# Correcting a substring edit error of bounded length

Yuanyuan Tang*, Sarvin Motamen*, Hao Lou*, Kallie Whritenour§,
Shuche Wang†, Ryan Gabrys‡, and Farzad Farnoud*§
*Electrical & Computer Engineering, §Computer Science, University of Virginia, U.S.A.,
{yt5tz,awz4up,hl2nu,kw5km,farzad}@virginia.edu
†Institute of Operations Research and Analytics, National University of Singapore, shuche.wang@u.nus.edu
‡Calit2, University of California-San Diego, U.S.A., rgabrys@ucsd.edu

***Abstract*—Localized errors, which occur in windows with bounded lengths, are common in a range of applications. Such errors can be modeled as $k$-*substring edits*, which replace one substring with another string, both with lengths upper bounded by $k$. This generalizes errors such as localized deletions or burst substitutions studied in the literature. In this paper, we show through statistical analysis of real data that substring edits better describe differences between related documents compared to independent edits, and thus commonly arise in problems related to data synchronization. We also show that for the dataset under study, assuming codes exist that can achieve the Gilbert-Varshamov (GV) bound, substring-edit-correcting codes can synchronize two documents with much lower overhead compared to general indel/substitution-correcting codes. Furthermore, given a constant $k$, we construct binary codes of length $n$ for correcting a single $k$-substring edit that achieves the GV bound and subsequently has redundancy of asymptotically $2\log n$, compared to $4k\log n$, the lowest redundancy achievable by an existing code for this problem. The time complexities of both encoding and decoding are polynomial with respect to $n$.**

***Index Terms*—Localized errors, $k$-substring edit, GV bound, statistical analysis, error-correcting codes, low redundancy**

## I. Introduction

In a wide range of applications, including wireless communication, magnetic data storage, DNA data storage, and document synchronization, errors that appear close to each other are common. Such errors are referred to as burst errors or localized errors. Due to the prevalence of these errors, problems related to their correction have drawn significant attention [1]–[11]. The focus of the present paper lies in correcting a $k$-*substring edit* error, where one substring is replaced with another string at the same position, with both substrings having lengths bounded by $k$. Since substring edits encompass various specific errors, including burst insertions, burst deletions, and burst substitutions, as well as combinations of these errors occurring in a bounded window, our work presents a unified solution to correcting a diverse set of burst/localized errors.

We first present an experimental analysis demonstrating the prevalence of burst errors in real data, which to the best of our knowledge is the first such analysis. Specifically, statistical analysis of two datasets, one containing two versions of the Bash source code and the other comprising noisy nanopore DNA sequencing reads, strongly supports rejecting the uniform distribution of errors/edits in these sequences. Furthermore, through this analysis, we demonstrate that for these representative datasets, substring edit-correcting codes achieve a lower redundancy compared to existing codes for correcting indels and substitutions.

Codes for correcting burst/localized errors have been studied by many recent works, including codes for bursts of substitutions [2], [4], a burst of exactly $k$ deletions [5]–[7], a burst of at most $k$ deletions (or $k$ insertions) [6], [8], [11], [12], and localized deletions occurring in a window with bounded length $k$ [9], [10]. The problem of correcting a substring edit, studied here, is closest to correcting deletions and bursts of deletions.However, as we will show in Lemma 1, a code that can correct a burst of at most $\ell$ deletions (or one that can correct a burst of at most $\ell$ insertions) cannot necessarily correct a $k$-substring edit, even if $\ell$ is much larger than $k$. On the other hand, a $k$-substring edit can be corrected by a code that can correct 2 bursts of at most $k$ deletions or, as we show in Lemma 1, by a code that can correct at most $2k$ deletions. These observations lead to the conclusion that existing codes for correcting multiple deletions [13], [14] or multiple bursts of deletions [15] can correct a $k$-substring edit with redundancy at least $4k\log n$ [14]. The goal of the current work is to construct codes that can correct a $k$-substring edit with less redundancy.

While codes for correcting a burst of at most $k$ deletions cannot correct a $k$-substring edit, the idea of first identifying an approximate location for the error presented in Lenz et al. [8] and Bitar et al. [9] using the position of specific patterns is useful for our problem. We divide $k$-substring edits into *strict* $k$-substring edits (that will change the length in the outputs) and bursts of at most $k$ substitutions, referred to as $k$-burst substitutions. For strict edits, we first extend the codes in [8], [9] to locate the error to be in an interval of length $O((\log n)^2)$ and then correct it. For $k$-burst substitutions, which cannot be located using the patterns mentioned above, we adapt the Fire code [2]. Then we combine the two error-correcting codes in a manner that enables polynomial-time encoding and decoding.

Compared to the conference version of this work [1], the current paper provides omitted proofs, presents models for all hypothesis tests in Section III, analyzes the average redundancy of correcting substring edit errors for the datasets discussed in Table II, and provides a more detailed analysis of Figure 4, regarding the differences between pairs of files in

This paper was presented in part at the IEEE ISIT2023 [1].
This work was supported in part by NSF grants under grant nos. CCF-1816409, CAREER-2144974, CCF-2212437, and CCF-2312871.

the Bash dataset.

The rest of the paper is organized as follows. Section II presents the notation and preliminaries. Section III discusses the prevalence of burst errors in real data and the utility of substring-edit-correcting codes. Finally, Section IV presents the code constructions and an analysis of the time complexity of encoding/decoding and the redundancy.

## II. Notation and Preliminaries

### A. Notation

Without loss of generality, let $\Sigma_q = \{0, 1, \ldots, q-1\}$ be a finite alphabet of size $q$. The set of length-$n$ strings and finite strings over $\Sigma_q$ are denoted as $\Sigma_q^n$ and $\Sigma_q^*$, respectively. The empty string, denoted $\Lambda$, is also considered a member of $\Sigma_q^*$. In this paper, we focus on the binary alphabet $\Sigma_2$. For $a, b \in \mathbb{Z}$, let $[a, b] = \{a, a+1, \ldots, b\}$ and $[b] = [1, b]$. Unless otherwise stated, logarithms are to the base of 2.

For $\boldsymbol{x}, \boldsymbol{y} \in \Sigma_2^n$, let $\boldsymbol{x}_{[a,b]} = x_a \cdots x_b$ and $\boldsymbol{x}_{[a,b)} = x_a \cdots x_{b-1}$, and let $\boldsymbol{xy}$ and $(\boldsymbol{x}, \boldsymbol{y})$ denote the concatenation of $\boldsymbol{x}, \boldsymbol{y}$. For $\boldsymbol{x}, \boldsymbol{v} \in \Sigma_q^*$, $\boldsymbol{v}$ is a substring of $\boldsymbol{x}$ if $\boldsymbol{x} = \boldsymbol{uvw}$ for some $\boldsymbol{u}, \boldsymbol{w} \in \Sigma_q^*$. Furthermore, $|\boldsymbol{x}|$ is the length of a sequence $\boldsymbol{x}$ and $\|S\|$ is the number of elements in a set $S$. Given an integer $r$ and a symbol $a \in \Sigma_2$, let $a^r$ denote a *run* of $r$ consecutive symbols $a$. Given an integer string $\boldsymbol{x} \in \mathbb{Z}^n$, define the *Varshamov-Tenengolts (VT) check sum* as $VT(\boldsymbol{x}) = \sum_{i=1}^n i x_i$.

### B. The $k$-substring edit channel

Given a string $\boldsymbol{x}$, a *$k$-burst deletion* (resp. *insertion*) in $\boldsymbol{x}$ removes (resp. inserts) at most $k$ consecutive symbols, while a *$k$-substring edit* replaces a substring $\boldsymbol{v}$ of $\boldsymbol{x}$ by another string $\boldsymbol{v}'$, where $|\boldsymbol{v}|, |\boldsymbol{v}'| \le k$ and at least one of $\boldsymbol{v}, \boldsymbol{v}'$ is non-empty. The $k$-substring edit is a *$k$-burst substitution* if $|\boldsymbol{v}| = |\boldsymbol{v}'|$ and we say it is a *strict* $k$-substring edit otherwise. In particular, $\boldsymbol{v}' = \Lambda$ results in a burst deletion addressed by previous works. For example, given $\boldsymbol{x} = 1001\underline{1101}1101 \in \Sigma_2^n$, a 4-substring edit may generate $\boldsymbol{z} = 1001\underline{010}1101$ by replacing $\boldsymbol{x}_{[5,8]} = 1101$ with $\boldsymbol{z}_{[5,7]} = 010$.

The next lemma discusses the relationship between deletion-correcting codes and codes that can correct a $k$-substring edit.

