presents a trade-off between the redundancy introduced by the RS code, given by $(c_1 + c_2)\ell$, and the decoding complexity which depends on the number of segments $K = \lceil k/\ell \rceil$. Ideally, we seek to minimize $\ell$ to reduce redundancy. However, selecting a small value of $\ell$ poses the following challenges: *(i)* The increase in the number of segments $K$ requires the decoder to process a larger number of guesses, leading to increased decoding complexity. *(ii)* The RS code imposes the condition $q \geq K + c_1 + c_2$, implying $2^\ell \geq \lceil k/\ell \rceil + c_1 + c_2$, which could be violated for small $\ell$. A typical choice for $\ell$ is $\ell = \lfloor \log k \rfloor$ as it maintains low redundancy and gives a good trade-off between the code properties; however, higher values of $\ell$ may be considered based on specific application requirements.

Concerning the choice of $c_1$, it primarily depends on the desired level of error correction. It can be adapted to the specific error model and application to achieve suitable trade-offs between redundancy and decoding failure rates. The parameter $\lambda$ affects decoding complexity in the context of the *secondary check*. We typically opt for small values of $\lambda$ to strike a balance between decoding complexity and the probability of decoding failure. The value of $\lambda$ could also be customized according to the number of *net* indels $\Delta$, where, for example, instances with $|\Delta| = j$ can be decoded with depth $\lambda_j$. Lastly, the choice of $c_2$ mainly influences the probability of undetectable decoding errors. For short codes, our simulations indicate that very small values of $c_2$ suffice to maintain a low probability of such errors.

### E. Parity encoding function

As previously mentioned, the purpose of the parity encoding function $f$ is to detect or correct edit errors in the parities $\boldsymbol{p} = \langle \boldsymbol{p}_G, \boldsymbol{p}_C \rangle$. More precisely, we are primarily interested in "protecting" the check parities $\boldsymbol{p}_C$ as their accurate recovery is vital for initiating the Guess & Check process outlined in Section III-B. On the other hand, errors within the guess parities $\boldsymbol{p}_G$ are implicitly addressed as part of the Guess & Check process. As discussed in the previous section, for typical values of the code parameters such as $\ell = \log k$ and small $c_2$, the length of the check parities $c_2\ell$ is relatively short compared to the information sequence. Thus, we can afford to encode these parities with certain codes that might be inefficient for longer lengths in terms of rate or complexity. Furthermore, for specific types of errors, such as burst or localized errors, it is possible to set up $f$ in a way that the errors impact either the information bits or the parities, but not both. In addition, $f$ can be designed to enable the detection of which of these two cases actually occurred. In the following, we present a non-exhaustive list of possibilities for selecting the parity encoding function $f$.

*1) Repetition code:* The check parities can be encoded using a $(t+1)$-repetition code, where each bit is repeated $t+1$ times, i.e., $f(\boldsymbol{p}_G, \boldsymbol{p}_C) = \langle \boldsymbol{p}_G, \mathsf{rep}_{t+1}(\boldsymbol{p}_C) \rangle$. At the decoder, the check parities $\boldsymbol{p}_C$ are recovered by taking the majority vote in each block of size $t+1$ in $\boldsymbol{y}_{[n'-(t+1)c_2\ell:n']}$, where $\boldsymbol{y}$ is the decoder input defined in Section III-B. Then, as previously explained, the Guess & Check process is applied over $\boldsymbol{y}_{[1:k+c_1\ell+\Delta]}$. The redundancy incurred by the repetition code is $r_f = tc_2\ell$, resulting in an overall redundancy of $n-k = (c_1+(t+1)c_2)\ell$. The impact of the repetition code on the overall encoding and decoding complexity is negligible.

*2) Brute force:* In theory, protecting the check parities with significantly less redundancy than the repetition code is possible through the use of brute-force decoding methods. In general, these methods have exponential time complexity. However, when applied to short inputs, such as the check parities with $\ell = \log k$ and small $c_2$, the complexity remains within polynomial limits in terms of $k$. One potential approach involves using a hash function based on graph colorings, similar to the one outlined in [6, Lemma 2]. However, even for short lengths, this approach proves cumbersome in practice.

