A Segmented-edit error-Correcting Code with Re-synchronization Function for DNA-based Storage Systems

Zihui Yan, Cong Liang, and Huaming Wu, *Senior Member, IEEE*

Abstract—As a powerful tool for storing digital information in chemically synthesized molecules, DNA-based data storage has undergone continuous development and received increasingly more attention. Efficiently recovering information from large-scale DNA strands that suffer from insertions, deletions, and substitution errors (collectively referred to as edit errors), is one of the major bottlenecks in DNA-based storage systems. To cope with this challenge, in this paper, we provide a segmented-edit error-correcting code with the re-synchronization function, termed the *DNA-LM* code. Compared with the previous segmented-error-correcting codes, it has a systematic structure and does not require the endpoint of the received segment as pre-requisite information for decoding. In the case that the number of edit errors exceeds the edit error-correcting capability of a segment, it can easily regain synchronization to ensure that the subsequent decoding continues. Both encoding and decoding complexity is linear in the codeword length. The redundancy of each segment is $\lceil \log k \rceil + 6$ quaternary symbols, where k is the length of the message segment. We further generalize the decoding algorithm to deal with duplicated DNA strands, whereas it still maintains linear time complexity in the codeword length and the number of duplications. Simulations under a stochastic edit errors model show that, at a low raw error rate of the "next-gen" sequencing, our code can enable error-free decoding by concatenating with the (255,223) RS code.

Index Terms—DNA-based storage system, Segmented-edit error-correcting codes, Synchronization errors, VT codes.

1 Introduction

With the rapid development of the Internet of Things (IoT) and intelligent applications, a large amount of new data is collected by different types of sensors on a daily basis [1], which leads to increasing demand for storage systems [2]. Along with the development of Deoxyribonucleic Acid (DNA) synthesis and sequencing technologies, DNA has become a promising storage medium for mass data storage due to its longevity and enormous information density. In recent years, the DNA-based storage system has received extensive attention [3], [4], [5], [6], [7], [8].

In a typical DNA-based storage system based on next-generation sequencing technologies (i.e., the Illumina sequencing technology), as depicted in Fig. 1, digital data is fragmented into pieces and encoded into short DNA strands that usually do not exceed 300 nucleotides due to the limitation of the synthesizing and sequencing technologies. These DNA strands are duplicated and spatially disordered in the experimental system. During the sequencing process, some DNA strands may get lost, which are called dropouts. This process is characterized as a shuffling-sampling channel [9]. Meanwhile, insertions, deletions, and substitutions may be introduced into individual DNA strands due to biological mutations. These errors are characterized as an insertion, deletion, and substitution (**IDS**) channel [10], [11]. For convenience, we refer to a single insertion or deletion as an indel error, and a single insertion, deletion, or substitution as an edit error. Following the above model, one of the related information-theoretic problems is the reliable transmission over the DNA-based storage channel. Error-correcting codes are required to ensure accurate recovery of the information stored on DNA strands.

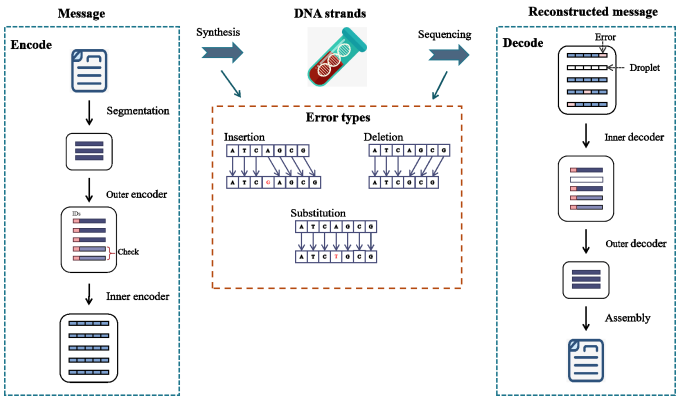


Fig. 1. A typical DNA-based data storage system.

To cope with dropouts and edit errors, a typical approach is to introduce a concatenated coding scheme. The outer code is used to correct erasures and substitutions after adding indexes to DNA strands [9]. Whereas the inner code is used to correct edit errors in individual DNA strands. Classic error-correcting codes for memoryless channels, such as the Reed-Solomon (RS) code [12], the Low-Density Parity-Check (LDPC) code [13], and Luby Transform (LT) code [7], [14] have shown reliable performance as the outer code. In contrast, the study on edit errors has received

• Z. Yan, C. Liang and H. Wu are with the Center for Applied Mathematics, Tianjin University, Tianjin, China, 300072.
E-mail: {yanzh, cong.liang, whming}@tju.edu.cn

(Corresponding author: Huaming Wu)

TABLE 1
Comparison of edit correction coding schemes.

Parameter	RS code [7], [18][1]	Press *et al.* [12][1]	Davey and Mackey [11]	Sima *et al.* [15]	Abroshan and Venkataramana [19][2]	Cai *et al.* [20][2]	Cai *et al.* [21][3]	This work
Error correction	substitutions	stochastic edit errors	stochastic edit errors	t-deletions	segmented indel errors	segmented indel errors	single edit error	segmented edit errors
Encoding structure	systematic	systematic	systematic	systematic	non-systematic	non-systematic	systematic	systematic
Redundancy (bits)	/	/	/	$4t\log n + O(\log n)$	$\log(n+1)+16$	$2\log n + 24$	$\log n + O(\log\log n) + 16$	$2\log k + 12$
Decoding complexity	$O(n\log n)$	$O(2^n)$	$O(n\log n)$	$O(n^{2t+1})$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

[1] For consistency, the table describes schemes that were designed for inner codes. Works on [7], [18] used conventional RS codes to select error-free readouts, whereas work on [12] designed a new code, termed HEDGES, which is capable to correct stochastic edit errors.
[2] Theoretical results of the works on [19], [20] are provided here, in which no practical systematic encoding algorithm is given.
[3] Presented here is the work in [21, Theorem 7], a systematic encoding algorithm found in [22].

limited attention in the literature. Although some attempts have been made to this problem, there is still a lack of a practical and efficient algorithm that performs on large-scale data, as will be discussed next. Thus, the most important objective of our work is, specifically, to provide an improved edit error correction code for reliable communication over the IDS channel.

We assume the requirements for the inner code to include two essential features: (i) the ability to correct stochastic edit errors; (ii) the encoding and decoding complexity should ideally be linear because of the need for large-scale dataset usage.

However, previous well-studied codes for correcting stochastic edit errors, such as the HEDGES code [12], the watermark code [11], and the q-ary t-deletion-correcting codes presented by Sima *et al.* [15], all require more than linear time to decode (see in Table 1). Additionally, the code introduced in [16] that can correct insertions, deletions, and transpositions also takes polynomial time in the codeword length to encode and decode. The work [10] provided an edit error-correcting algorithm for convolutional codes based on the BCJR algorithm. However, for a (n, k, m) convolutional code, its decoding operations perform in polynomial time in $(k+m)$. A recent survey on such codes is shown in [17]. Unfortunately, all of these codes are not optimal due to their decoding complexity.

Another feature of DNA-based storage that needs to be accounted for is its error rate over the inner channel. Analysis of the experimental data based on the Illumina sequencing technology indicates that the conventional DNA-based storage system suffers a low raw error rate, almost 1% [7], [8], [23]. For example, the experiment in [8] obtained that the average nucleotide error rate per position is 0.6%, with 0.4% substitutions, 0.2% deletions, and 0.04% insertions. This inspired us to design the edit error-correcting code from a new perspective, i.e., to build a structure that can be kept synchronized against stochastic errors based on the segmented error correction code.

In this work, to achieve efficient encoding and decoding for large-scale DNA storage, we design a new edit error-correcting code for the IDS channel with low complexity, low redundancy, and the ability to integrate information from multiple reads, termed *DNA-LM* code (a hybrid Levenshtein code and marker code for the DNA-based storage system). The codeword of the *DNA-LM* code implicitly contains disjoint segments, each of which consists of two parts called the marker codeword and the data-block codeword. The marker codeword is 2 quaternary symbols encoded from its adjacent segments. The data-block codeword has a generalized Levenshtein code structure encoded from the message segment. Our contributions are as follows.

- We provide a new segmented-edit error-correcting code with linear time complexity for both encoding and decoding, much lower than that of previous DNA-based storage systems [7], [10], [12], [18] (see in Table 1).
- We design a new code structure such that each codeword segment can correct a single edit error without knowing its endpoint and regain the synchronization of the next segment, which is particularly different from conventional segmented-error-correcting codes [19], [20], [24]. When multiple edit errors occur in a segment, our code marks this segment as erasures and easily resyncs to continue decoding subsequent segments. The redundancy per segment is $\lceil \log k \rceil + 6$ quaternary symbols (k is the length of the message segment), which is also lower than previous segmented-error-correcting codes.
- We generalize our decoding algorithm so that it can decode duplicated DNA strands. The superiority of our work is that trace reconstruction can be performed simply from our code structure, and can work in conjunction with error correction, while the overall decoding still has linear time complexity.
- To test the edit error-correcting capability of our code, we implement its encoding and decoding algorithms and simulate them under a stochastic error channel. Simulations show that the *DNA-LM* code is also stable against stochastic edit errors and can achieve the required low decoding error rate with an appropriate number of segments. In particular, by concatenating with an efficient outer code (e.g., the (255,223) RS code), this cascade coding scheme is capable to achieve error-free decoding.

This paper is organized as follows. In Section 2, we introduce the DNA alphabet, the inner channel model for DNA storage, and provide three coding schemes of segmented-edit error-correcting codes over the quaternary alphabet. In Section 3, we present our new *DNA-LM* code and prove its edit error-correcting capability. In Section 4, we provide a linear decoding algorithm of our code. In Section 5, we describe how to generalize our code in the case of duplications. In Section 6, we discuss the capacities of the *DNA-LM* code, including the code rate, the algorithm complexity, and the decoding error rate. We also test the performance of the cascade coding scheme by concatenating our *DNA-LM* code and the (255,223) RS code. Finally, we conclude the paper in Section 7.