**Lemma 1.** *The codes in the statements below are over $\Sigma_q^n$, where $n, q \ge 2$. Let $k$ be a positive integer.*

1) *A code that can correct 2 $k$-burst deletions can correct a $k$-substring edit.*
2) *For any $\ell < n$, there exists a code that can correct a burst of at most $\ell$ deletions but not a $k$-substring edit.*
3) *For any $\ell < n$, if $\ell < 2k$, then there exists a code that can correct up to $\ell$ deletions but not a $k$-substring edit.*

*Proof:*

1) Let $S(\boldsymbol{x})$ denote the set of all strings that can be generated from $\boldsymbol{x}$ by a $k$-substring edit. Let $\boldsymbol{x} = \boldsymbol{a}\boldsymbol{x}_1\boldsymbol{b}\boldsymbol{x}_2\boldsymbol{c}$, $\boldsymbol{y} = \boldsymbol{a}\boldsymbol{y}_1\boldsymbol{b}\boldsymbol{y}_2\boldsymbol{c}$, and $\boldsymbol{z} = \boldsymbol{a}\boldsymbol{y}_1\boldsymbol{b}\boldsymbol{x}_2\boldsymbol{c}$ with $|\boldsymbol{x}_1|, |\boldsymbol{x}_2|, |\boldsymbol{y}_1|, |\boldsymbol{y}_2| \le k$. Then $\boldsymbol{z} \in S(\boldsymbol{x}) \cap S(\boldsymbol{y})$. Furthermore, $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{c}$ are substrings of both $\boldsymbol{x}$ and $\boldsymbol{y}$. Let $\boldsymbol{z}' = \boldsymbol{abc}$. Then $\boldsymbol{z}'$ can be generated by simply deleting $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from $\boldsymbol{x}$ and similarly deleting $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ from $\boldsymbol{y}$. Since $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{y}_1$, and $\boldsymbol{y}_2$ are substrings with length bounded by $k$, $\boldsymbol{z}'$ is generated from $\boldsymbol{x}$ and $\boldsymbol{y}$ by 2 $k$-burst deletions. If an error-correcting code $\mathcal{C}$ can correct 2 $k$-burst deletions and $\boldsymbol{x} \in \mathcal{C}$, then it can distinguish $\boldsymbol{x}$ from $\boldsymbol{y}$ by $\boldsymbol{z}'$ as well as $\boldsymbol{z}$.
2) Consider the code $\mathcal{C} = \{0^n, 10^{n-2}1\}$, which can correct any burst of at most $\ell < n$ deletions since $0^n$ can only produce $0^m, m \in [n]$, while the $10^{n-2}1$ cannot. Note that $0^{n-1}1$ can be obtained from both $0^n$ and $10^{n-2}1$ through a single substitution. So $\mathcal{C}$ cannot correct a $k$-substring edit for $k > 0$.
3) Let $h = \min\{n, 2k\}$ and note that $\ell < h$. Consider the code $\{0^h\boldsymbol{v}, 1^h\boldsymbol{v}\}$, $\boldsymbol{v} \in \Sigma_q^{n-h}$, which can correct any $\le \ell$ deletions. However, each of the codewords can produce $0^{\lfloor h/2 \rfloor}1^{\lceil h/2 \rceil}\boldsymbol{v}$ via a $k$-substring edit.

■

Given a code $\mathcal{C} \subseteq \Sigma_q^n$, the redundancy of the code $\mathcal{C}$ is defined as $n \log q - \log \|\mathcal{C}\|$. For binary, which is the focus of our code construction, part 1 of the above lemma implies the code given in [14] for correcting $\le 2k$ deletions can correct a $k$-substring edit over the binary alphabet with the redundancy of asymptotically $4k \log n$ bits. Furthermore, the $k$-substring edit can also be viewed as $s$ substitutions and $h$ insertions (or deletions) with $s + h \le k$, then the code in [14] can correct a substring edit with the redundancy of asymptotically $4k \log n$ bits. To the best of our knowledge, that is the lowest redundancy that can be achieved by an existing code for this problem. The code we present in Theorem 19 has redundancy of asymptotically $2 \log n$.

Given a string $\boldsymbol{x} \in \Sigma_q^n$, let $D_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^*$ denote the set of strings generated from $\boldsymbol{x}$ by at most $b$ $k$-substring edit errors and let $B_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^n$ denote the *confusable set* of $\boldsymbol{x}$, i.e., the set of sequences $\boldsymbol{y}$ other than $\boldsymbol{x}$ for which $D_{b,k}(\boldsymbol{x}) \cap D_{b,k}(\boldsymbol{y}) \neq \varnothing$. When $b = 1$ and $k$ is clear from the context, we use $D(\boldsymbol{x}), B(\boldsymbol{x})$ instead of $D_{b,k}(\boldsymbol{x}), B_{b,k}(\boldsymbol{x})$.

We now find the *Gilbert-Varshamov (GV) bound* on the size of the code. Define $r_n(b, k) = \max_{\boldsymbol{x} \in \Sigma_q^n} \|B_{b,k}(\boldsymbol{x})\| + 1$.

**Lemma 2.** *Assuming an alphabet of size $q$, we have*

$$r_n(b, k) \le (n + bk)^{2b}(k+1)^{4b} q^{2kb}$$

*and there exists a code $\mathcal{C} \subseteq \Sigma_q^n$ of length $n$ capable of correcting at most $b$ $k$-substring edits with the size at least*

$$\|\mathcal{C}\| \ge q^n / r_n(b, k).$$

*Proof:* For $\boldsymbol{x} \in \Sigma_q^n$, let $B_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^n$ denote the set of sequences that can produce the same output as $\boldsymbol{x}$ by at most $b$ $k$-substring edits. That is $\boldsymbol{y} \in B_{b,k}(\boldsymbol{x})$ if and only if there exists $\boldsymbol{z}$ that can be produced from both $\boldsymbol{x}$ and $\boldsymbol{y}$ through at most $b$ $k$-substring edit errors. Furthermore, each $k$-substring edit is reversible, i.e., if $\boldsymbol{x}_i \in D(\boldsymbol{x}_{i-1})$, then $\boldsymbol{x}_{i-1} \in D(\boldsymbol{x}_i)$. Then each $\boldsymbol{y} \in B_{b,k}(\boldsymbol{x})$ can be generated from $\boldsymbol{x}$ by $\boldsymbol{x} \to \boldsymbol{x}_1 \to \cdots \to \boldsymbol{x}_{b-1} \to \boldsymbol{z} \to \boldsymbol{y}_{b-1} \to \cdots \to \boldsymbol{y}_1 \to \boldsymbol{y}$ by $2b$ $k$-substring edits, where $\boldsymbol{x} = \boldsymbol{x}_0$ and $\boldsymbol{y} = \boldsymbol{y}_0$. This sequence of edits consists of at most $2b$ burst deletions and 2b burst insertions. There are $\le (k+1)^{4b}$ ways to choose their lengths and $\leq q^{2kb}$ to choose the inserted strings. We claim, to be proved later, that for each string, there are at most $n + bk$ possible positions for the edit, yielding the Lemma.

To prove the claim, note that for a string of length $m$, there are $m + 1$ possible positions for a substring edit that involves only an insertion and $m$ positions if the edit contains a nontrivial deletion. The strings $\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}$ either have length less than $n + bk$ or if their length is equal to $n + bk$ (only possible for $\boldsymbol{z}$), then the edit must contain a deletion. ■

Assuming $b, k$ are constants, the redundancy is bounded above by $2b \log n + o(\log n)$, which is the same as the redundancy of the efficiently encodable/decodable codes proposed in this paper for $b = 1$ and the binary alphabet.

## *C. Relevant Prior Results*

We first recall a result from *syndrome compression*, a technique used to construct codes with low redundancy [16], to our problem. A function $f : \Sigma_n^2 \to \mathbb{N}$ is a *labeling function with respect to sets* $\{B(\boldsymbol{x}) : \boldsymbol{x} \in \Sigma_n^2\}$ if for any $\boldsymbol{x} \in \Sigma_n^2$ and any $\boldsymbol{y} \in B(\boldsymbol{x})$, we have $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$.

**Theorem 3** (c.f. [16, Theorem 5]). *Let $f : \Sigma_2^n \to \Sigma_{2^{\mathcal{R}(n)}}$ be a labeling function with respect to sets $\{B(\boldsymbol{x}) : \boldsymbol{x} \in \Sigma_n^2\}$, where $\mathcal{R}(n) = o(\log \log n \cdot \log n)$. Then, for $\boldsymbol{x} \in \Sigma_2^n$, there exists an integer $a \leq 2^{\log \|B(\boldsymbol{x})\| + o(\log n)}$ such that for all $\boldsymbol{y} \in B(\boldsymbol{x})$, we have $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a$.*

We will use the following definitions and results from [8], [9]. These works correct a burst of deletions with low redundancy by first locating the approximate position of the error.