*3) Buffer:* Suppose that the edit errors are localized within any window of $w < n$ consecutive bit positions, where the location of the window is unknown to the decoder but its size is known. Then, it is possible to insert a buffer between the information and parity bits to achieve the following: *(i)* Ensure that edit errors cannot simultaneously affect the information and parity bits. *(ii)* Enable the detection of whether edit errors have affected the information bits or not when $\Delta \neq 0$. Recent works [17], [32], [33] have explored such buffers. Following [17, Lemma 17], we adopt the buffer $\boldsymbol{b} \triangleq \langle \mathbf{1}^{w+1}, \mathbf{0}^{w+1}, \mathbf{1}^{w+1} \rangle$ of length $3(w+1)$, and set the parity encoding function as $f(\boldsymbol{p}_G, \boldsymbol{p}_C) = \langle \boldsymbol{b}, \boldsymbol{p}_G, \boldsymbol{p}_C \rangle$. If $\Delta \neq 0$, the decoder utilizes the buffer to either simply output the error-free information bits or use the error-free parities to initiate the *primary check*. Here, the decoding is simplified by fixing the values of $\boldsymbol{p}_G$ throughout the Guess & Check process and operating over $K$ segments instead of $K + c_1$. Moreover, for appropriate choices of the code parameters, as discussed in Section V, the need for the *secondary check* can be entirely eliminated. Note that the buffer is ineffective when $\Delta = 0$. Thus, in such cases, the *fast check* involves discarding the buffer bits and passing the remaining data to the RS decoder to correct substitutions. The overall redundancy in this scenario is $n - k = (c_1 + c_2)\ell + 3(w+1)$, and the impact of $f$ on the encoding/decoding complexity of the GC+ code is negligible.

## IV. Application to DNA Storage

Consider the common process of encoding arbitrary binary data into DNA illustrated in Fig. 1. Due to the limitations of state-of-the-art DNA synthesis technologies, the binary data is generally encoded in the form of several short DNA sequences, called oligos, typically a few hundred nucleotides long. This is accomplished by segmenting the binary file into non-overlapping fragments and then transcoding each binary fragment into a sequence of nucleotides $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Also, metadata is added to each fragment, including indexes for the oligos, to facilitate the reconstruction of the original file from these fragments during data retrieval. Amid this workflow, both an outer and an inner error-correction code can be integrated to provide robustness against errors that may occur during various stages of the DNA storage process. The inner code is applied to each individual fragment, requiring a short code capable of correcting errors within each oligo. On the other hand, the outer code is applied over the collection of fragments, aiming to recover lost oligos or correct residual errors from the inner code.

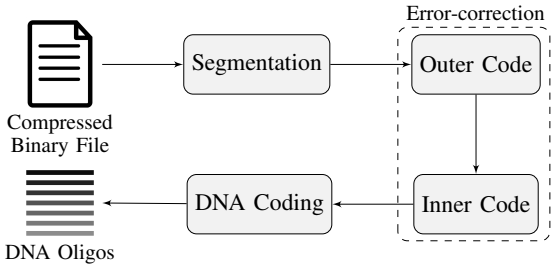


Fig. 1. Encoding a binary file into DNA oligos with error-correction

We propose integrating the GC+ code as the inner code in the workflow depicted in Fig. 1. Intuitively, we argue that the GC+ code is a suitable choice for the inner code since: *(i)* It can effectively correct edit errors at short code lengths. *(ii)* It contributes to reducing the read costs by minimizing the required sequencing depth. *(iii)* It addresses indels occurring during synthesis and storage, which are beyond the scope of alignment algorithms. *(iv)* Its ability to acknowledge decoding failures can be leveraged by the outer code to treat the lost data as erasures rather than substitutions, thereby optimizing the performance of the outer code.

In the next section, we substantiate our intuition through numerical simulations. In these simulations, we focus on unconstrained DNA transcoding (2 bits per nucleotide) given by the mapping: $00 \leftrightarrow$ A, $01 \leftrightarrow$ C, $10 \leftrightarrow$ G, $11 \leftrightarrow$ T. The investigation of scenarios involving constrained DNA coding is deferred to future work. It is important to note that under the unconstrained mapping, every edit error in the DNA sequence equates to up to two consecutive errors in the corresponding binary sequence. Furthermore, to assess the achievable error-correction limits with minimal read costs, we assume that only one read per oligo is available for the data retrieval process.