2 Preliminaries and Problem Definition

2.1 DNA Alphabet

DNA strands consist of chains made from four types of nucleotide subunits with different bases: adenine (A), cytosine (C), guanine (G), and thymine (T). We map these four DNA nucleotides $\mathcal{D} = \{A, T, G, C\}$ to the quaternary alphabet $\Sigma = \{0, 1, 2, 3\}$, as follows:

$$A \leftrightarrow 0, \quad T \leftrightarrow 1, \quad C \leftrightarrow 2, \quad G \leftrightarrow 3.$$

Given any DNA sequence $\boldsymbol{\sigma} \in \mathcal{D}^N$, it could be one-to-one mapped to a quaternary sequence $\boldsymbol{x} \in \Sigma^N$. As a result, we focus on the codes over the quaternary alphabet in this work.

2.2 Problem definition: Designing Edit error-correcting codes for IDS channels

The main problem we consider is to correct edit errors over the IDS channel introduced in [10], [11]. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ denote the information sequence, $\mathbf{Enc} : \Sigma^k \to \Sigma^n$ denote an encoder map. Then the codeword is $\mathbf{Enc}(\boldsymbol{x}) = \boldsymbol{c}$, which will be transmitted over the IDS channel. Let $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$ denote the channel output sequence, and $\mathbf{Dec} : \Sigma^N \to \Sigma^k$ denote a decoder map. Here, the length of the channel output sequence N is random and depends on the insertion and deletion probabilities. Our objective is to design an encoder map $\mathbf{Enc}$ and a decoder map $\mathbf{Dec}$ such that (i). $\mathbf{Enc}$ has the lowest possible redundancy, (ii). both $\mathbf{Enc}$ and $\mathbf{Dec}$ have linear complexity, and (iii). the Hamming distance between $\mathbf{Dec}(\boldsymbol{y})$ and $\boldsymbol{x}$ is as small as possible. Here, the most commonly used performance measure is the nucleotide-error rate, denoted NER.

For illustration, the transition process characterizing a single use of the channel is shown in Fig. 2. At the channel use of x_i, three events may occur: (i). With probability p_i, an insertion event occurs where a uniformly random symbol is inserted into the received stream; (ii). With probability p_d, the enqueued symbol x_i is deleted; (iii). With probability $p_t = 1 - p_i - p_s$, the enqueued symbol x_i is transmitted, i.e., put into the received stream, with a probability of suffering a substitution error. We assume that error probabilities are independent and identically distributed (i.i.d.).

2.3 Related Work

The segmented-edit error channel was introduced by Liu and Mitzenmacher [24], and subsequently studied by Abroshan *et al.* [19]. Recently, it has been applied to the DNA-based storage system [20]. In these studies, the segmented-edit error channel model refers to a channel where input sequences are implicitly divided into disjoint segments, with at most one insertion or deletion occurring in each segment. Each segmented codeword contains fixed-pattern prefixes and suffixes to determine the boundaries between segments. Since a substitution should be considered as a deletion plus an insertion, these codes can not cope with substitutions. To avoid confusion in this paper, we call the above codes the segmented-indel error-correcting codes, and call a code that could correct one indel error or one substitution in each segment a segmented-edit error-correcting code. Besides, these codes are greedily constructed and thereby have no systematic encoding algorithm. The theoretical performance of these codes is shown in Table 1.

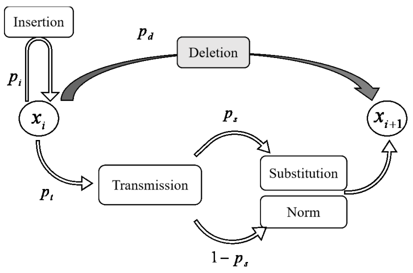


Fig. 2. The transmission probabilities of a single symbol in the IDS channel.

Given that previous segmented-indel error-correcting codes are not robust to substitutions, and they have not been provided with practical coding algorithms, we have not been able to compare the numerical results of the decoding performance between our codes and them. We follow the same assumption of segmented errors and design modified coding schemes which can correct edit errors. We concatenate the marker code [25] (the marker codeword) with quaternary single-indel/edit error-correcting codes (the data-block codeword) to form segments. The marker can play a similar role to the prefix/suffix of segmented-indel error-correcting codes. We set the sequence “001” as the marker, which is inspired by [19]. The difference is that in order to achieve systematic encoding, we separate the marker from the codeword. We now provide three representative single-edit error-correcting codes as codeword components.

To correct a single edit error, we have a celebrated class of the Varshamov-Tenengolts (VT) code [26]. The VT code originally refers to a class of binary algebraic block codes that consists of all binary vectors of length n belonging to

$$VT_{a,m}(n) = \left\{ \boldsymbol{x} \in \{0,1\}^n : \sum_{i=1}^{n} i x_i \equiv a \pmod{m} \right\}, \qquad (1)$$

where m is a predetermined integer. a is an integer with $0 \le a \le m-1$, usually called the syndrome or the remainder of the sequence $\boldsymbol{x}$. Levenshtein [27] later proved that for $m \ge n+1$ and a fixed integer a with $0 \le a \le m-1$, the VT codes are asymptotically optimal single-indel error-correcting codes. Levenshtein also showed that when $m \ge 2n$, the VT code can correct an edit error. The structure of (1) has been generalized to many forms, which are often referred to as VT codes collectively. The following are classic quaternary VT codes.

2.3.1 The Tenengolts Code for Non-Binary Alphabets

Tenengolts first generalized the VT codes to non-binary alphabets in 1984 [28]. For any q-ary sequence $\boldsymbol{s} = (s_1, s_2, \cdots, s_n)$, Tenengolts defined a corresponding $(n-1)$-length auxiliary binary sequence $A_{\boldsymbol{s}} = (\alpha_1, \alpha_2, \cdots, \alpha_{n-1})$, where $\alpha_i = 0$ if $s_i < s_{i-1}$ and otherwise $\alpha_i = 1$. For any $0 \le a \le n-1$ and $0 \le b < q$, the Tenengolts code is defined as follows:

$$Ten_{a,b}(n) \triangleq \Big\{ \boldsymbol{s} \in \mathbb{Z}_q^n : \sum_{i=1}^{n} s_i \equiv b \,(\bmod\ q), \sum_{i=1}^{n-1} i\alpha_i \equiv a \,(\bmod\ n) \Big\},$$

where $Ten_{a,b}(n)$ is a single-indel error-correcting code.

Unfortunately, Tenengolts did not provide an efficient algorithm for encoding a message into such a code with this algebraic structure. Only recently, Abroshan *et al.* [29] proposed an encoder to systematically map a k-bits message sequence onto a n-length q-ary codeword. Therefore, we term it **Encoder *TA*** (introduced by Tenengolts and encoded by Abroshan *et al.*).

2.3.2 The Interleaved Binary Encoder

A systematic encoding method for the binary VT code that can correct a single indel error was first introduced in 1998 by Abdel-Gaffar and Ferreira [30]. Later, Saowapa *et al.* [31] adopted it to get a systematic encoder for codes that can correct a single edit error. We call it **Encoder *SS*** (satisfied systematic code structure and encoded by Saowapa), and briefly restate the linear encoding method here since we will utilize this method for an interleaved encoding scheme below.

Encoder *SS*: For any message sequence $\boldsymbol{x} = \{x_1, x_2, \cdots, x_k\} \in \{0,1\}^k$, encoder SS sticks it into a codeword $\boldsymbol{y} = SS(\boldsymbol{x}) \in VT_{a,2n}(n)$, where $k = n - \lceil \log n \rceil - 1$. The encoder inserts "parity" bits at dyadic positions, i.e., c_{2^i}, for $0 \le i \le t-2$ and c_n, and attaches message symbols to other positions, to ensure that $\sum_{i=1}^{n} iy_i \equiv a \pmod{2n}$. Here, $t = n - k$ is the number of redundancy bits.

Example: Given a message sequence $\boldsymbol{x} = 01011$ and a fixed syndrome $a = 0$, we have $n = 10$, $t = 5$ and $m = 20$. The codeword $\boldsymbol{y} = (y_1, y_2, \cdots, y_{10})$ should satisfy that $\sum_{i=1}^{10} iy_i \equiv \sum_{j=1}^{4} 2^{j-1} y_{2^{j-1}} + 10 \cdot y_{10} + 0 \cdot 3 + 1 \cdot 5 + 0 \cdot 6 + 1 \cdot 7 + 1 \cdot 9 \equiv 0 \pmod{20}$. Then expand $19 - 10 = 9$ into the binary form $1 \cdot 2^3 + 1 \cdot 2^0$. We obtain the codeword $\bar{1}\bar{0}0\bar{0}101\bar{1}1\bar{1}$, where bits with overbars are check bits.

Since we are interested in codes over the quaternary alphabet, it is also quite intuitive to construct a code by interleaving two binary Levenshtein codewords. Here we call it **Encoder *IBS*** (obtained by interleaving two binary VT codes and encoded by Saowapa) and describe it in detail.

Interleaved Sequence: For a sequence $\boldsymbol{y} = (y_1, y_2, \cdots, y_k) \in \Sigma^k$, let $y_i = y_i^{(0)} 2^0 + y_i^{(1)} 2^1$ be the binary form of symbol y_i, then set $\boldsymbol{y^{(0)}} = (y_1^{(0)}, \cdots, y_n^{(0)})$, $\boldsymbol{y^{(1)}} = (y_1^{(1)}, \cdots, y_n^{(1)})$. Here, we say that $\boldsymbol{y}$ is the interleaved sequence of $\boldsymbol{y^{(0)}}$ and $\boldsymbol{y^{(1)}}$, and denote it by $\boldsymbol{y^{(1)}} \parallel \boldsymbol{y^{(0)}}$.