Given sequences $\boldsymbol{x} \in \Sigma_2^n$ and a *pattern* (string) $\mathcal{P}$, define $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x}) \in \Sigma_2^n$ as the indicator vector whose $i$th element is 1 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} = \mathcal{P}$ and is 0 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} \neq \mathcal{P}$ or $i + |\mathcal{P}| - 1 > n$. Further, let $n_{\mathcal{P}}(\boldsymbol{x})$ denote the number of 1's in $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ represent a length-$(n_{\mathcal{P}}(\boldsymbol{x}) + 1)$ vector whose $j$th element is the distance between positions of the $(j-1)$-th and the $j$th 1 in the string $(1, \mathbf{1}_{\mathcal{P}}(\boldsymbol{x}), 1)$ for $j \in [n_{\mathcal{P}}(\boldsymbol{x}) + 1]$. A sequence $\boldsymbol{x}$ is $(\mathcal{P}, \delta)$-dense if each interval of length $\delta$ in $\boldsymbol{x}$ contains at least one substring $\mathcal{P}$, implying that each element of $a_{\mathcal{P}}(\boldsymbol{x})$ is at most $\delta$, which is denoted by $a_{\mathcal{P}}(\boldsymbol{x}) \leq \delta$. The set of $(\mathcal{P}, \delta)$-dense binary strings of length $n$ is denoted by $\mathcal{D}_{\mathcal{P},\delta}(n)$. Based on [8, Lemma 1], for $\mathcal{P} = 0^k 1^k$, $n \geq 5$, and $\delta = k 2^{2k+1} \lceil \log n \rceil$, we have

$$\|\mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n\| \geq 2^{n-1}. \tag{1}$$

For $j \geq 2$, we have $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})_j \geq 2k$ due to the length of $\mathcal{P}$.

Consider $\boldsymbol{x} \in \Sigma_2^n$ and $\boldsymbol{y} \in D(\boldsymbol{x})$, with $m = |\boldsymbol{y}|, k' = |\boldsymbol{x}| - |\boldsymbol{y}|$. The error transforming $\boldsymbol{x}$ to $\boldsymbol{y}$ has occurred in interval $[j, j + \ell - 1]$ in $\boldsymbol{y}$ if $\boldsymbol{y}_{[1,j-1]} = \boldsymbol{x}_{[1,j-1]}$ and $\boldsymbol{y}_{[j+\ell,m]} = \boldsymbol{x}_{[j+\ell+k',n]}$. We say that a decoder can locate an error in an interval of length $\ell$ if it can find $j$ such that the error has occurred in interval $[j, j + \ell - 1]$ in the received word. We next recall the code in [9] used to locate the burst of deletions or localized deletions in an interval.

**Lemma 4** (cf. [9]). *Let $\mathcal{P} = 0^k 1^k$. Given integers $c_1 \in [0, 4]$, $c_2 \in [0, 6n - 1]$, and $\delta = k 2^{2k+1} \lceil \log n \rceil$, there exists a code*

$$\begin{aligned} \mathcal{C}_d = \{&\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) = c_1 \mod 5, \\ &VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) = c_2 \mod 6n.\} \end{aligned} \tag{2}$$

*that can locate a burst of deletions or localized deletions in an interval with length $O((\log n)^2)$.*

## III. Substring edits in nanopore sequencing and document editing

In this section, we investigate the hypothesis that in real-world settings, errors/edits commonly occur in a bursty manner, rather than being distributed uniformly. We also study whether substring-edit-correcting codes can achieve lower redundancy than general edit-correcting codes. We performed experimental studies on two real-world datasets, corresponding to two applications of the codes:

- *Error-correction:* We investigate the set of errors encountered in nanopore sequencing [17] in DNA data storage. The data consists of 1000 input-output pairs, where the input represents the (true) base sequence of a DNA molecule. Each input sequence is of length 200 [1], with bases randomly generated from $\{A, C, G, T\}$. The output represents the sequence detected by nanopore, as simulated by using the nanopore deep simulator [19] and the Nanopore's Guppy basecaller.
- *Data synchronization:* Suppose Alice, who knows only $\boldsymbol{x}$, needs to communicate $\boldsymbol{x}$ to Bob, who knows only $\boldsymbol{z}$. Consider two documents $\boldsymbol{x}$ and $\boldsymbol{z}$, where $\boldsymbol{x}$ is the edited version of $\boldsymbol{z}$. This task is referred to as *data synchronization*, and can be accomplished using error-correcting codes [20]. We study the characteristics of the editing process, which affects the number of bits needed for synchronization. The dataset consists of versions 5.0 and 5.1 of the source code for the Bash utility[2], where each common file in version 5.1 is viewed as an edited version of the one in version 5.0.

Recall that our goal is to determine whether edits/errors are i) bursty or ii) uniformly/independently distributed. To rigorously answer this question, one way is to first define a uniform/independent random process for errors and edits and then use hypothesis testing to determine if such a model explains the observed data, which will be discussed in Subsection III-B. First, however, we perform an intuitive but less rigorous test over the alignment of pairs of sequences in Subsection III-A. Finally, in Subsection III-C, we consider choosing the model that leads to the lowest cost in error-correction and synchronization tasks for our data.

## *A. Independence test on alignment*

Let $\boldsymbol{z}$ be the erroneous/edited version of $\boldsymbol{x}$. An alignment between $\boldsymbol{x}$ and $\boldsymbol{z}$ identifies the positions where the sequences match and how they differ (see Figure 1). From the alignment, let us produce the binary sequence, denoted $\boldsymbol{a}$, in which 1 represents

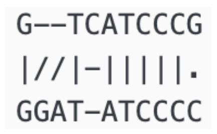


Figure 1: An alignment of two DNA sequences, where the top sequence can be obtained from the bottom one via deletions (/), insertions (−), and substitutions (·).

[1] It is a typical length of the synthetic DNA sequences in a current DNA storage system [18]

[2] https://ftp.gnu.org/gnu/bash/

a match and 0 represents an edit (substitution, insertion, deletion). If edits/errors are distributed in a uniform manner over $\boldsymbol{x}$, e.g., resulting from an "independent" process, then it is reasonable to expect the alignment $\boldsymbol{a}$ to resemble an iid sequence. Therefore, we apply the Wald-Wolfowitz runs-test [21].

*1) Wald-Wolfowitz runs-test:* Consider a binary sequence $\boldsymbol{a} = a_0 a_1 \cdots a_{N-1}$. The Wald-Wolfowitz runs test [21] tests the null hypothesis that $\boldsymbol{a}$ is iid using the number $R$ of runs as the test statistic. Conditioned on $t_1$ 1's and $t_0$ 0's, it can be shown that

$$\Pr(R = 2s | t_1, t_0) = \frac{2\binom{t_1-1}{s-1}\binom{t_0-1}{s-1}}{\binom{N}{t_1}}, \tag{3}$$

$$\Pr(R = 2s+1 | t_1, t_0) = \frac{\binom{t_1-1}{s-1}\binom{t_0-1}{s} + \binom{t_1-1}{s}\binom{t_0-1}{s-1}}{\binom{N}{t_1}}. \tag{4}$$

To see (3), note that $\binom{t_1-1}{s-1}$ (resp. $\binom{t_0-1}{s-1}$) is the number of compositions of $t_1$ (resp. $t_0$) into exactly $s$ parts, i.e., the number of ways $t_1$ 1's (resp. $t_0$ 0's) can be divided into $s$ runs. There is a factor of 2 as the sequence may start with a run of 1's or 0's and the denominator is the total number of sequences with $t_1$ 1's and $t_0$ 0's. Similarly, we have (4). Note that both very large and very small values of $R$ imply dependence between $a_i$'s, and thus suggestrejecting the null hypothesis. In all examples we tested, the actual number of runs $r$ was small. Therefore, for simplicity, we consider the one-sided test with $p$-value $\Pr(R \le r | t_1, t_0) = \sum_{y=1}^{r} \Pr(R = y | t_1, t_0)$.

Here, the null hypothesis is that $\boldsymbol{a}$ is iid generated and the number $R$ of runs in $\boldsymbol{a}$ is the test statistic. Conditioned on the number of 0's $t_0$ and the number of 1's $t_1$, the $p$-value is $\Pr(R \le r | t_0, t_1)$, as we do not expect to see very few runs in an iid sequence. As in "RUNS-TEST" column of Table I, this test strongly suggests that edits are not iid for both datasets.