## V. Simulations

### A. Error Model & Setup

We evaluate the error correction capability of the GC+ code for both the binary and DNA cases over the following random channel model. Let $\boldsymbol{x} \in \Sigma^{n_1}$ and $\boldsymbol{y} \in \Sigma^{n_2}$ be input and output of the channel, respectively, where $\Sigma$ denotes the alphabet. Each symbol in $\boldsymbol{x}_{[i:i+w-1]}$ is edited independently with probability $P_{edit}$, where $i$ is sampled uniformly at random from $\{1, 2, \ldots, n_1 - w + 1\}$ and $w \leq n_1$ is a channel parameter. All the symbols in $\boldsymbol{x}_{[1:i-1]}$ and $\boldsymbol{x}_{[i+w:n_1]}$ are retained. Given that a symbol is edited, the conditional probabilities of deletion, insertion, and substitution are represented by $P_d$, $P_i$, and $P_s$, respectively, with $P_{edit} = P_d + P_i + P_s$. For a given input symbol $x$, the output of the channel $y$ for the three types of errors is described as follows: if $x$ is deleted, the output $y$ is an empty string; if $x$ is affected by an insertion, then $y = \langle \sigma, x \rangle$ where $\sigma$ is chosen uniformly at random from $\Sigma$; if $x$ is substituted, then $y = \tilde{x}$, where $\tilde{x}$ is chosen uniformly at random from $\Sigma \setminus x$. Thus, the channel is characterized by the parameters $(w, n_1, P_{edit}, P_d, P_i, P_s)$, and under this model, we have $|n_1 - n_2| \leq n_1$. The average edit rate introduced by this channel, denoted by $\varepsilon_{av}$, is given by $\varepsilon_{av} = P_{edit} \times \frac{w}{n_1}$.

Note that the case of $w = n_1$ reduces to the scenario of i.i.d. edits, with $\varepsilon_{av} = P_{edit}$. In our simulations, we study the case of $w < n_1$ with high values of $P_{edit}$ to simulate scenarios with burst/localized edits, which are prevalent in DNA storage [17]. Additionally, we also investigate the case where $w = n_1$ with lower values of $P_{edit}$ to model scenarios with i.i.d. edits. The simulations are conducted on synthetic data, where in each run the input message/file is generated uniformly at random. The results on the probability of decoding error are averaged over a series of independent runs. The simulations were implemented in MATLAB using tools from the MATLAB Communications Toolbox [34].

### B. Binary case

We simulated the overall probability of decoding error (decoding failures + undetectable decoding errors) of the binary GC+ code, as a standalone, for the message length $k = 133$, with the segmentation parameter set to $\ell = \lfloor \log k \rfloor = 7$.

*1) IID edits:* We studied the values of $P_{edit}$ ranging from $0.1\%$ to $1\%$, with either $P_s = P_d = P_i$ or $P_s = 2P_d = 4P_i$. The number of parities of the GC+ code was set to $(c_1, c_2) = (8, 2)$. The $(t + 1)$ repetition code was used as the parity encoding function, with $t = 2$ resulting in a $(231, 133)$ code with rate $R \approx 0.58$, and also with $t = 4$ resulting in a $(259, 133)$ code with $R \approx 0.51$. The results, given in Fig. 2, show that the code can effectively correct i.i.d. edits with average error rates of up to $\varepsilon_{av} = 1\%$, while maintaining code rates $R > 0.5$. Also, for fixed $P_{edit}$, the decoding performance is influenced by the conditional probabilities $(P_d, P_i, P_s)$, with a higher probability of decoding error observed for equiprobable edits.