Encoder *IBS*: The IBS code is defined as:

$$IBS_{k,a}(n) = \Big\{ \boldsymbol{y} = \boldsymbol{y^{(1)}} \parallel \boldsymbol{y^{(0)}} : \boldsymbol{x} = \boldsymbol{x^{(1)}} \parallel \boldsymbol{x^{(0)}} \in \Sigma^k,$$
$$\boldsymbol{y^{(i)}} = \textbf{Encoder } SS(\boldsymbol{x^{(i)}}) \in VT_{a,2n}(n), i = 0, 1 \Big\},$$

where $k = n - \lceil \log_2(n) \rceil - 1$. Obviously, the $IBS_{k,a}(n)$ code is a single-edit error-correcting code over the quaternary alphabet.

Example: Given a message $\boldsymbol{x} = 02312$ and a fixed syndrome $a = 0$, we have $n = 10$ and $m = 20$. we expand the message into 01101 ∥ 00110, then encode $SS(01101) =$ 0000110010 and $SS(00110) =$ 1101011000. Hence, we have the codeword $\boldsymbol{y} =$ 0000110010 ∥ 1101011000 = 1101231020.

2.3.3 The Order-Optimal Code

Cai *et al.* [21] provided a single-edit error-correcting code over the quaternary alphabet by enforcing the k-sum balanced constraint on the message sequence and appending the syndrome next to it. Here we call it **Encoder *Cai***. In general, the redundancy of n-length VT codewords is $K \log n + o(\log n)$ bits, where K is a constant that varies by code structure. Small K is preferred to ensure a high code rate. The redundancy of Encoder *Cai* is $\log n + O(\log\log n) + 16$ bits. It has $K = 1$, as a result, it was called order-optimal. This code has an efficient code rate when n is large (say $n > 512$). However, when the message sequence is short the advantage of order-optimality is impacted because some constant redundant bits are always required, which decreases the practical code rate when n is small (see in Table 1).

More recently, this work was extended in [32], which provides another simpler proof of the result on optimal redundancy and a single-edit error-correcting code with lower redundancy (modified from Encoder *Cai*). Since it does not provide a practical coding algorithm, we will focus more on Encoder *Cai* in the next subsection.

2.4 Quaternary Segmented-indel/edit Error-correcting Code Construction

Armed with the above typical VT codes, we now construct new segmented-indel/edit error-correcting codes by concatenating the marker with each VT codeword. For illustration, the code structure is shown in Fig. 3. Each codeword segment includes a data-block codeword and a marker codeword. We use a fixed pattern "001" as the marker code, which is a conventional choice [19], [25]. We use the codes described above in turn to be the data-block codeword, and call the concatenated codes the ***4-ary TA-Marker*** code, the ***IBS-Marker*** code, and the ***Cai-Marker*** code, respectively.

Since markers can be used to locate the segment boundaries and regain synchronization, these codes can correct segmented-indel errors (the *TA-Marker* code) or segmented-edit errors (the *IBS-Marker* code and the *Cai-Marker* code). However, since re-synchronization can only rely on markers, they may not be robust enough to stochastic edit errors.

3 The DNA Levenshtein-Marker code

In this section, we present a new segmented-edit error-correcting code, which is more robust than codes described in Section 2.3 in that it can correct a single edit error in each codeword segment including the marker, and has a double guarantee for re-synchronization. This code is designed over the quaternary alphabet for DNA-based storage applications, and we term it the DNA Levenshtein-Marker (***DNA-LM***) code.

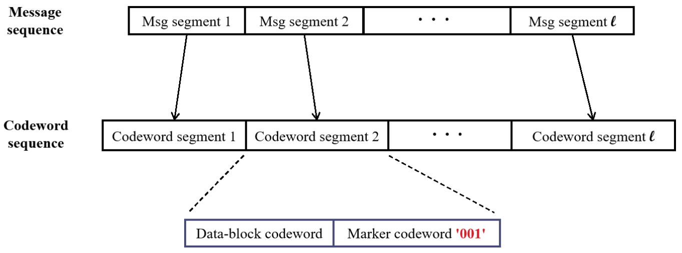


Fig. 3. The concatenated encode structure of data-block codewords and Sellers marker codewords.

3.1 The Structure of *DNA-LM* Code

As depicted in Fig. 4, the *DNA-LM* code is concatenated by a series of segmented codewords. Each codeword segment is systematically encoded by the DNA segment-Levenshtein-Marker (***DNA-sLM***) code, which is concatenated by a marker code and a data-block code. We call the redesigned marker code **the marker**. Each data-block codeword consists of four components, namely, **the message**, **the check**, **the separator**, and **the syndrome**. The separator and the marker work collaboratively to detect whether an indel error exists and locate which component contains it. The check detects and locates the substitution. If the edit error is in the message, the syndrome will correct it. Otherwise, the error-free message will correct other components. By majority vote, the marker, the separator, and the syndrome can further regain the synchronization of the next segment.

In practice, the encoder first divides the message sequence into several short segments, then encodes each message segment into a *DNA-sLM* codeword, and finally concatenates them together to construct a complete codeword (Fig. 4). Here, the first segment does not need the marker to keep synchronization, so the encoder omits it.

Before giving the structure of the *DNA-sLM* code, we define some important symbols as follows:

- Define $Syn : \{0,1\}^k \rightarrow [0, 2k)$ as the syndrome of a binary sequence, where
$$Syn(z) \equiv \sum_{i=1}^{k} i z_i \pmod{2k}. \tag{2}$$
We use $BSyn(z)$ to represent the binary form of $Syn(z)$.
For any message sequence $\boldsymbol{x} = \boldsymbol{x}^{(1)} \parallel \boldsymbol{x}^{(0)}$, we define the function $IBSyn : \Sigma^k \rightarrow \Sigma^t$ to satisfy
$$IBSyn(\boldsymbol{x}) = BSyn(\boldsymbol{x}^{(1)}) \parallel BSyn(\boldsymbol{x}^{(0)}), \tag{3}$$
where $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(0)} \in \{0,1\}^k$ and $t = \lceil \log(2k) \rceil$. We call $IBSyn(\boldsymbol{x})$ the syndrome of $\boldsymbol{x}$.
- Define $Ck : \Sigma^k \rightarrow \Sigma$ as the check function of the message $\boldsymbol{x}$, where
$$Ck(\boldsymbol{x}) \equiv \sum_{i=1}^{k} x_i \pmod 4. \tag{4}$$
- Define $Sp : \Sigma^3 \rightarrow \Sigma$ as the separator function. Specifically, $Sp(a,b,c)$ satisfies, if $a+2 \neq b, c$, set $Sp(a,b,c) = a+2$, if $a+3 \neq b,c$, set $Sp(a,b,c) = a+3$, otherwise, $Sp(a,b,c) = a+1$.
- Define $Mk : \Sigma^3 \rightarrow \Sigma$ as the marker function, which is the same as the definition of the separator function Sp.

Then, the structure of the *DNA-sLM* code is defined as follows.

Definition 1. *The* DNA-sLM *code C, is defined as*
$$C = \Big\{(m, m, x_1, x_2, \cdots, x_k, c, s, s, s, a_1, \cdots, a_t) \in \Sigma^{k+t+6} :$$
$$c = Ck(x_1, x_2, \cdots, x_k), (a_1, a_2, \cdots, a_t) = IBSyn(x_1, x_2, \cdots, x_k),$$
$$s = Sp(c, a_1, a_2), m = Mk(f, x_1, x_2)\Big\},$$
where f is the last symbol of the previous codeword segment. We call (m, m) the marker, $\boldsymbol{x} = (x_1, x_2, \cdots, x_k)$ the message, (c) the check, (s, s, s) the separator, and $\boldsymbol{a} = (a_1, a_2, \cdots, a_t)$ the syndrome.

Example: Let the message sequence be $\boldsymbol{x} = 013210$ with two segments $\boldsymbol{x}_1 = 013$ and $\boldsymbol{x}_2 = 210$. To obtain the first codeword segment, we have $\boldsymbol{x}_1^{(0)} = 011$ and $\boldsymbol{x}_1^{(1)} = 001$, then $Syn(\boldsymbol{x}_1^{(0)}) = 5$ and $Syn(\boldsymbol{x}_1^{(1)}) = 3$, thus $IBSyn(\boldsymbol{x}_1) = 011 \parallel 101 = 123$, $Ck(\boldsymbol{x}_1) = 0$, and $Sp(0,1,2) = 3$. Here, we omit its marker, so the codeword is 0130333123. For the second codeword segment, we have $Mk(3,2,1) = 0$, and other parts are obtained similarly to the first segment, thus it is 002103222012. The entire codeword is
$$\boldsymbol{y} = 0130333123002103222012. \tag{5}$$

3.2 Proof of Error-Correcting Capability of *DNA-LM* Code

In this subsection, we show that the *DNA-LM* code can correct segmented edit errors and regain synchronization after each segment decoding. For conventional segmented-indel error-correcting codes [19], [20], since the prefix and the suffix within a segment do not have errors simultaneously, they can mutually ensure that the segment boundary can be recognized. In analogy with them, markers in our code structure play the same role as segment prefixes of their codes. However, our code structure has no suffixes to identify segment endpoints. Instead, we elaborately design the *DNA-LM* code structure so that it can both correct a single edit error and regain synchronization. The following discussion is conducted under the condition that the received segmented codeword has an undetermined endpoint.