## B. A probabilistic edit process

Another approach is to perform hypothesis testing on a probabilistic edit process. Again, let $\boldsymbol{x}$ be our data sequence of length $n$, and $\boldsymbol{z}$ its edited version. Based on an intuitive interpretation of "uniform edits", we define the following simple edit process: i) A random number $K_i$ of insertions are uniformly distributed over the $n+1$ "bins" between $x_i$ and $x_{i+1}$, $i = 0, \ldots, n$, where $x_0$ and $x_{n+1}$ are defined as the empty symbol. ii) A random number $K_d$ of substitutions and deletions are uniformly applied on $x_1, \ldots, x_n$.

The null hypothesis is that $\boldsymbol{z}$ is generated by this edit process. Since insertions occur independently of substitutions and deletions, we apply two separate tests. For insertions, we consider $W$, the number of runs of insertions, as the test statistic. $W$ has the same distribution as the number of nonempty bins when $k = K_i$ balls are distributed uniformly into $N = n+1$ distinct bins. We may assume that the balls are labeled as this does not affect the distribution of $W$. For $1 \le w \le \min(N, k)$, the probability of observing $w$ nonempty bins is

$$\Pr(W = w | k, N) = \frac{B_w^k \cdot w! \cdot \binom{N}{w}}{N^k}, \tag{5}$$

where $B_w^k$ is the number of partitions of a size-$k$ set into $w$ parts, also known as Stirling numbers of the second kind. To explain, there are $\binom{N}{w}$ ways of choosing the nonempty bins and $w! B_w^k$ ways of filling them with $k$ balls. For the denominator, there are $N^k$ choices as each ball can be placed in 1 of $N$ distinct bins. If most insertions cluster in a small number of bins, i.e., $W$ being small, then we reject the null hypothesis. The $p$-value $\Pr(W \le w | K_i)$ is summarized in "INS-TEST" column of Table I. Note that a new interpretation of the "INS-TEST" is discussed in Appendix A.

For substitutions and deletions, we consider again $R$, the number of runs in the edit pattern (excluding insertions) as the test statistic, and reject the null hypothesis if $R$ is small. The results are given in "SUBDEL-TEST" column of Table I. High rejection rates for both tests suggest that edits are not uniform.

## C. Operational evaluation of error/edit models

Given two sequences $\boldsymbol{x} \in \Sigma_q^n$ and $\boldsymbol{z} \in \Sigma_q^*$, their differences can be described via $b$ $k$-substring edits for a range of possible pairs $(b, k)$. Operationally, the best description, i.e., $(b, k)$ pair, is the one that leads to the lowest cost for the task at hand. For error-correction, where $\boldsymbol{z}$ is an erroneous copy of $\boldsymbol{x}$, the cost is the redundancy of the code that allows correcting the errors in $\boldsymbol{z}$. If $\boldsymbol{z}$ can be obtained from $\boldsymbol{x}$ via $b$ $k$-substring edits, based on the GV bound, there exists such a code of length $n$ with redundancy $\log r_n(b, k)$. For synchronization, where $\boldsymbol{x}$ is the edited version of $\boldsymbol{z}$ (or vice versa), the cost is the information exchange, i.e., the number of bits needed to be transmitted. It can be shown [20] that exchanging $\log r_n(b, k)$ bits is sufficient, achievable using a systematic code with $n$ information symbols and $\log_q r_n(b, k)$ check symbols.

## D. Experiment results

*1) Alignment:* Our experiment starts with computing the alignment of data sequence pairs in both datasets. We consider the 1000 input-output pairs in the nanopore sequencing dataset and the 589 pairs of common files (with lengths at most 20000 bytes and contain edits) in the bash dataset. We use the conventional dynamic programming approach [22] for computing the alignment. Table I presents the fraction of sequences rejecting the null hypothesis at $p$-value threshold of 5% for two datasets for three tests.

| | RUNS-TEST | INS-TEST | SUBDEL-TEST |
|---|---|---|---|
| Bash | 97.2% | 100% | 97.6% |
| DNA | 95.7% | 97.1% | 76.1% |

Table I: Fraction of sequences rejecting the null hypothesis at $p$-value threshold of 5%.

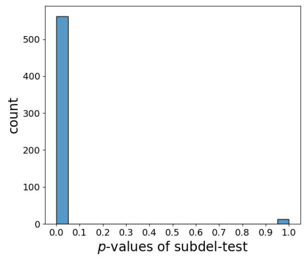
*2) Runs-test on alignment:* For each alignment, the runs-test is applied for determining if it resembles an iid sequence. In particular, the alignment is first converted into a binary edit pattern sequence with 1 representing a match and 0 one of the three types of edits. The $p$-value of the runs-test is computed according to (3) and (4), where $t_1, t_0, r$ are the numbers of 1's, 0's, and runs in the edit pattern, respectively.

In our experiment, the runs-test is applied on all 1000 input-output pairs in the nanopore sequencing dataset and 589 pairs of edited files in the bash dataset. Table I contains fraction of sequences rejecting the null hypothesis at $p$-value threshold of 5%.

*3) Testing the edit process:* The second part of our analysis focuses on the two testings for the proposed edit process. We assume that the alignment between two data sequences gives the actual edits that happened. Let $n$ be the length of the unedited data sequence.

- INS-TEST: For the insertions, we obtain the number of insertions $K_i$ and the number of runs of insertions $w$ from the alignment. The $p$-value is defined as the probability of seeing at most $w$ non-empty bins, i.e., $\Pr(W \leq w | k = K_i, N = n+1)$, where the pmf of $W$ is given by (5).
- SUBDEL-TEST: For the substitutions and deletions, we first remove the insertions from the alignment. Next, we convert the alignment to the binary edit pattern and count the number of 1's $K_d$ (corresponding to edits) and the number of 0's $n - K_d$. The test statistic is again $R$, the number of runs in the edit pattern. It is clear that under the assumed edit process, the distribution of $R$ given $K_d$ is given by (3) and (4) with $t_1 = K_d, t_0 = n - K_d$. The $p$-value is computed in the same way.

INS-TEST and SUBDEL-TEST are both applied on the two datasets, with fractions of rejections in Table I. Note that we only apply INS-TEST on data sequences that contain at least 5 insertions, and only apply SUBDEL-TEST on data sequences that contain at least 1 deletion or substitution. Apart from Table I, Figure 2 shows the histogram of the $p$-values for the "SUBDEL-TEST". Strong evidence for rejecting the uniform edit process can be observed.


(a) Bash dataset

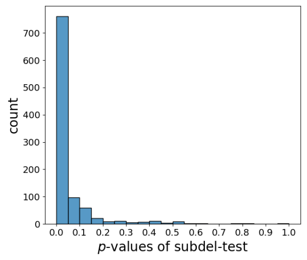

(b) Nanopore sequencing dataset

Figure 2: Histograms of $p$-values for SUBDEL-TEST.

*4) Operational evaluation of error/edit models:* For each pair of sequences in the genome data set, among all valid $(b, k)$ pairs, we find the one that minimizes $r_n(b, k)$, where $n = 200, q = 4$ for all DNA sequences.The histogram of the best $(b, k)$ pairs is given in Figure 3, which indicates that viewing the errors as 13 2-substring edits minimizes the redundancy for the largest number of input-output pairs.[3] This suggests that edits with $k > 1$ better describe our data and codes correcting $b$ $k$-substring edits for $k > 1$ are of use in DNA data storage. Note, however, that a priori we do not know the error that will occur and may have to over-provision to achieve reliability.

[3]We point out that only 272 sequence pairs are included as the rest are so erroneous that they have their minimum redundancy $\log r_n(b, k)$ larger than the original length (400 bits for $n = 200, q = 4$).

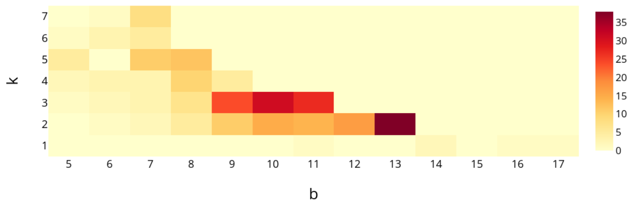


Figure 3: Histogram of optimal $(b, k)$ values for the Nanopore sequencing dataset.

Similarly, for each edited file in the bash dataset, we find the $(b, k)$ pair with the minimum redundancy, i.e., $\log r_n(b, k)$ with $n$ being the unedited file size and alphabet size $q = 256$, among all valid pairs. In our experiment, we let $k$ range from 1 to 10 and only consider files of lengths smaller than 3000 bytes (392 in total) to avoid large running time. The valid $(b, k)$ pairs are found by computing the smallest $b$ for each $k$ using a dynamic programming based algorithm. By only including files whose minimum redundancy is smaller than the original length ($\log r_n(b, k) < 8n$), we are left with 122 files. Figure 4 shows the histogram of the best $(b, k)$ pairs for these 122 files. It can be seen that for majority of the files studied, viewing edits as $k$-substring edits for $k > 1$ minimizes the redundancy. Interestingly, for $k = 10$, a group of pairs of files differ by $b = 66$ $k$-substring edits. A more careful analysis reveals that the files in the new version 5.1 included a claim of copyrights with length of roughly 660 bytes which is absent in the corresponding files in the old version 5.0. Both Figure 3 and Figure 4 represent that the editing of files is prone to bursty manners.