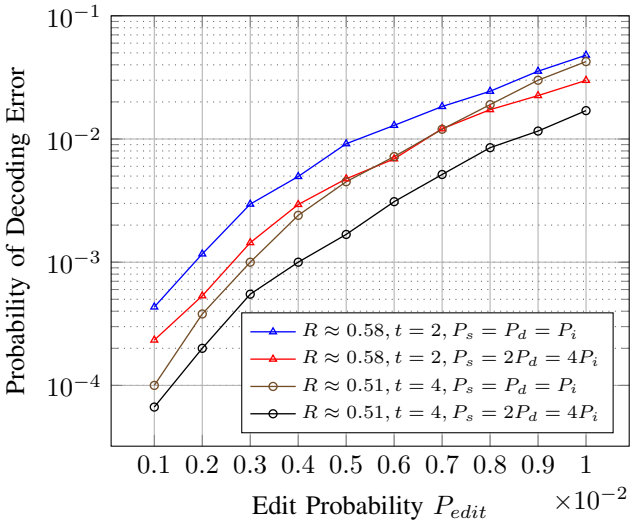


Fig. 2. Empirical probability of decoding error of binary GC+ code versus i.i.d. edit probabilities for message length $k = 133$. The code parameters are set to $\ell = \lfloor \log k \rfloor$, $(c_1, c_2) = (8, 2)$, $t \in \{2, 4\}$, $\lambda = 1$ for $|\Delta| \leq 1$ and $\lambda = 0$ otherwise. The results are averaged over $5 \times 10^4$ independent runs.

*2) Localized edits:* We studied the values of $w < n$ varying between $\ell + 1 = 8$ and $4\ell + 1 = 29$, while fixing $P_{edit} = 99\%$, with $P_s = P_d = P_i$. The number of parities is set to $c_1 = c_2 = (w-1)/\ell + 1$. The parity encoding function based on the buffer $\boldsymbol{b}$ of length $3(w + 1)$ defined in Section III-E3 was used. The resulting code rate $R$ varies based on $w$. The results presented in Fig. 3 demonstrate that for localized edits, the GC+ code can handle significantly higher edit rates $\varepsilon_{av}$ compared to the i.i.d. case, with reduced decoding error probability and often higher code rates. Moreover, the *secondary check* decoding phase is omitted for this simulation, leading to faster decoding.

| Window length | Avg. edit rate | Code rate | Prob. error |
|---|---|---|---|
| $w = 8$ | $4.2\%$ | $0.71$ | $= 2.5e^{-4}$ |
| $w = 15$ | $6.7\%$ | $0.60$ | $< 1.0e^{-5}$ |
| $w = 22$ | $8.4\%$ | $0.52$ | $< 1.0e^{-5}$ |
| $w = 29$ | $9.8\%$ | $0.45$ | $< 1.0e^{-5}$ |

Fig. 3. Empirical probability of decoding error of binary GC+ code for localized edits with varying window length $w$ (channel parameter). Edit probability is fixed at $P_{edit} = 99\%$, with $P_s = P_d = P_i$. Code parameters: $k = 133$, $\ell = \lfloor \log k \rfloor$, $c_1 = c_2 = (w - 1)\ell + 1$. The average edit rate $\varepsilon_{av}$ and code rate $R$ are reported for each value of $w$. Buffer-based parity encoding is used, and the *secondary check* is omitted. Results are averaged over $10^5$ independent runs. Entries with $< 1.0e^{-5}$ indicate no decoding errors recorded in $10^5$ runs.

### C. DNA case

Following the workflow depicted in Fig. 1, we consider a binary file of size 1.6 Mb, segmented into $10^4$ fragments of size 160 bits each. These fragments are encoded using an outer systematic RS code in $\mathbb{F}_{2^{14}}$ of rate $R_{out}$, generating $10^4(R_{out}^{-1}-1)$ additional "parity" fragments of the same size. Each fragment is then encoded using an inner $(304, 160)$ GC+ code with rate $R_{in} \approx 0.53$ and parameters $\ell = 8$, $(c_1, c_2) = (8, 2)$, and $t = 4$. The coded fragments are transcoded into DNA using the basic 2 bits/NT mapping, resulting in a total of $10^4 R_{out}^{-1}$ DNA oligos of length $304/2 = 152$ NTs each. We consider the random edit channel model applied over these DNA oligos, with each nucleotide edit translating to up to two consecutive bit edits in the underlying binary code. We focus on the i.i.d. edits scenario (i.e., $w = 152$), as almost error-free decoding can be achieved for localized edits using only the inner code. To assess the achievable error-correction limits with minimal read costs, we assume only one read per oligo is available for data retrieval, with each oligo implicitly containing metadata facilitating file reconstruction.