Theorem 1. *The* DNA-sLM *code can correct a single edit error and identify the starting position of the next segment. As a result, the* DNA-LM *code can correct segmented edit errors through iterative decoding of the* DNA-sLM *code.*

Before proving this theorem, we need to establish some lemmas. Levenshtein has proved that codes have structure (1) with $m = 2n$ can correct an edit error [27]. Based on this, Lemma 1 shows that the syndrome can detect the edit error even when the boundary of a received codeword is unknown. Inspired by Levenshtein's proof (in which determines the type of error based on the length of the sequence) [27], our proof further shows that our decoder may detect errors even if it is uninformed of the boundary by using a distinct code structure. Lemma 2 proves that our data block code is able to correct a single edit error, the idea of which is derived from [28]. Lemma 3 further proves

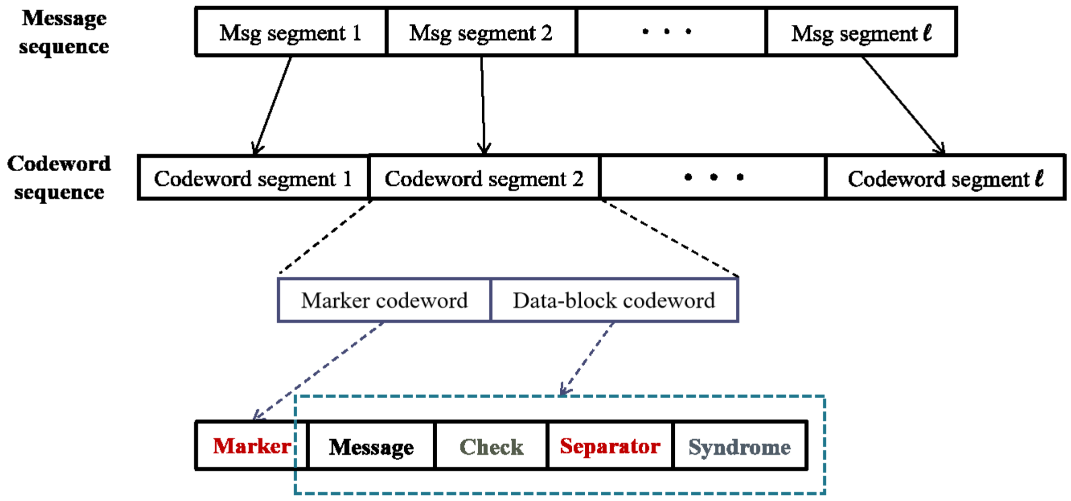


Fig. 4. The structure of the proposed DNA Levenshtein-Marker codeword.

that if an edit error occurs in the marker, our code can still detect it and obtain the correct message. Finally, Lemma 4 shows that our code can determine where the segment ends after decoding. It indicates that the decoding of the *DNA-sLM* code can be performed segment-by-segment so that the *DNA-LM* code is capable of correcting segmented edit errors and keeping synchronization.

For a k-length message, we define

$$\mathcal{D}_{ins}(\boldsymbol{x}) \triangleq \left\{\overline{\boldsymbol{x}}_{ins} \in \Sigma^n : (x_1,\cdots,x_{j-1},r,x_j,\cdots,x_{k-1}) \neq \boldsymbol{x}\right\},$$
$$\mathcal{D}_{del}(x) \triangleq \left\{\overline{\boldsymbol{x}}_{del} \in \Sigma^n : (x_1,\cdots,x_{j-1},x_{j+1},\cdots,x_k,r) \neq \boldsymbol{x}\right\},$$
$$\mathcal{D}_{sub}(x) \triangleq \left\{\overline{\boldsymbol{x}}_{sub} \in \Sigma^n := (x_1,\cdots,x_{j-1},r,x_{j+1},\cdots,x_k) \neq \boldsymbol{x}\right\},$$

to represent the first k elements of a received *DNA-sLM* codeword which has an insertion, a deletion, and a substitution error in the data-block, respectively. Here, if a deletion occurs, the subsequent symbol will be added. And if an insertion occurs, x_k will be truncated.

Lemma 1. *For any sequence* $\overline{\boldsymbol{x}} \in \mathcal{D}_{ins}(x) \cup \mathcal{D}_{del}(x) \cup \mathcal{D}_{sub}(x)$, $IBSyn(\overline{\boldsymbol{x}}) \neq IBSyn(\boldsymbol{x})$.

Proof. We first consider the insertion error. Let $\overline{a}^{(0)}$ and $\overline{a}^{(1)}$ be syndromes of $\overline{\boldsymbol{x}}^{(0)}_{ins}$ and $\overline{\boldsymbol{x}}^{(1)}_{ins}$, where $\overline{\boldsymbol{x}}^{(1)}_{ins} \parallel \overline{\boldsymbol{x}}^{(0)}_{ins} = \overline{\boldsymbol{x}}_{ins}$. And let $a^{(0)}, a^{(1)}$be the correct syndrome of $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$, where $\boldsymbol{x}^{(1)} \parallel \boldsymbol{x}^{(0)} = \boldsymbol{x}$. We assume that $IBSyn(\overline{\boldsymbol{x}}_{ins}) = IBSyn(\boldsymbol{x})$, so that $\overline{a}^{(0)} = a^{(0)}$and $\overline{a}^{(1)} = a^{(1)}$. Without loss of generality, we omit the superscript, it follows that

$$a - \overline{a} \equiv kx_k - \sum_{i=j}^{k-1} x_i - jr \pmod{2k}.$$

If $r = 0$, since $0 \leq \sum_{i=j}^{k-1} x_i \leq k-1$, we have $a - \overline{a} \equiv 0 \pmod{2k}$ if and only if $x_j, \cdots, x_{k-1}, x_k$ are all equal to 0. And if $r = 0$, since $j \leq j + \sum_{i=j}^{k-1} x_i \leq k$, we have $a - \overline{a} \equiv 0 \pmod{2k}$ if and only if $x_j, \cdots, x_{k-1}, x_k$ are all equal to 1. This is contrary to our assumption $\boldsymbol{x} \notin \mathcal{D}_{ins}(x)$, so the assumption does not hold. The same could be proved for the deletion and substitution errors. □

Since the encoding of the data-block code is independent of the marker code, we first show the edit error-correcting capability of the data-block code.

Lemma 2. *The data-block code can correct a single edit error.*

Proof. For any message $\boldsymbol{x}$, denote its data-block codeword as $\boldsymbol{y} = (\boldsymbol{x}, c, s, s, s, \boldsymbol{a})$. Let the error domain obtained from $\boldsymbol{y}$ via at most one edit error be

$$\mathcal{D}(\boldsymbol{y}) \triangleq \left\{\boldsymbol{r} : \boldsymbol{r} \text{ is obtained from } \boldsymbol{y} \text{ via an edit error}\right\} \cup \left\{\boldsymbol{y}\right\}.$$

The data-block code is a single-edit error-correcting code if and only if for different data-block codewords $\boldsymbol{y_1} \neq \boldsymbol{y_2}$,

$$\mathcal{D}(\boldsymbol{y_1}) \cap \mathcal{D}(\boldsymbol{y_2}) = \emptyset.$$

We first discuss about the error domain $\mathcal{D}(\boldsymbol{y})$. Let $\boldsymbol{r}_1^k$ denote the first k elements of $\boldsymbol{r}$. Take $\overline{\boldsymbol{c}} = Ck(\boldsymbol{r}_1^k)$, and $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_1^k)$. We first define some events to represent diverse error cases.

- Suppose $\mathcal{D}(Ck)$ represents the event in which the received sequence satisfies the following form:

  $$\mathcal{D}(Ck) \triangleq \left\{\boldsymbol{r} : r_{k+1} = Ck(\boldsymbol{r}_1^k)\right\}.$$

  We can see that $\mathcal{D}(Ck)$ is the subset of $\mathcal{D}(\boldsymbol{y})$, and if the message and the check are both correct in the received sequence, the received sequence $\boldsymbol{r}$ should belong to $\mathcal{D}(Ck)$. However, the opposite is not correct.
- Suppose $\mathcal{D}(Sp_{-1}), \mathcal{D}(Sp_0)$, and $\mathcal{D}(Sp_{+1})$ respectively represent events in which the received sequence satisfies the following form:

  $$\mathcal{D}(Sp_0) \triangleq \left\{\boldsymbol{r} : r_{k+2} = r_{k+3} = r_{k+4}\right\},$$
  $$\mathcal{D}(Sp_{+1}) \triangleq \left\{\boldsymbol{r} : r_{k+3} = r_{k+4} = r_{k+5} \neq r_{k+2}\right\},$$
  $$\mathcal{D}(Sp_{-1}) \triangleq \left\{\boldsymbol{r} : r_{k+1} = r_{k+2} = r_{k+3} \neq r_{k+4}\right\}.$$

  It is obvious that $\mathcal{D}(Sp_{-1}), \mathcal{D}(Sp_0)$, and $\mathcal{D}(Sp_{+1})$ are subsets of $\mathcal{D}(\boldsymbol{y})$ as well. Assuming that the separator is correct, then if there is no indel error in the message and the check, $\boldsymbol{r} \in \mathcal{D}(Sp_0)$; if there is a deletion in the message or the check, $\boldsymbol{r} \in \mathcal{D}(Sp_{-1})$; and if there is an insertion in the message or the check, $\boldsymbol{r} \in \mathcal{D}(Sp_{+1})$. Besides, due to the structure of the separator, it is apparent that $\mathcal{D}(Sp_{-1}), \mathcal{D}(Sp_0)$ and $\mathcal{D}(Sp_{+1})$ are mutually disjoint. If the separator is incorrect, the received sequence will not satisfy any of the above forms. We use $\mathcal{D}(Sp) = \mathcal{D}(Sp_{-1}) \cup \mathcal{D}(Sp_0) \cup \mathcal{D}(Sp_{+1})$ to represent the event that the separator is correct.

- Suppose $\mathcal{D}(Syn_{-1}), \mathcal{D}(Syn_0)$ and $\mathcal{D}(Syn_{+1})$ represent events, respectively, in which the received sequence satisfies the following form:

$$\begin{aligned}\mathcal{D}(Syn_i) \triangleq \Big\{ \boldsymbol{r} : (r_{k+5+i}, r_{k+6+i}, \cdots, r_{k+4+t+i}) \\ = IBSyn(\boldsymbol{r}_1^k) \Big\} \cap \Big(\mathcal{D}(Sp_i) \cup \mathcal{D}(Sp)^c \Big),\end{aligned}$$

where $i \in \{-1, 0, +1\}$. We use $i = +1$, -1, and 0 to indicate the symbol-shift of the syndrome. If the message and the syndrome are correct in the received sequence, $\boldsymbol{r}$ should satisfy one of the above forms. We use $\mathcal{D}(Syn) = \mathcal{D}(Syn_{-1}) \cup \mathcal{D}(Syn_0) \cup \mathcal{D}(Syn_{+1})$ to represent the event that the syndrome and the message are both correct.