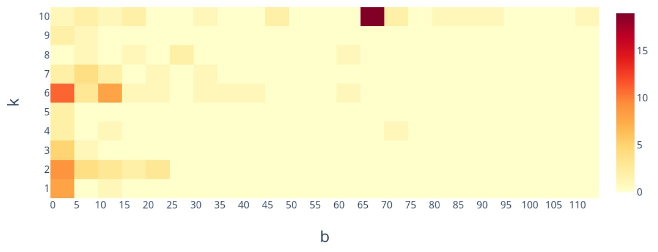


Figure 4: Histogram of optimal $(b, k)$ values for the bash dataset.

| | Bash dataset | | Nanopore sequencing dataset | |
|---|---|---|---|---|
| $k$ | $k \geq 1$ | $k = 1$ | $k \geq 1$ | $k = 1$ |
| average $\frac{\log_q r_n(b,k)}{n}$ | 0.30 | 0.62 | 0.84 | 1.06 |

Table II: Comparison of redundancy of burst-error correcting codes with $k \geq 1$ and independent indel-error-correcting codes with $k = 1$.

Based on Figure 3 and Figure 4, codes correcting burst errors achieve lower redundancy compared to codes correcting independent indels. To further support the results, Table II presents the average redundancy of indel-correcting codes

($k = 1$) and burst-correcting codes ($k \geq 1$) to correct a symbol over two datasets, respectively. In this experiment, suppose a dataset contains $N$ files with file size $\{n_1, \ldots, n_N\}$, we define the average redundancy of correcting the datasets as $\sum_{i=1}^{N} \log_q r_{n_i}(b_i, k_i) / \sum_{i=1}^{N} n_i$, where $(b_i, k_i)$ is the pair for the $i$th file. Based on the numerical results, for both datasets, the burst-error-correcting codes ($k \geq 1$) achieve a lower redundancy compared to the indels-error-correcting codes ($k = 1$).

## IV. Error-correcting codes for a $k$-substring edit

Given a constant $k$, this section focuses on constructing codes of length $N$ for correcting a $k$-substring edit error with redundancy asymptotically $2 \log N$ and polynomial time complexities in both the encoding and decoding processes. Unless otherwise stated, let $n$ represent the length of $(\mathcal{P}, \delta)$-dense strings and $\mathcal{P} = 0^k 1^k$.

Based on Lemma 4, given an input $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$, locating the burst of deletions relies on the number of patterns $n_{\mathcal{P}}(\boldsymbol{x})$ and the relative distances of every two adjacent patterns $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$. Compared with locating a burst of deletions, locating a $k$-substring edit is more complicated. Suppose $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$ is an output generated from $\boldsymbol{x}$ by a $k$-substring edit. We need to overcome the following challenges. First, when the $k$-substring edit is a burst of substitutions, i.e., $|\boldsymbol{x}| = |\boldsymbol{y}|$, it is possible for both $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ to remain the same, preventing us from locating the error. Second, even if the substring edit is strict, i.e., $|\boldsymbol{x}| \neq |\boldsymbol{y}|$, there is no guarantee that the changes affecting $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ will enable us to identify the approximate location of the error. To address these issues, this paper will extend the previous error-correcting code to correct a strict substring edit and adapt a low-redundancy code to correct a burst of substitutions, respectively.

In order to construct error-correcting codes reaching the GV bound, we have the following corollary, which also summarizes our approach.

**Corollary 5.** *The code $\mathcal{C} \subseteq \Sigma_2^N$ is capable of correcting a $k$-substring edit if it can correct either a $k$-burst substitution or a strict $k$-substring edit error.*

In subsections IV-A and IV-B, error-correcting codes to correct a strict $k$-substring edit and a $k$-burst substitution are presented, respectively, followed by the final code construction in Subsection IV-C and the analysis of the time complexities of encoding/decoding in Subsection IV-D.

### A. Error-correcting code for a strict $k$-substring edit

Similar to works in [8], [9], given a constant $k$, this subsection focuses on correcting a strict $k$-substring edit by first localizing the error and then correcting it in the interval. More specifically, it consists of two steps. First, we extend the error-locating code in Lemma 4 from [9] to locate the strict $k$-substring edit in an interval of length $L = O((\log n)^2)$ with redundancy asymptotically $\log n$. Second, a modified *syndrome compression code* [11], [16] is designed to correct the error in the specific interval with redundancy asymptotically $O(\log \log n)$.

*1) Locating the error in an interval:* This subsection focuses on extending the codes in Lemma 4 to locate a strict $k$-substring edit since it will affect at least one of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$.

**Lemma 6.** *Given $\mathcal{P} = 0^k 1^k$ and $\delta = k 2^{2k+1} \lceil \log n \rceil$, let $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n$ be a $(\mathcal{P}, \delta)$-dense string and $\boldsymbol{y} \in D(\boldsymbol{x})$. Then a $k$-substring edit does not create nor destroy more than two* adjacent *patterns $\mathcal{P}$ in $\boldsymbol{x}$, i.e., $n_{\mathcal{P}}(\boldsymbol{y}) \in [n_{\mathcal{P}}(\boldsymbol{x}) - 2, n_{\mathcal{P}}(\boldsymbol{x}) + 2]$.*

*Proof:* Let $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{v}\boldsymbol{w}$ and $\boldsymbol{y} = \boldsymbol{u}\boldsymbol{v}'\boldsymbol{w}$. Observe that

$$n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) \overset{\text{(a)}}{\leq} n_{\mathcal{P}}(\boldsymbol{x}) \overset{\text{(b)}}{\leq} n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) + 2,$$
$$n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) \overset{\text{(c)}}{\leq} n_{\mathcal{P}}(\boldsymbol{y}) \overset{\text{(d)}}{\leq} n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) + 2,$$

where the upper bounds hold because $|\boldsymbol{v}|, |\boldsymbol{v}'| \leq k$ and the length of the pattern is $2k$. The upper bound on $n_{\mathcal{P}}(\boldsymbol{y})$ follows from (d) and (a), and the lower bound from (c) and (b). ■

Then we have the following corollary.

**Corollary 7.** *Let $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$. Then $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ can be generated from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ as a result of a 3-substring-edit.*

According to the changes of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$, we extend the code in Bitar et al. work [9] as the following construction that helps to locate a strict $k$-substring edit occurring in the $(\mathcal{P}, \delta)$-dense string for $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$. Compared to localized deletions only reducing the length, the following modification can deal with the case of increasing the length and locating a strict $k$-substring edit in a wider interval.

**Construction 8.** *Given $0 \leq c_1 \leq 4$ and $0 \leq c_2 < 7n$, let*

$$\mathcal{C}_{loc}(c_1, c_2) = \{\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) \equiv c_1 \bmod 5,$$
$$VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) \equiv c_2 \bmod 7n\}.$$

Then we have the following lemma.

**Lemma 9.** *Let $k$ be a constant. Given $\boldsymbol{x} \in \mathcal{C}_{loc}(c_1, c_2)$, a strict $k$-substring edit occurring in $\boldsymbol{x}$ can be located in a substring of $\boldsymbol{x}$ with length $O(\delta^2) = O((\log n)^2)$.*

*Proof:* The lemma is proved by adapting a similar method from the work [9]. Suppose each $\boldsymbol{x} \in \mathcal{C}_{loc}(c_1, c_2)$ is partitioned into substrings by the pattern $\mathcal{P} = 0^k 1^k$. To locate the strict $k$-substring edit, one approach is to find the substrings that are affected by the $k$-substring edit. In this proof, we construct a monotonic function with respect to the index of the first substring affected by the strict $k$-substring edit (given in (8)). This will enable us to find a range of the indices, leading to an interval in which the strict $k$-substring edit occurs.

Consider $\boldsymbol{x} \in \mathcal{C}_{loc}(c_1, c_2)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$. We get $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ and $n_{\mathcal{P}}(\boldsymbol{x})$ from $\boldsymbol{x}$ and similarly $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ and $n_{\mathcal{P}}(\boldsymbol{y})$ from $\boldsymbol{y}$. Since $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$, we have $2k \leq a_{\mathcal{P}}(\boldsymbol{x})_i \leq \delta$ for $i \geq 2$.