Using the $(304, 160)$ GC+ code as the inner code, we evaluated the maximum outer code rate (i.e., minimum redundancy) needed to achieve *error-free* file retrieval. Our analysis relies on the ability of the outer RS code to simultaneously correct fragment erasures and substitutions. Specifically, each decoding failure in the inner GC+ code is treated as a fragment erasure, requiring *one* error-free parity fragment for recovery. Meanwhile, every undetectable decoding error in the inner code translates to a fragment substitution, requiring *two* error-free parity fragments for correction. The findings presented in Fig. 4 indicate that for i.i.d. nucleotide edit rates of up to $1\%$, error-free file retrieval is achievable with high-rate outer RS codes, given only one read per oligo.

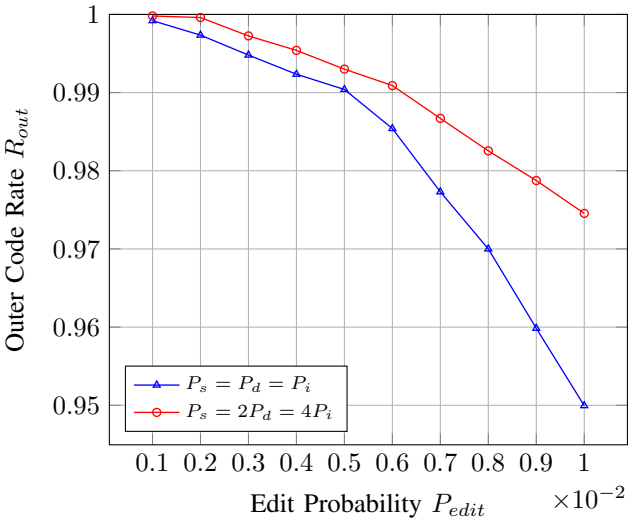


Fig. 4. Rate of outer RS code to achieve error-free file retrieval versus edit probabilities. The inner code is a binary $(304, 160)$ GC+ code. The file of size 1.6 Mb is encoded into $10^4 R_{out}^{-1}$ DNA oligos of length 152 NTs each.

## VI. Conclusion

In this work, we introduced the GC+ code, a systematic binary code designed to correct edit errors at short code lengths suitable for DNA storage applications. In the context of DNA storage, we focused on assessing how far we can push error correction limits with minimal read costs, i.e., one read per oligo. Some interesting directions for future research include:

1) Improving the decoding complexity of the GC+ code, particularly during the *secondary check* phase. In [35], the authors provided valuable insights on achieving fast decoding with the original version of Guess & Check codes [36] using maximum-likelihood inference of deletion patterns on trellis graphs. It would be interesting to investigate similar approaches for edit correction.
2) Deriving theoretical bounds on the probability of decoding error of the GC+ code.
3) Considering other mathematical edit error models or noise simulators for *in silico* studies, in addition to conducting *in vitro* experiments on real DNA data.
4) Extending the results in Section V-C by studying the achievable trade-offs between the rate of the inner GC+ code and the rate of the outer code for error-free file retrieval with one read per oligo.
5) Studying the three-way trade-offs that can be achieved between the inner code rate, outer code rate, and sequencing depth, for error-free file retrieval with more than one read per oligo. This analysis would give insights on the trade-offs between synthesis and sequencing costs.
6) Exploring constrained DNA coding approaches, such as binary to DNA mappings that restrict the occurrence of homopolymers and repeated patterns, and/or ensure a balanced G/C content.
7) Constructing edit correcting codes over the DNA alphabet.

## References


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

[2] 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.

[3] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the dna data storage channel," *Scientific reports*, vol. 9, no. 1, p. 9663, 2019.

[4] R. Varshamov and G. Tenengol'ts, "Correction code for single asymmetric errors," *Automat. Telemekh*, vol. 26, no. 2, pp. 286–290, 1965.

[5] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

[6] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3403–3410, May 2018.

[7] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3360–3375, 2021.

[8] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 769–774.

[9] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1971–1985, 2017.

[10] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the damerau distance for deletion and adjacent transposition correction," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 2550–2570, 2017.

[11] I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi, "Single-deletion single-substitution correcting codes," *IEEE Transactions on Information Theory*, 2023.

[12] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, "Beyond single-deletion correcting codes: substitutions and transpositions," *IEEE Transactions on Information Theory*, vol. 69, no. 1, pp. 169–186, 2022.