Armed with the above definitions, we define the following events,

$$\begin{aligned}
\mathcal{D}(S_1) &\triangleq \mathcal{D}(Ck) \cap \mathcal{D}(Sp_0) \cap \mathcal{D}(Syn_0); \\
\mathcal{D}(S_2) &\triangleq \Big(\mathcal{D}(Ck) \cap \mathcal{D}(Syn)\Big) \cup \Big(\mathcal{D}(Ck) \cap \mathcal{D}(Sp_0)\Big) \\
&\quad \bigcup_{i \in \{-1,0,+1\}} \Big(\mathcal{D}(Sp_i) \cap \mathcal{D}(Syn_i)\Big); \\
\mathcal{D}(S_3) &\triangleq \Big(\mathcal{D}(Ck)^c \cap \mathcal{D}(Sp_0) \cap \mathcal{D}(Syn_0)^c\Big) \\
&\quad \bigcup_{i \in \{-1,+1\}} \Big(\mathcal{D}(Sp_i) \cap \mathcal{D}(Syn_i)^c\Big). \qquad (6)
\end{aligned}$$

Then, when a single edit error occurs, we can make observations:

- If the whole received segment has no edit error, $\boldsymbol{r}$ should belong to set $\mathcal{D}(S_1)$;
- If the message is correct but the other part has an edit error, we can see that $\boldsymbol{r} \in \mathcal{D}(S_2)$. The proof is obvious. Based on our assumptions, when the message is correct, at most only one function of the check, the separator, or the syndrome does not match the message. For example, $\boldsymbol{r} \in \mathcal{D}(Sp_{-1})$ represents an event where the separator is correct but moved forward by one position, which indicates that a deletion occurred before the separator. In this case, the syndrome must also move one position forward. Otherwise, the received sequence $\boldsymbol{r}$ has an error in both the check and the syndrome, which contradicts our hypothesis. Secondly, since the message is correct, $IBSyn(\boldsymbol{r}_1^k)$ is equal to the received syndrome $(r_{k+4}, r_{k+5}, \cdots, r_{k+3+t})$, which means that $\boldsymbol{r}$ satisfies $\mathcal{D}(Syn_{-1})$. Therefore, when the message is correct and the check is deleted, $\boldsymbol{r}$ satisfies $\mathcal{D}(Sp_{-1}) \cap \mathcal{D}(Syn_{-1})$.
- If the message has an edit error, $\boldsymbol{r} \in \mathcal{D}(S_3)$. The proof is as follows. If a substitution occurs in the message, $IBSyn(\boldsymbol{r}_1^k)$ will not be equal to $(r_{k+5}, r_{k+6}, \cdots, r_{k+4+t})$, so that $\boldsymbol{r} \notin \mathcal{D}(Syn_0)$. And because the separator and the check match the original message, according to Lemma 1, $\boldsymbol{r} \in \mathcal{D}(Ck)^c \cap \mathcal{D}(Sp_0) \cap \mathcal{D}(Syn_0)^c$. The same relations can be obtained when an insertion occurs in the message, $\boldsymbol{r} \in \mathcal{D}(Sp_{+1}) \cap \mathcal{D}(Syn_{+1})^c$; when a deletion occurs in the message, $\boldsymbol{r} \in \mathcal{D}(Sp_{-1}) \cap \mathcal{D}(Syn_{-1})^c$.

In summary, $\mathcal{D}(S_1)$ indicates that the whole data-block codeword has no edit error; $\mathcal{D}(S_2)$ indicates that the message is correct and we decode the first k-symbols of the received sequence as the message segment; $\mathcal{D}(S_3)$ indicates that the message has an edit error. Furthermore, it is easy to Figure out that $\mathcal{D}(S_1) \subset \mathcal{D}(S_2)$, and $\mathcal{D}(S_2) \cap \mathcal{D}(S_3) = \emptyset$, $\mathcal{D}(S_2) \cup \mathcal{D}(S_3) = \mathcal{D}(\boldsymbol{y})$. Here, due to space constraints, we omit the proof.

Suppose that there are two different codewords $\boldsymbol{y_1}, \boldsymbol{y_2}$, and a received sequence $\boldsymbol{r}$, which simultaneously belongs to $\mathcal{D}(\boldsymbol{y_1})$ and $\mathcal{D}(\boldsymbol{y_2})$. Assume that the message segments of codewords $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$ are $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, respectively.

On the one hand, if the received sequence $\boldsymbol{r}$ belongs to $\mathcal{D}(S_3)$, the message is incorrect and both the separator and the syndrome are correct. In this case, the position of the syndrome is uniquely determined by checking which event of $\mathcal{D}(Sp_0)$, $\mathcal{D}(Sp_{-1})$, and $\mathcal{D}(Sp_{+1})$ satisfies. Since the syndrome is fixed, $\boldsymbol{x_1}^{(s)}$ and $\boldsymbol{x_2}^{(s)}$ belong to the same VT codebook, for $s = 0, 1$. However, for any VT codebook described in (1), if $m = 2n$ where n is the codeword length (in our code, there should be k), then their edit error distance and the Hamming distance are both at least 3. This conflicts with the assumption that there is only one edit error within the received sequence $\boldsymbol{r}$. On the other hand, if $\boldsymbol{r}$ belongs to the set $\mathcal{D}(S_2)$, the message in $\boldsymbol{r}$ is correct. We have $\boldsymbol{x_1} = \boldsymbol{x_2}$, and $\boldsymbol{y_1} = \boldsymbol{y_2}$. This contradicts the assumption.

Based on the above discussion, we can see those error domains of different codewords are disjoint. It means that the data-block code can decode each error sequence into a unique codeword.

□

Lemma 2 proved that the data-block code is a single-edit error-correcting code, which can be regarded as a situation where the marker codeword is always correct. Based on this, we now prove that the *DNA-sLM* code is still a single-edit error-correcting code even when the marker codeword has one edit error.

TABLE 2
Table for error types in the received sequence.

Error type	**Marker**	**Check**	**Separator**	**Remainder**
sub	(x, m) or (m, x)	$\mathcal{D}(Ck_0)$	$\mathcal{D}(Sp_0)$	$\mathcal{D}(Syn_0)$
del	m	$\mathcal{D}(Ck_{-1})$	$\mathcal{D}(Sp_{-1})$	$\mathcal{D}(Syn_{-1})$
ins	(x, m, m) or (m, x, m)	$\mathcal{D}(Ck_{+1})$	$\mathcal{D}(Sp_{+1})$	$\mathcal{D}(Syn_{+1})$

Lemma 3. *The DNA-sLM code can correct an edit error.*

Proof. Assuming that the received sequence is $\boldsymbol{r}$. For the consistency of proof, let (r_{-1}, r_0) denote the first two symbols of $\boldsymbol{r}$, which is presumably the marker. We can determine the correctness of the marker code by checking the first two symbols of the received sequence. If $r_{-1} = r_0$, it means that there is no edit error in the marker, and then Lemma 2 works. If $r_0 \neq r_1$, it indicates that an edit error occurred in the marker code due to the specific design of the marker function. It remains to show that the edit error-correcting capability could be realized in this case.

Following previous notations, let $\mathcal{D}(Ck_m)$ denote the event that the received sequence satisfies:

$$Ck\big((r_{1+m}, r_{2+m}, \cdots, r_{k+m})\big) = r_{k+1+m}.$$

The definitions of $\mathcal{D}(Sp_m)$ and $\mathcal{D}(Syn_m)$ are the same as Lemma 2. Here $m = -1, 0, 1$ indicates the symbol-shift. Table 2 lists the events that the received sequence satisfies when different types of errors occur in the marker.

Here, x can be an arbitrary symbol in Σ except for m. Let the error domain of codeword $\boldsymbol{y}$ after taking one edit error in the marker be

$$\mathcal{D}_{marker}(\boldsymbol{y}) \triangleq \Big\{\boldsymbol{r} : \boldsymbol{r} \text{ is obtained from } \boldsymbol{y} \text{ by one edit error}, r_{-1} \neq r_0\Big\}.$$

We now prove that the sets $\mathcal{D}_{marker}(\boldsymbol{y})$ for different $\boldsymbol{y} \in C$ are disjoint.

The structure of the separator ensures that the correct separator meets only one event in Table 2, so that the $k+1$ to 2 symbols before the separator are determined to be the message. In other words, $\mathcal{D}_{marker}(\boldsymbol{y}_1) \neq \mathcal{D}_{marker}(\boldsymbol{y}_2)$ if $\boldsymbol{y}_1 \neq \boldsymbol{y}_2$.

The above analysis shows that when the received sequence satisfies any event in the above list, we do not use the marker to find the beginning of the data-block codeword. Instead, we use the position of the separator to detect the edit error in the marker to regain synchronization. □

To summarize, Lemma 3 shows that the *DNA-sLM* code is a single-edit error-correcting code. Next, we will show that the beginning of the next segment can be determined by the special structure of the *DNA-slM* code so that the decoding can be performed segment-by-segment.