In the following, we analyze the changes from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ to $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$. Based on Corollary 7, the vector $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ can be generated from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ by replacing a substring $\boldsymbol{u}$ in $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$

with another substring $\boldsymbol{v}$ in $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$, i.e.,

$$\begin{aligned}\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x}) &= (a_{\mathcal{P}}(\boldsymbol{x})_{[1,t)}, \boldsymbol{u}, a_{\mathcal{P}}(\boldsymbol{x})_{[e_1,n_{\mathcal{P}}(\boldsymbol{x})]}) \\ &= (a_{\mathcal{P}}(\boldsymbol{y})_{[1,t)}, \boldsymbol{u}, a_{\mathcal{P}}(\boldsymbol{y})_{[e_2,n_{\mathcal{P}}(\boldsymbol{y})]}), \\ \boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y}) &= (a_{\mathcal{P}}(\boldsymbol{x})_{[1,t)}, \boldsymbol{v}, a_{\mathcal{P}}(\boldsymbol{x})_{[e_1,n_{\mathcal{P}}(\boldsymbol{x})]}) \\ &= (a_{\mathcal{P}}(\boldsymbol{y})_{[1,t)}, \boldsymbol{v}, a_{\mathcal{P}}(\boldsymbol{y})_{[e_2,n_{\mathcal{P}}(\boldsymbol{y})]}).\end{aligned} \tag{6}$$

where $0 \le |\boldsymbol{u}|, |\boldsymbol{v}| \le 3$, $t \le e_1, e_2 \le t+3$. Note that $a_{\mathcal{P}}(\boldsymbol{y})_i$ satisfies $2k \le a_{\mathcal{P}}(\boldsymbol{y})_i \le 3\delta+k < 4\delta$, where the lower bound is obtained since $\mathcal{P}$ has length $2k$ for $i \ge 2$. Furthermore, based on Lemma 6, we have $|n_{\mathcal{P}}(\boldsymbol{y}) - n_{\mathcal{P}}(\boldsymbol{x})| \le 2$ and $||\boldsymbol{u}| - |\boldsymbol{v}|| \le 2$. Therefore, the value of $a_{\mathcal{P}}(\boldsymbol{y})_i$ is upper bounded by $3\delta + k$ after breaking two patterns and inserting a substring of length upper bounded by $k$.

Let $d = |\boldsymbol{u}| - |\boldsymbol{v}|$. Then $d \in [-2, 2]$ can be uniquely computed from $\boldsymbol{y}$, i.e., $d \equiv n_{\mathcal{P}}(\boldsymbol{y}) - c_1 \bmod 5$. Let $k' = |\boldsymbol{x}| - |\boldsymbol{y}|$, then we have $k' = \sum_{i=t}^{e_1-1} a_{\mathcal{P}}(\boldsymbol{x})_i - \sum_{i=t}^{e_2-1} a_{\mathcal{P}}(\boldsymbol{y})_i$, where $0 < |k'| \le k$. Note that $k' \neq 0$ for a strict $k$-substring edit.

Then we have

$$\begin{aligned} Z :&= VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})) - VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) \\ &= d \sum_{i=e_2}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - t \left( \sum_{i=t}^{e_1-1} a_{\mathcal{P}}(\boldsymbol{x})_i - \sum_{i=t}^{e_2-1} a_{\mathcal{P}}(\boldsymbol{y})_i \right) \\ &\quad - \sum_{i=t+1}^{e_1-1} (i-t) a_{\mathcal{P}}(\boldsymbol{x})_i + \sum_{i=t+1}^{e_2-1} (i-t) a_{\mathcal{P}}(\boldsymbol{y})_i \\ &= d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk' + E, \end{aligned} \tag{7}$$

where $0 < |k'| \le k$ and $E = d \sum_{i=e_2}^{t+2} a_{\mathcal{P}}(\boldsymbol{y})_i - \sum_{i=t+1}^{e_1-1} (i-t) a_{\mathcal{P}}(\boldsymbol{x})_i + \sum_{i=t+1}^{e_2-1} (i-t) a_{\mathcal{P}}(\boldsymbol{y})_i$. Since $e_2 \le t+3$, the total components of the first and the last summation are at most three each with coefficients $\max(|d|, e_2 - t - 1) \le 2$. Then we have $|E| \le 3 \cdot 2 \cdot \max(a_{\mathcal{P}}(\boldsymbol{y})_i) < 24\delta$. Then the difference of VT check sums satisfies

$$Z = d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk' + E$$

where $0 < |k'| \le k$ results from a strict $k$-substring edit compared to $k \ge k' > 0$ for a burst deletion. Because $|d| \le 2$ and $|tk'| \le n$, we have $|d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk'| \le 3n$. This, along with the fact that $|E| < 24\delta$, implies $-3n - 24\delta < Z < 3n + 24\delta$. Since $7n > 6n + 48\delta + 1$ and $VT(a_{\mathcal{P}}(\boldsymbol{x})) = c_2 \bmod 7n$, the integer $Z \in [-3n - 24\delta + 1, 3n + 24\delta - 1]$ can be uniquely obtained based on $Z = VT(a_{\mathcal{P}}(\boldsymbol{y})) - c_2 \bmod 7n$.

In the following, we define a function

$$H(t) := d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk'. \tag{8}$$

Our task is to prove that $H(t)$ is a strictly monotonic function with respect to $t$ for $0 < |k'| \le k$, not only $k \ge k' > 0$:

- Firstly, suppose $0 < k' \le k$. If $d \ge 0$, $H(t)$ is a strictly decreasing function of $t$. If $d < 0$, then by the function $a_{\mathcal{P}}(\boldsymbol{y})_i \ge 2k > k'$, then $H(t)$ is a strictly increasing function of $t$.
- Secondly, suppose $-k \le k' < 0$. If $d \le 0$, $H(t)$ is a strictly increasing function of $t$. If $d > 0$, since $a_{\mathcal{P}}(\boldsymbol{y})_i \ge 2k > -k'$, $H(t)$ is a strictly decreasing function of $t$.

Given $Z$, $k'$, and $d$, we can locate $t$ in the set

$$I = \{t : H(t) \in [Z - 24\delta, Z + 24\delta]\}.$$

For a monotonic function $H(t)$, there are at most $48\delta + 1$ choices of $t$. Since $a_{\mathcal{P}}(\boldsymbol{y})_i < 4\delta$, we can locate the strict $k$-substring edit in an interval of length at most $\mathcal{L} = 192\delta^2 + 4\delta = O(\delta^2) = O((\log n)^2)$. The time complexity to compute $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$, $n_{\mathcal{P}}(\boldsymbol{y})$, and the interval is $O(n)$. ■

*2) Correcting the error in an interval:* Assuming the strict $k$-substring edit is located in an interval of length $O(\delta^2)$, we present error-correcting codes that can correct this error.

Similar to [11], we generate two sets of blocks of length $2L$ by partitioning $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$. More specifically, given $\hat{N} = n/2L$, let $S_e = (\boldsymbol{s}_{e1}, \boldsymbol{s}_{e2}, \ldots, \boldsymbol{s}_{e\hat{N}})$ and $S_o = (\boldsymbol{s}_{o1}, \boldsymbol{s}_{o2}, \ldots, \boldsymbol{s}_{o(\hat{N}-1)})$ denote the set of even and odd blocks respectively, where $\boldsymbol{s}_{ei} = \boldsymbol{x}_{[2(i-1)L+1, 2iL]}$ for $i \in [\hat{N}]$ and $\boldsymbol{s}_{oi} = \boldsymbol{x}_{[(2i-1)L+1, (2i+1)L]}$ for $i \in [\hat{N} - 1]$.

If the $k$-substring edit is located in a specific interval with the length upper bounded by $L$, then it will only affect one length-$2L$ block of either $S_e$ and $S_o$, explicitly shown in the proof of Lemmea 12. In the following, we apply a modified syndrome compression code [16] to correct a strict $k$-substring edit in a length-$2L$ string.

Based on Theorem 3, for a labeling function $f$ and a stored sequence $\boldsymbol{u} \in \Sigma_2^{2L}$, the decoder can recover $\boldsymbol{u}$ from the retrieved word $\boldsymbol{v} \in D(\boldsymbol{u})$, the integer $a$, and $(f(\boldsymbol{u}) \bmod a)$. Observe that the number of bits required to represent $f(\boldsymbol{u})$ does not affect the redundancy since the redundancy needed to convey $a$ and $(f(\boldsymbol{u}) \bmod a)$ is $2 \log a$. According to Theorem 3, this latter quantity is affected by the size of $B(\boldsymbol{u})$. Since a strict $k$-substring edit can be viewed as a $k$-burst deletion followed by a $k$-burst insertion, we introduce a labeling function that can correct at most $2k$ insertions, deletions, and substitutions. The following theorem introduces such a function from [14].