[13] W. Song, N. Polyanskii, K. Cai, and X. He, "Systematic codes correcting multiple-deletion and multiple-substitution errors," *IEEE Transactions on Information Theory*, vol. 68, no. 10, pp. 6402–6416, 2022.

[14] H. Mahdavifar and A. Vardy, "Asymptotically optimal sticky-insertion-correcting codes with efficient encoding and decoding," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2683–2687.

[15] S. Wang, V. K. Vu, and V. Y. F. Tan, "Codes for correcting t limited-magnitude sticky deletions," in *2023 IEEE International Symposium on Information Theory (ISIT)*, 2023, pp. 1148–1153.

[16] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, "Correcting a single indel/edit for dna-based data storage: Linear-time encoders and order-optimality," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3438–3451, 2021.

[17] Y. Tang, S. Motamen, H. Lou, K. Whritenour, S. Wang, R. Gabrys, and F. Farnoud, "Correcting a substring edit error of bounded length," in *2023 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2023, pp. 2720–2725.

[18] M. C. Davey and D. J. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, 2001.

[19] E. A. Ratzer, "Marker codes for channels with insertions and deletions," in *Annales des télécommunications*, vol. 60, no. 1-2. Paris, Societe de la Revue optique., 2005, pp. 29–44.

[20] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[21] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso, B. W. Pruitt, and G. M. Church, "Forward error correction for dna data storage," *Procedia Computer Science*, vol. 80, pp. 1011–1022, 2016.

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

[23] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free dna-based data storage," *Scientific reports*, vol. 7, no. 1, p. 5011, 2017.

[24] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman *et al.*, "Overcoming high nanopore basecaller error rates for dna storage via basecaller-decoder integration and convolutional codes," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8822–8826.

[25] W. H. Press, J. A. Hawkins, S. K. Jones Jr, J. M. Schaub, and I. J. Finkelstein, "Hedges error-correcting code for dna storage corrects indels and allows sequence constraints," *Proceedings of the National Academy of Sciences*, vol. 117, no. 31, pp. 18 489–18 496, 2020.

[26] I. Maarouf, A. Lenz, L. Welter, A. Wachter-Zeh, E. Rosnes, and A. G. i Amat, "Concatenated codes for multiple reads of a DNA sequence," *IEEE Transactions on Information Theory*, vol. 69, no. 2, pp. 910–927, 2023.

[27] B. Hamoum, A. Ezzeddine, and E. Dupraz, "Synchronization algorithms from high-rate ldpc codes for dna data storage," in *2023 24th International Conference on Digital Signal Processing (DSP)*. IEEE, 2023, pp. 1–5.

[28] S. Kas Hanna and S. El Rouayheb, "Guess & check codes for deletions and synchronization," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2693–2697.

[29] G. C. Clark Jr and J. B. Cain, *Error-correction coding for digital communications*. Applications of Communications Theory. New York: Plenum Press, 1981.

[30] T. K. Moon, *Error correction coding: mathematical methods and algorithms*. John Wiley & Sons, 2020.

[31] N. Chen and Z. Yan, "Complexity analysis of reed-solomon decoding over GF($2^m$) without using syndromes," *EURASIP Journal on Wireless Communications and Networking*, vol. 2008, pp. 1–11, 2008.

[32] S. Kas Hanna and S. El Rouayheb, "Codes for correcting localized deletions," *IEEE Transactions on Information Theory*, vol. 67, no. 4, pp. 2206–2216, 2021.

[33] R. Bitar, S. Kas Hanna, N. Polyanskii, and I. Vorobyev, "Optimal codes correcting localized deletions," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1991–1996.

[34] The MathWorks Inc., "Communications Toolbox (R2023b)," Natick, Massachusetts, United States, 2023. [Online]. Available: https://mathworks.com/products/communications.html

[35] G. Ma, X. Jiao, J. Mu, Y.-C. He, and H. Han, "Maximum-likelihood deletion error location for decoding marker guess & check codes," *IEEE Communications Letters*, vol. 25, no. 8, pp. 2497–2501, 2021.

[36] S. Kas Hanna and S. El Rouayheb, "Guess & check codes for deletions, insertions, and synchronization," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 3–15, Jan 2019.