Lemma 4. *In the case of an edit error within the first segment, the decoder can determine the segmented endpoint to keep synchronization, thereby continuing to decode the next segment.*

Proof. The proof is divided into two parts as follows. Firstly, consider the case of no edit error within the syndrome, i.e.,

$$\boldsymbol{r} \in \Big(\mathcal{D}(Ck) \cap \mathcal{D}(Syn)\Big) \cup_{i\in\{-1,0,+1\}} \Big(\mathcal{D}(Sp_i) \cap \mathcal{D}(Syn_i)\Big).$$

In such a situation, it is easy to find the boundary of the syndrome. After the syndrome, it is the beginning of the next segment. Secondly, consider the case of an incorrect syndrome. For brevity, because the separator is correct and can be located, we use $\boldsymbol{r}$ to represent the remaining sequence trimmed after the separator. And we use $\boldsymbol{a} = IBSyn(\boldsymbol{x})$ to represent the syndrome calculated by the message. Let

$$\mathcal{D}(M_1) \triangleq \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_t = r_{t+1} \neq a_t, r_{t+2} \neq r_{t+1}\Big\},$$
$$\mathcal{D}(M_2) \triangleq \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_{t+1} = r_{t+2} \neq a_t\Big\},$$
$$\mathcal{D}(M_3) \triangleq \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_{t+2} = r_{t+3} \neq a_t, r_{t+1} = a_t\Big\}.$$

Recall that the marker function should satisfy the rule: $m \notin \{a, b, c\}$, if m is inserted between a and b, c. Therefore, $\mathcal{D}(M_1)$, $\mathcal{D}(M_2)$ and $\mathcal{D}(M_3)$ are pairwise disjoint sets. And they represent the cases of one deletion, one substitution, and one insertion in the syndrome, respectively, while the marker of the next segment is correct. Then we can trim the received sequence before the correct marker and decode the next segment. However, if the received sequence is not in any above sets, it means that the marker of the next segment has an edit error. Under the assumption that a single edit error within each segment, the separator of the next segment is correct to help the decoder relocate the message and regain synchronization. Therefore, the received sequence can re-synchronize itself after decoding the current segment. □

Based on the above lemmas, we now prove Theorem 1. According to Lemma 2 and Lemma 3, we obtained that the first segment can correct an edit error whether the edit error is in the marker codeword or the data-block codeword. According to Lemma 4, after decoding the first segment, the decoder can determine the beginning of the next segment, so that codeword segments can be decoded in sequence. This means that the *DNA-LM* code is a segmented-edit error-correcting code.

4 Decoder of the DNA Levenshtein-Marker Code

In this section, we show the decoding algorithm of the *DNA-LM* code. Denote a received sequence as $\boldsymbol{r}$. The decoding algorithm is performed segment-by-segment. Specifically, each round contains two phases: decoding the message of the first segment and regaining the synchronization of the next segment.

Step 1: Decode the foremost message segment.

- Case 1: $r_1 = r_2$. The decoder first removes (r_1, r_2) and restarts $\boldsymbol{r}$. Then according to Lemma 2 (6), when the remaining sequence $\boldsymbol{r} \in \mathcal{D}(S_2)$, the message is correct and is the first k symbols of $\boldsymbol{r}$. When $\boldsymbol{r} \in \mathcal{D}(S_3)$, the error type of the message $\boldsymbol{x}$ is as follows:
  1) $\boldsymbol{r} \in \mathcal{D}(Sp_0) \Leftrightarrow \boldsymbol{x}$ has a substitution;
  2) $\boldsymbol{r} \in \mathcal{D}(Sp_{+1}) \Leftrightarrow \boldsymbol{x}$ has an insertion;
  3) $\boldsymbol{r} \in \mathcal{D}(Sp_{-1}) \Leftrightarrow \boldsymbol{x}$ has a deletion.

  The decoder uses the correct syndrome, the t symbols after the separator, to correct two de-interleaved binary message sequences through the Levenshtein decoder.
- Case2: $r_1 \neq r_2$. According to Lemma 3, the decoder checks which $i \in -1, 0, 1$ makes $\boldsymbol{r}$ satisfy $\mathcal{D}(Ck_i) \cap \mathcal{D}(Sp_i) \cap \mathcal{D}(Syn_i)$, and sets the message as $(r_{(i+3)}, \cdots, r_{(k+i+2)})$.
- Case 3: the received sequence does not satisfy any above events. It indicates that more than one edit error within the segment. In this situation, the decoding fails. Each corresponding position of this message segment is marked as an erasure symbol $\boldsymbol{e}$, waiting to be recovered by outer codes.

Step 2: Locate the start of the subsequent segment.

- In Case 1 and Case 2, the start of the subsequent segment is determined according to Lemma 4, and the decoder will restart $\boldsymbol{r}$ from this point;
- In Case 3, the decoder will scan the received sequence sequentially to detect subsequence (a, m, m, b, c) and (a, s, s, s, b, c), where $m = Mk(a, b, c)$ and $s = Sp(a, b, c)$. They jointly point to the position of the next segment with high probability.

Step 3: Repeat steps 1-2 until all segments are decoded.

Following the above steps, the sequence $\boldsymbol{r}$ can be decoded segment-by-segment. In particular, Step 2 shows that the specially designed structure of the data-block code provides a second guarantee for keeping and regaining synchronization of the whole sequence, while the first is the marker code. It makes the decoding of the subsequent segment to still have a chance to process smoothly, even when multiple edit errors occur in a segment.

5 Coding for Duplications

Recall that in a typical DNA-based storage system, digital information is synthesized into many short DNA strands, and each of these strands is massively miscopied through the sequencing process [23]. Errors generated in DNA synthesis will be inherited and appear in the corresponding sequencing readouts. We suppose that error patterns due to synthesis are clustered into a class since they are hardly discarded directly by the algorithm below, but rather need to be corrected by the decoding algorithm. Without loss of generality, we assume that the remaining error patterns of these duplications are independent, and the positions of edit errors within a readout are uniformly random.

To deal with these duplications, a typical approach is to perform a multiple sequence alignment (MSA) of the received sequences and the subsequent a majority decision on the alignment [33], [34]. However, because edit errors make sequence alignment extremely difficult, it is thought to be the stage that takes up the most space and time in DNA-based information storage [10]. For instance, the time complexity of a popular MSA algorithm "MUSCLE" is $O(MN^2 + M^3N)$, where N and M are the length and number of sequences, respectively. Moreover, in experiments for DNA storage, the MSA procedure usually requires more than 100 times the sequencing depth to recover data [7], [8].

In this section, we provide a joint decoding algorithm that combines our code structure and our decoding algorithm with the reconstruction of traces, skipping the complex MSA process. This idea is inspired by the algorithms introduced in [10], which also combines its unique code structure (i.e., the convolutional code) with duplications' decoding via the BCJR algorithm. Compared to MSA techniques, our decoding strategy can process over a single sequence case. Furthermore, some algorithms examine the trace reconstruction problem over a fixed number of sequences [35], [36]. However, these works process only uncoded sequences, whereas we focus on coded transmissions.

We illustrate the generalized decoding method through the following example.

$$\boldsymbol{r}_1 = 0130333123 \quad 0003222012;$$
$$\boldsymbol{r}_2 = 010333123 \quad 00103222012.$$

where $\boldsymbol{y}$ is the codeword of (5), $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are its two duplications with different edit errors.

The decoder operates in two phases. In the first phase, the decoder identifies whether a sequence satisfies the event $\mathcal{D}(Sp_0)$ through the sliding window algorithm. If there are many qualifying sequences, this message segment will be obtained by the majority voting algorithm. Otherwise, the decoder will check whether the unique qualifying sequence belongs to event $\mathcal{D}(S_2)$, in the same way as the algorithm described in section 4. In the example above, $\boldsymbol{r}_1$ satisfies the event $\mathcal{D}(Sp_0)$, thus the decoder will operate on it. In the second phase, the decoder deals with the case that none of the sequences satisfy the event $\mathcal{D}(Sp_0)$ or further $\mathcal{D}(S_2)$, but there exist sequences satisfying the event $\mathcal{D}(Sp_{-1})$ or $\mathcal{D}(Sp_{+1})$, such as the second segment of $\boldsymbol{r}_2$. The message can be obtained by decoding such sequences. However, there could be possible that no sequence satisfies the event $\mathcal{D}(Sp)$, causing the decoding to fail and this message segment to be marked as erasures. Besides, this decoding algorithm also proceeds segment-by-segment, as described in Section 4.

Assuming edit errors are i.i.d. on the DNA strands, it is apparent that the number of qualifying sequences increases with the number of duplications, which further achieves higher coding performance gains. However, we note that duplications also come with an added cost to the sequencing process. Again, our code can work on any scale of duplications.

6 Code Capability

In this section, we analyze the coding performance of the *DNA-LM* code. We use N and n to represent the length of the *DNA-LM* codeword and *DNA-sLM* codeword encoded by the K and k message symbols, respectively. We use l to indicate the number of segments.

6.1 Code Complexity Analysis

For each length-k segment, the encoding procedure consists of calculating four main components of the *DNA-sLM* code: the syndrome, the check, the separator, and the marker. All of these calculations can be accomplished in linear time. The decoding algorithm of one segment can also be computed in time $O(n)$. Since the encoding and the decoding of the entire sequence are performed sequentially among segments, the entire encoding or decoding process can be completed in time $O(N)$.

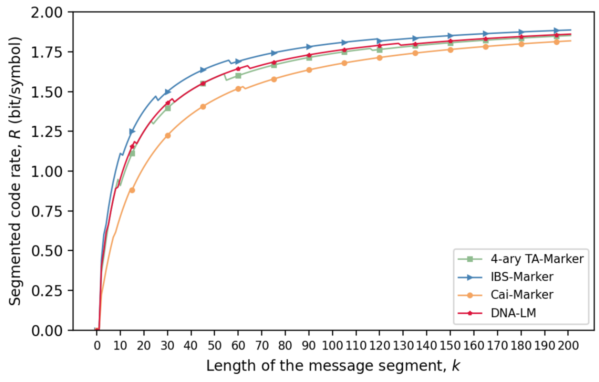
In the case of decoding m duplicated DNA strands, both the majority voting algorithm and the error correction algorithm work on at most m sequences. Therefore, the time complexity is $O(mN)$.