**Theorem 10** (cf. [14]). *Given a constant $k$, $t = 2k$, and $L = O((\log n)^2)$, there exists a labeling function $g : \Sigma_2^{2L} \to \Sigma_{2^{\mathcal{R}(t,2L)}}$ such that for any two distinct strings $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ confusable under at most $t$ insertions, deletions, and substitutions, we have $g(\boldsymbol{s}_1) \neq g(\boldsymbol{s}_2)$, where $\mathcal{R}(t, 2L) = ((t^2+1)(2t^2+1) + 2t^2(t-1)) \log 2L + o(\log 2L) = O(\log\log n) + o(\log\log n)$.*

Based on Lemma 2, given $\boldsymbol{u} \in \Sigma_2^{2L}$, the size of the confusable set $B(\boldsymbol{u})$ satisfies $\|B(\boldsymbol{u})\| < (2L+k)^2(k+1)^4 2^{2k}$. For $i \in [\hat{N}]$ and each $\boldsymbol{s}_{ei} \in \Sigma_2^{2L}$, there exists an integer $a_{ei}$ (which depends on $\|B(\boldsymbol{s}_{ei})\|$) such that for every $\boldsymbol{w}_{ei} \in B(\boldsymbol{s}_{ei})$, $g(\boldsymbol{s}_{ei}) \not\equiv g(\boldsymbol{w}_{ei}) \bmod a_{ei}$, where $a_{ei} \le 2^{\log \|B(\boldsymbol{s}_{ei})\| + o(\log L)}$. The same property holds for each $\boldsymbol{s}_{oi} \in \Sigma_2^{2L}$ for $i \in [\hat{N} - 1]$.

Based on the two sets of messages $S_e, S_o$, we have the following construction for a $k$-substring edit.

**Construction 11.** *Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_6)$. Given $\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2)$ with length $n$, we form two sets of message blocks $S_e$ and $S_o$. Let $\beta_3, \ldots, \beta_6 < \alpha$. Then we have*

$$\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha) = \{\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2),$$

$$\sum_{i=1}^{\hat{N}} a_{ei} \equiv \beta_3 \bmod \alpha, \sum_{i=1}^{\hat{N}} (g(\boldsymbol{s}_{ei}) \bmod a_{ei}) \equiv \beta_4 \bmod \alpha,$$
$$\sum_{i=1}^{\hat{N}-1} a_{oi} \equiv \beta_5 \bmod \alpha, \sum_{i=1}^{\hat{N}-1} (g(\boldsymbol{s}_{oi}) \bmod a_{oi}) \equiv \beta_6 \bmod \alpha.\}$$

*where* $\alpha \geq (2L+k)^2(k+1)^4 2^{2k} 2^{o(\log L)} > \max(a_{e1}, a_{o1}, \ldots, a_{e(\hat{N}-1)}, a_{o(\hat{N}-1)}, a_{eN})$*,* $0 \leq \beta_1 \leq 4$*,* $0 \leq \beta_2 \leq 7n$*, and* $0 \leq \beta_3, \beta_4, \beta_5, \beta_6 < \alpha$*.*

Note that a similar construction appeared recently in [11], [12] for burst deletions. However, our construction includes a more powerful error-locating code and modified modulus so that our code can correct a strict $k$-substring edit.

**Lemma 12.** *Given a constant* $k$*, the error-correcting code* $\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ *in Construction 11 can correct a strict* $k$*-substring edit error with redundancy* $\log n + 16 \log\log n + o(\log\log n)$ *bits.*

*Proof:* Let $\boldsymbol{x} \in \mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ and $\boldsymbol{z} \in D(\boldsymbol{x})$ with $k' = |\boldsymbol{z}| - |\boldsymbol{x}| \neq 0$. Let $\mathcal{L}$ denote the length of the interval in which the strict $k$-substring is located, as given at the end of the proof of Lemma 9. Specifically, based on Construction 8 and Lemma 9, we can find $\hat{j} \in [n - \mathcal{L}]$ such that the strict $k$-substring edit is located in $\boldsymbol{z}_{[\hat{j}, \hat{j}+\mathcal{L}-1]}$. We choose $L$ to satisfy $L > \mathcal{L} + k$. Let $\hat{n} = \lfloor (\hat{j} - 1)/L \rfloor$. Then, $(\hat{n}+1)L + 1 > \hat{j}$ and thus $(\hat{n}+2)L + k' + 1 > \hat{j} + \mathcal{L}$. Therefore, $\boldsymbol{z}_{[\hat{n}L]} = \boldsymbol{x}_{[\hat{n}L]}$ and $\boldsymbol{z}_{[(\hat{n}+2)L+k'+1, n+k']} = \boldsymbol{x}_{[(\hat{n}+2)L+1, n]}$. If $\hat{n}$ is even, we will aim to reconstruct $S_e$ and if it is odd, $S_o$. Suppose $\hat{n}$ is even and let $S'_e$ be the reconstruction of $S_e$. From $\boldsymbol{z}_{[\hat{n}L]}$ and $\boldsymbol{z}_{[(\hat{n}+2)L+k'+1, n+k']}$, we can reconstruct all blocks of $S'_e$ error-free, except $\boldsymbol{s}'_{ei}, i = \hat{n}/2 + 1$, for which we set $\boldsymbol{s}'_{ei} = \boldsymbol{z}_{[\hat{n}L+1, (\hat{n}+2)L+k']}$. Then this block $\boldsymbol{s}'_{ei}$ contains the whole interval of length $\mathcal{L}$ where the strict $k$-substring edit occurs. Thus $\boldsymbol{s}'_{ei} \in D(\boldsymbol{s}_{ei})$ and the other blocks in $S'_e$ are error-free. For all error-free blocks $\boldsymbol{s}'_{ej}$ with $j \neq i$, we can recover $a_{ej}$ and $g(\boldsymbol{s}_{ej})$. We can uniquely recover $a_{ei}$ and $g(\boldsymbol{s}_{ei}) \bmod a_{ei}$, followed by recovering $\boldsymbol{s}_{ei}$ based on $\boldsymbol{s}'_{ei}$, $a_{ei}$, and $g(\boldsymbol{s}_{ei}) \bmod a_{ei}$. Hence, we can recover $\boldsymbol{x}$ from $S'_e$. The case for odd $\hat{n}$ is similar.

According to (1), there are at least $2^{n-1}$ $(\mathcal{P}, \delta)$-dense strings, which are partitioned in Constructions 8 and 11 into $5 \cdot 7n \cdot \alpha^4$ sets. Hence, there exists some code $\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ such that $\|\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)\| \geq \frac{2^{n-1}}{5 \cdot 7n \cdot \alpha^4}$. Let $\alpha = (2L+k)^2(k+1)^4 2^{2k} 2^{o(\log L)}$ with $L = O((\log n)^2)$. The redundancy is $n - \log \|\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)\| \leq 1 + 4\log\alpha + \log 7n + \log 5$
$= \log n + 16 \log\log n + o(\log\log n)$. ■

## B. Error-correcting codes for a $k$-burst substitution

In this subsection, we present a code that can correct a $k$-burst substitution error.

**Construction 13** (cf. [23], Fire code)**.** *Let* $g_0(x)$ *be an irreducible polynomial of degree* $m \geq k$ *that does not divide* $x^{2k-1} - 1$*. Then, there exists a linear cyclic code, called the* Fire code*, of length* $n_1 = \mathrm{LCM}(2k-1, 2^m - 1)$ *with the* generator polynomial $g(x) = (x^{2k-1} - 1)g_0(x)$ *and* $\deg(g(x)) = m + 2k - 1$*. The Fire code* $\mathcal{C}_F$ *is an* $[n_1, n]$ *code with code length* $n_1$ *and dimension* $n = n_1 - (m + 2k - 1)$*.*

**Theorem 14** (cf. [23])**.** *The Fire code* $\mathcal{C}_F$ *can correct a single* $k$*-burst substitution.*

The following lemma gives the redundancy of $\mathcal{C}_F$.