6.2 Code Rate Analysis

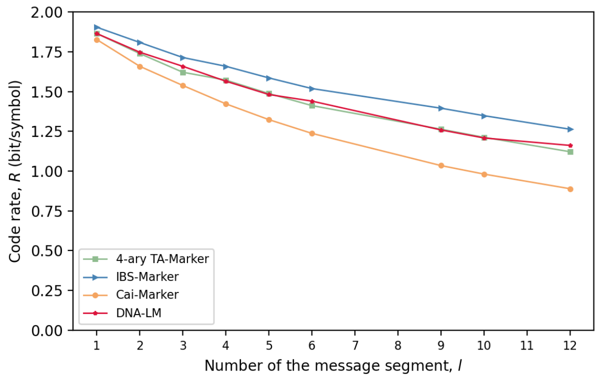
We first consider the segmented code rate of the *DNA-LM* code. Since the *DNA-LM* code is concatenated by a series of *DNA-sLM* codeword segments, and the marker of the first segment is omitted to reduce redundancy, the segmented code rate is asymptotically close to the integrated code rate. As illustrated in Definition 1, there are three symbols for the separator, one symbol for the check, $\lceil \log k \rceil$ symbols for the syndrome, and two symbols for the check in each segment with k-length message symbols. Thus, the segmented code rate of the *DNA-LM* code is

$$R = \frac{2k}{k + \lceil \log k \rceil + 6} \quad \frac{\text{bits}}{\text{symbol}}. \tag{7}$$

By simulations, we plotted the segmented code rates of the *DNA-LM* code, the *4-ary TA-Marker* code, the *IBS-Marker* code and the *Cai-Marker* code with varied segment

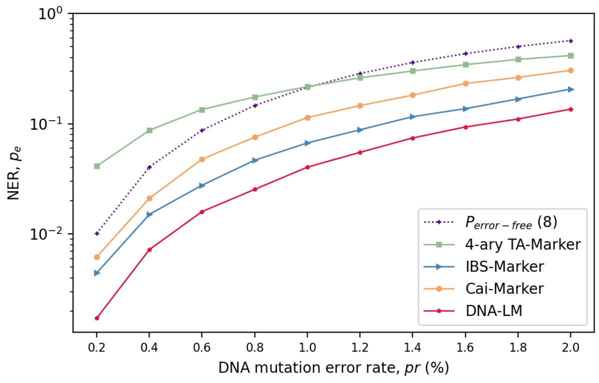


(a) Varied lengths of the message segment

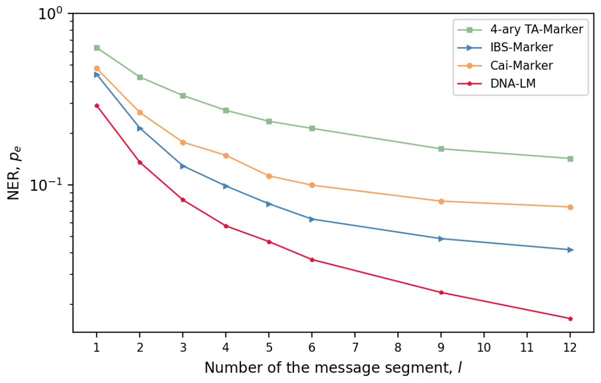


(b) Varied numbers of the message segment ($K = 180$)

Fig. 5. The code rate of our codes. (a) shows the segmented code rate versus the length of the message segments (in nucleotides). (b) shows the code rate versus the number of the message segment with the fixed message sequence length $K = 180$.



(a) Different DNA mutation error rates



(b) Different segment numbers

Fig. 6. NER performance. (a) shows the NER as the function of the DNA mutation error probability, where pr increases from 0 to 2%. (b) shows the NER as the function of the segment number with the DNA mutation error probability $pr = 0.01$.

lengths, as shown in Fig. 5(a). These curves show that when the length of the message segment is less than 200, the segmented code rate of *IBS-Marker* code is the highest. The *DNA-LM* code has a slightly higher code rate than the *4-ary TA-Marker* code, ranking second. The *Cai-Marker* code has the lowest code rate when the length is below 200. This is because the codeword segment of the *Cai-Marker* code requires some constant redundant symbols, although its segmented code is order optimal asymptotically. Indeed, the difference between the segmented code rate of these codes gets smaller as the length of the message segment increases. Furthermore, we calculated the code rate when the segment number l varies with a fixed total message length K (Fig. 5(b)). As expected, when we divided the message into more segments, we observed a lower code rate.

6.3 Decoding Error Rate

We now test the edit error-correcting capability of our codes through the inner channel introduced in Section 2.2. Under the channel assumptions, the error probabilities are shown in Table 3.

Given that the codeword sequence is concatenated by l such segments, the probability of at most one edit error within each segment as

TABLE 3
Table for error probabilities.

Total error number	Insertion	Deletion	Substitution	Probability
0	0	0	0	$p_t(1-p_s)$
1	0	1	0	p_d
1	0	0	1	$p_t p_s$
1	1	0	0	$p_i p_t(1-p_s)$
1[1]	1	1	0	$p_i p_d$

[1] Given that a deletion after a random insertion can be seen as a substitution, this case also introduces only one edit error.

$$P_L = (p_0^n + np_0^{n-1}p_1)^l, \tag{8}$$

where $p_0 = p_t(1–p_s)$ and $p_1 = p_d+p_tp_s+p_ip_t(1–p_s)+p_ip_d$. This is the lower bound on the theoretical probability of error-free decoding.

Next, we tested NER of the *DNA-LM* code, the *4-ary TA-Marker* code, the *IBS-Marker* code, and the *Cai-Marker* code at varied conditions through simulations. The NER is defined as the ratio of the total number of residual incorrect symbols and erasures compared to the original message sequence. We assumed that the DNA mutation error rate per nucleotide $pr = p_i + p_d + p_s$, and $p_s : p_i : p_d = 2 : 1 : 1$, which roughly approximates the proportion of errors in Illumina sequencing. The parameters and their default values are listed in Table 4. Without losing generality, we first consider the case where $m = 1$, i.e., there are no duplications.

TABLE 4
Parameters used for the simulations.

Parameter	Paraphrase	Default value
K	length of the message sequence	180
k	length of the message segment	30
l	number of the segments	6
m	number of duplications	1
pr	DNA mutation error rate	0.01
T	number of simulations	10000

We first plotted the NER as a function of the DNA mutation error rate. As shown in Fig. 6(a), when the hyper-parameters (k, l, pr) are the same, the *DNA-LM* code achieves the lowest nucleotide error rate. The *4-ary TA-Marker* code shows the worst performance because it can only correct indel errors but no substitutions. The *IBS-Marker* code and the *Cai-Marker* code achieve a moderate performance, due to the limitation that they can only correct a single edit error in each data-block codeword, while edit errors in marker codewords may disrupt synchronization. We further compared the lower bound of the theoretical probability of error-free decoding (8) (dotted line) with simulations (solid lines). Simulations indicate that our segmented-edit error-correcting codes have achieved better performance than theoretical results. This is because our encoders and decoders are carefully designed to cope with some, although not all, multiple edit errors within segments to avoid out-of-sync events. Fig. 6(b) illustrates that the NER of codes decreases as the number of segments increases. It is reasonable to assume that it can achieve a lower NER by separating sparse errors into different segments.

To investigate the bias in edit error-correcting capabilities of different segments, we plotted the NER per position in Fig. 7. The curve of the *DNA-LM* code seems more volatile than other codes, but this is due to the setting of the coordinate system. All curves show a small jump between adjacent segments, as decoding may lose synchronization if multiple edit errors within a segment. In addition, we noticed a slight decrease in the last segment. One explanation is that the boundary of the last segment is determined by the length of the sequence so that the decoding can be re-synchronized.

To focus more on the *DNA-LM* code, we further plotted the NER as a function of the DNA mutation error rate with varied numbers of segments (Fig. 8). It shows that the NER decreases significantly as the number of segments increases. Besides, since the code rate increases with the number of segments (Fig. 5(b)), there is a trade-off between the code rate and the NER. In practice, the error probabilities may not be identically distributed among positions. Since the encoding of each message segment is independent of each other, one can set the lengths of message segments to vary with the error probability within DNA strands. For example, considering that the two ends of the DNA strand are more prone to errors [23], the encoder can choose a shorter segment length towards the ends of the DNA strands.

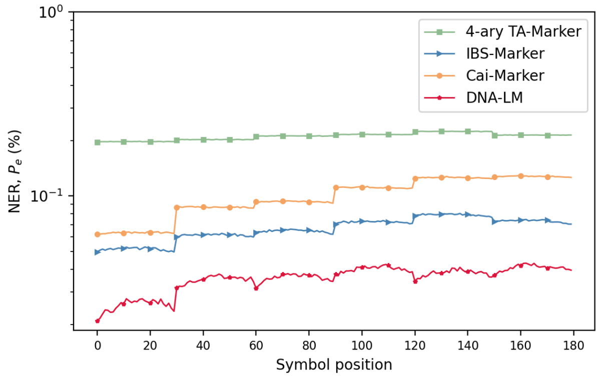


Fig. 7. NER performance at different positions in the message sequence.

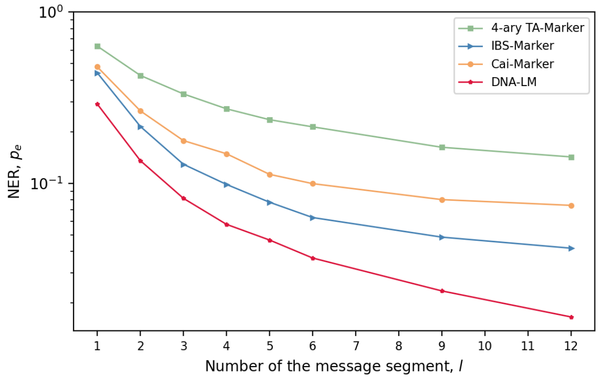


Fig. 8. NER performance of the *DNA-LM* code with varied numbers of the message segment.

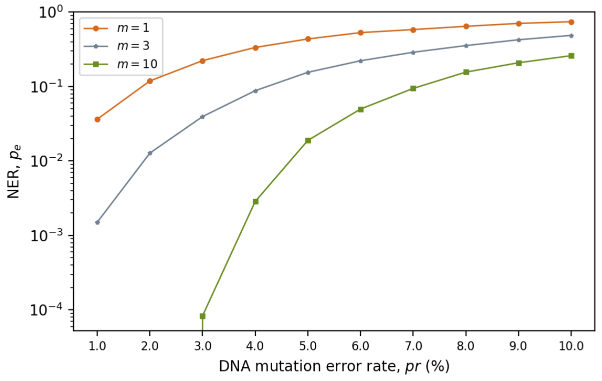


Fig. 9. NER performance of the *DNA-LM* code with varied numbers of duplications.

As depicted in Fig. 9, we can observe how the NER decreases with the number of duplications. In agreement with the results of Section 5, the NER is significantly lower for multiple reads as compared to a single read.

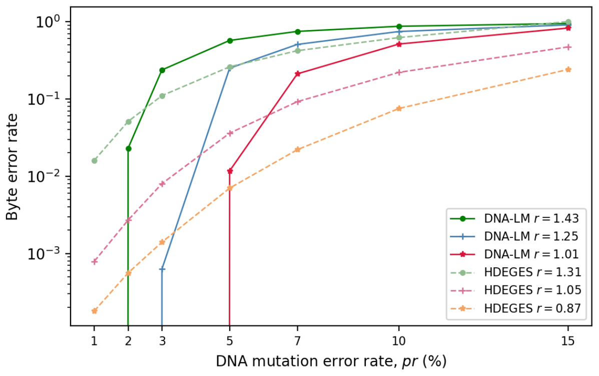


Fig. 10. Byte error rate performance of the cascade coding scheme.

Finally, we verified that the messages could be recovered without error via concatenating an outer code (specifically, (255,223) RS code) to our *DNA-LM* code, but only up to a maximum tolerable error rate that increased with decreasing code rate. As an initial validation of coding performance, we first randomly generated messages of length 223×31 bytes. Second, as the outer encoding, we used a concatenated coding scheme to produce 255 strands via an outer code (i.e., the (255,223) RS code) and appended a unique 1-byte prefix (i.e., 0 to 255) to each strand as the index. Third, as the inner encoding, we encoded each strand of length 128 quaternary symbols via our *DNA-LM* code with the segment of length $(16, 32, 64)$, yielding DNA strands of length $(214, 174, 152)$. To simulate the overall DNA storage channel, we pooled and shuffled all of the strands and then duplicated them as if sequencing to depth 5, i.e., each strand was duplicated a Poisson random number of times, with a mean of 5. We further induced stochastic edit errors to each strand in the pool based on the inner channel model, where the total error rate p_r traversed $\{0.01, 0.02, 0.03, 0.05, 0.07, 0.10, 0.15\}$ and $p_s : p_i : p_d = 1 : 1 : 1$. We constructed a statistical error model to analyze the byte error rate.

As a result, Fig. 10 shows the byte error rate as a function of the DNA mutation error rate. We also plotted the decoding performance of [12], in which the inner code is the HEDGES code and the outer code is the same (255,223) RS code. The HEDGES code is a well-designed convolutional code and is able to correct stochastic edit errors, however, its decoding complexity is $O(2^n)$. The comparison indicates that at close code rates, although our *DNA-LM* code performs poorly at high raw error rates, it allows a higher raw error rate for error-free decoding than [12]. Specifically, the *DNA-LM* code with rate $r = 1.47$ is capable of error-free decoding at 1% DNA mutation error rate (i.e., via the "next-gen" sequencing).

7 Conclusion

In this work, we have designed a new segment-edit error-correcting code, called the *DNA-LM* code. Each codeword segment is a concatenation of a marker codeword and a data-block codeword. It is distinctive in that it does not require segment endpoints as a prerequisite for decoding, but rather can decode segment by segment through its special structure. Another advantage of the *DNA-LM* code is that its data-block code also has the same re-synchronization function as the marker code, but without the additional code redundancy. It makes the *DNA-LM* code more likely than other segment-error-correcting codes to restart decoding, even if multiple edit errors occur within a segment. Furthermore, it can be simply generalized to decode duplicated DNA strands. Simulations showed that the *DNA-LM* code is capable of error-free decoding for the "next-gen" sequencing readouts via concatenating with an efficient outer code (e.g., the (255,223) RS code).

To summarize, the *DNA-LM* code is useful when the amount of data increases: (i) its codewords are systematic; (ii) it only takes linear time complexity for encoding and decoding; (ii) it allows varying the code rate, with correspondingly greater tolerance of edit errors at lower code rates; (iv) it produces erasures when segmented decoding fails, making it easier for outer code to recover the residual errors; (v) when generalized to decoding duplicated DNA strands, it still maintains linear time complexity; (vi) it enables error-free decoding at a low row error rate. Given this, the *DNA-LM* code can be applied to a wide range of storage systems with synchronization errors, especially being used as an inner code in the DNA-based storage system.

For future work, in order to cope with more complex error cases and effectively correct the burst errors that may occur in a DNA strand, we will further combine a class of burst-error-correcting codes (e.g., RS codes) with our *DNA-LM* code.

Acknowledgment

This work is supported by the National Key R&D Program of China (grant No. 2020YFA0712100), the National Natural Science Foundation of China (grant No. 12001401 and 62071327) and Tianjin Science and Technology Planning Project (grant No. 22ZYYYJC00020).

References

[1] F. Tavella, A. Giaretta, T. M. Dooley-Cullinane, M. Conti, L. Coffey, and S. Balasubramaniam, "DNA molecular storage system: Transferring digitally encoded information through bacterial nanonetworks," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1566–1580, jul 2021.

[2] S. Ebrahimi, R. Salkhordeh, S. A. Osia, A. Taheri, H. R. Rabiee, and H. Asadi, "RC-RNN: Reconfigurable cache architecture for storage systems using recurrent neural networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1492–1506, 2021.

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

[4] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. Leproust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, 01 2013.

[5] 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," *Angew Chem Int Ed Engl*, vol. 54, no. 8, pp. 2552–2555, 2015.

[6] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 3, pp. 230–248, 2015.

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

[8] L. Organick, S. D. Ang, Y. J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, and B. Nguyen, "Random access in large-scale DNA data storage," *Nature Biotechnology*, vol. 36, no. 3, 2018.

[9] I. Shomorony and R. Heckel, "DNA-based storage: Models and fundamental limits," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3675–3689, 2021.

[10] 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*, pp. 1–1, 2022.

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

[12] W. Press, J. Hawkins, S. Jones, J. Schaub, and I. Finkelstein, "HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints," *Proceedings of the National Academy of Sciences*, vol. 117, no. 31, pp. 18 489–18 496, 2020.

[13] S. Chandak, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, T. Weissman, and H. Ji, "Improved read/write cost tradeoff in DNA-based data storage using LDPC codes," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, pp. 147–156.

[14] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–280.

[15] 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.

[16] L. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552–2557, 1999.

[17] B. Haeupler and A. Shahrasbi, "Synchronization strings and codes for insertions and deletions—a survey," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3190–3206, 2021.

[18] P. L. Antkowiak, J. Lietard, M. Z. Darestani, M. M. Somoza, W. J. Stark, R. Heckel, and R. N. Grass, "Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction," *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.

[19] M. Abroshan, R. Venkataramanan, and A. Guillén i Fàbregas, "Coding for segmented edit channels," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 3086–3098, 2018.

[20] K. Cai, H. M. Kiah, M. Motani, and T. T. Nguyen, "Coding for segmented edits with local weight constraints," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1694–1699.

[21] 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.

[22] D. C. H. Tan, "Single edit correcting code," https://github.com/dtch1997/single-edit-correcting-code.git.

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

[24] Z. Liu and M. Mitzenmacher, "Codes for deletion and insertion channels with segmented errors," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 224–232, 2010.

[25] F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, 1962.

[26] R. Varšamov and G. Tenengolts, "A code which corrects single asymmetric errors," *Avtomat. i Telemeh*, vol. 26, no. 4, pp. 288–292, 1965.

[27] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965.

[28] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[29] M. Abroshan, R. Venkataramanan, and A. G. I. Fabregas, "Efficient systematic encoding of non-binary VT codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 91–95.

[30] K. A. Abdel-Ghaffar and H. C. Ferreira, "Systematic encoding of the Varshamov-Tenengol'ts codes and the Constantin-Rao codes," *IEEE Transactions on Information Theory*, vol. 44, no. 1, pp. 340–345, 1998.

[31] K. Saowapa, H. Kaneko, and E. Fujiwara, "Systematic deletion/insertion error correcting codes with random error correction capability," in *Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems*, 1999, pp. 284–292.

[32] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, "Beyond single-deletion correcting codes: Substitutions and transpositions," *IEEE Transactions on Information Theory*, pp. 1–1, 2022.

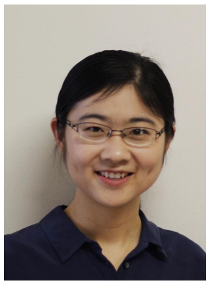
[33] R. C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic acids research*, vol. 32, no. 5, pp. 1792–1797, 2004.

[34] K. Reinert, T. H. Dadi, M. Ehrhardt, H. Hauswedell, S. Mehringer, R. Rahn, J. Kim, C. Pockrandt, J. Winkler, E. Siragusa, G. Urgese, and D. Weese, "The SeqAn C++ template library for efficient sequence analysis: A resource for programmers," *Journal of Biotechnology*, vol. 261, pp. 157–168, 2017.

[35] M. Cheraghchi, J. Ribeiro, R. Gabrys, and O. Milenkovic, "Coded trace reconstruction," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[36] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin, "Trellis BMA: Coded trace reconstruction on IDS channels for DNA storage," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2453–2458.

Zihui Yan received the B.S. degree in mathematics from Sichuan University, Chengdu, China, in 2018. She is currently working toward the PhD degree at Tianjin University, Tianjin, China, under the supervision of Professor William Y.C. Chen. Her main interests include error control coding and communication algorithms.



Cong Liang received her B.S. degree from Peking University in 2012 and her Ph.D. degree from Yale University in 2018. She is currently working at the Center for Applied Mathematics, Tianjin University. Her research interests include applications of statistical methods in engineering and biology.

Huaming Wu received his B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Freie Universität Berlin, Germany in 2015. He is currently an Associate Professor at the Center for Applied Mathematics, Tianjin University, China. His research interests include edge computing, internet of things, complex networks, and DNA storage.