**Lemma 15.** *Given a constant* $k$*, the Fire code* $\mathcal{C}_F$ *corrects a* $k$*-burst substitution with the redundancy asymptotically* $\log n + 2k + o(\log n)$*.*

*Proof:* Based on Construction 13, the redundancy of the Fire code is

$$r(\mathcal{C}_F) = n_1 - \log_2 \|\mathcal{C}_F\| = m + 2k - 1.$$

Given $\boldsymbol{x} \in \Sigma_2^n$, $m \geq k$, and $n_1 = \mathrm{LCM}(2k-1, 2^m - 1)$, the length of Fire code satisfies

$$2^m - 1 \leq n_1 = n + m + 2k - 1 \leq (2k-1)(2^m - 1).$$

Hence, we have $m = \log n_1 + o(\log n_1) = \log n + o(\log n) > k$ as $n \to \infty$. Thus the redundancy of the Fire code is asymptotically $\log n + 2k + o(\log n)$ when $k$ is a constant. ■

Hence, given $\boldsymbol{x} \in \Sigma_2^n$, there exists a function $h_F(\boldsymbol{x})$ of length asymptotically $\log n + 2k + o(\log n)$ such that $(\boldsymbol{x}, h_F(\boldsymbol{x})) \in \mathcal{C}_F$ can correct a $k$-burst substitution.

## C. Combined error-correcting codes

Based on Corollary 5, given $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n$, the receiver can correct a $k$-substring edit from $\boldsymbol{y} \in D(\boldsymbol{x})$ if $h_F(\boldsymbol{x})$ and $(\boldsymbol{\beta}, \alpha)$ are sent to the receiver by an error-free channel. For simplicity, let $\boldsymbol{r_x} := (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ be a binary representation of the data. Then each codeword of the final error-correcting codes can be generated by concatenating two parts, i.e., $(\boldsymbol{x}, \boldsymbol{r_x})$. Furthermore, we may add a buffer between $\boldsymbol{x}$ and $\boldsymbol{r_x}$ such that a $k$-substring edit affects either $\boldsymbol{x}$ or $\boldsymbol{r_x}$. Finally, We also need another function that can detect or correct a $k$-substring edit occurring in $\boldsymbol{r_x}$. We start by finding a suitable buffer $\boldsymbol{b_x}$.

Since a $k$-substring edit should not affect both $\boldsymbol{x}$ and $\boldsymbol{r_x}$, the length of the buffer satisfies $|\boldsymbol{b_x}| > k$. The buffer in the following lemma helps distinguish whether the strict $k$-substring edit affects $\boldsymbol{x}$ or $\boldsymbol{r_x}$.

**Lemma 16.** *Let* $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b_x}\boldsymbol{u}$ *with* $\boldsymbol{x} \in \Sigma_2^n$*,* $\boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$*, and* $\boldsymbol{u} \in \Sigma_2^*$*. Let* $\boldsymbol{z}$ *be obtained from* $\boldsymbol{w}$ *through a strict* $k$*-substring edit. There exists a decoder that given* $\boldsymbol{z}$ *returns either* $\boldsymbol{x}$ *or an element of* $D(\boldsymbol{x})$ *along with* $\boldsymbol{u}$*.*

*Proof:* Suppose the edit replaces some substring $\boldsymbol{w}_{[i+1, j-1]}$ of $\boldsymbol{w}$ with a string $\boldsymbol{t}$, where $j - i - 1 \leq k, |\boldsymbol{t}| \leq k, j - i - 1 \neq |\boldsymbol{t}|$. (If $j = i + 1$, the edit deletes nothing and inserts $\boldsymbol{t}$.) Let $k' = |\boldsymbol{w}| - |\boldsymbol{z}| = (j - i - 1) - |\boldsymbol{t}|$. Note that for all $m \leq i$ and all $\ell \geq j$, we have $w_m = z_m$ and $w_\ell = z_{\ell - k'}$.

Recall that $\boldsymbol{w}_{[n+1, n+3k+3]} = \boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$. Let $m^*$ be the smallest $m \in [n+1, n+3k+3]$ such that $w_m \neq z_m$ and $\ell^*$ the largest $\ell \in [n+1, n+3k+3]$ such that $w_\ell \neq z_{\ell-k'}$. Because $0 < |k'| \leq k$ and due to the structure of $\boldsymbol{b_x}$, at least one of $m^*$ and $\ell^*$ exists. Conditioned on existence, $i < m^*, j > \ell^*$.

Suppose $m^*$ exists and $m^* \leq n+2k+4$. Since $i < m^*$, $j \leq i+k+1 \leq m^*+k \leq n+3k+4$. Thus $\boldsymbol{u}$ is not affected by the edit and $\boldsymbol{z}_{[|\boldsymbol{z}|-|\boldsymbol{u}|+1,|\boldsymbol{z}|]} = \boldsymbol{u}, \boldsymbol{z}_{[1,n-k']} \in D(\boldsymbol{x})$.

Next, suppose $m^*$ does not exist or $m^* > n+2k+4$. Then $z_{n+2k+2} = w_{n+2k+2} = 0$ and $z_{n+2k+3} = w_{n+2k+3} = 1$. We consider the cases of $k' > 0$ and $k' < 0$. Suppose first $k' > 0$. Then $n+2k+k'+2 \in [n+2k+3, n+3k+2]$ and thus $w_{n+2k+k'+2} = 1$. Hence, $0 = z_{n+2k+2} \neq w_{n+2k+k'+2} = 1$, implying that $\ell^*$ exists and $\ell^* \geq n+2k+k'+2 \geq n+2k+3$. Second, suppose $k' < 0$. Then $n+2k+k'+3 \in [n+k+3, n+2k+2]$ and thus $w_{n+2k+k'+3} = 0$. Hence, $1 = z_{n+2k+3} \neq w_{n+2k+k'+3} = 0$, implying that $\ell^*$ exists and $\ell^* \geq n+2k+k'+3 \geq n+k+3$. Thus, regardless of the sign of $k'$, we have $\ell^* \geq n+k+3$. It follows that $i \geq j-k-1 \geq \ell^*-k \geq n+3$. Hence, $\boldsymbol{x}$ is not affected by the edit and $\boldsymbol{x} = \boldsymbol{z}_{[1,n]}$.

Based on the preceding discussion, the decoder computes $m^*$. If $m^*$ exists and $m^* \leq n+2k+4$, then it returns $\boldsymbol{z}_{[|\boldsymbol{z}|-|\boldsymbol{u}|+1,|\boldsymbol{z}|]} = \boldsymbol{u}$ and $\boldsymbol{z}_{[1,n-k']} \in D(\boldsymbol{x})$. Else, it returns $\boldsymbol{z}_{[1,n]} = \boldsymbol{x}$. ■

**Lemma 17.** *Given a constant $k$ and $\boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$, a burst-substitution-detecting function $\mathcal{E}_1$ in $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$ is sufficient to decode $\boldsymbol{x}$ for a $k$-substring edit.*

*Proof:* Recall that since $|\boldsymbol{b_x}| > k$, a $k$-substring edit will not affect $\boldsymbol{x}$ and $\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$ simultaneously. We present how a burst-substitution-detecting function is sufficient to decode $\boldsymbol{x}$. Given $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$, let $\boldsymbol{z} = D(\boldsymbol{w})$. Furthermore, let $k' = |\boldsymbol{w}| - |\boldsymbol{z}|$. Then we decode $\boldsymbol{x}$ in the following process.

- Suppose $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$ suffers a strict $k$-substring edit and $k' \neq 0$. Based on Lemma 16, we can distinguish whether $\boldsymbol{x}$ is affected. If $\boldsymbol{x}$ is error-free, we are done. If $\boldsymbol{x}$ is affected by a strict $k$-substring edit, then $\boldsymbol{r_x}$ is error-free. Since $\boldsymbol{z}_{[1,n-k']} \in D(\boldsymbol{x})$, then $\boldsymbol{x}$ can be recovered from $\boldsymbol{z}_{[1,n-k']}$ and $(\boldsymbol{\beta}, \alpha)$ in $\boldsymbol{r_x}$.
- Suppose $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$ suffers a $k$-burst substitution and $k' = 0$. Then we decode $\boldsymbol{x}$ in following cases:
  - If $\boldsymbol{z}_{[n+1,n+3k+3]} = 1^{k+1}0^{k+1}1^{k+1}$, the burst substitution affects either $\boldsymbol{x}$ or $\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$. If the burst-substitution-detecting code $\mathcal{E}_1$ does not detect an error in $\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$, then $\boldsymbol{x}$ is affected. Then we can decode $\boldsymbol{x}$ from $\boldsymbol{z}_{[1,n]} \in D(\boldsymbol{x})$ and $h_F(\boldsymbol{x})$ in $\boldsymbol{r_x}$. If an error is detected in $\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$, $\boldsymbol{x} = \boldsymbol{z}_{[1,n]}$ is error-free.
  - If $\boldsymbol{z}_{[n+1,n+3k+3]} \neq 1^{k+1}0^{k+1}1^{k+1}$ but $\boldsymbol{z}_{[n+1,n+2k+2]} = 1^{k+1}0^{k+1}$, $\boldsymbol{x}$ is error-free.
  - If $\boldsymbol{z}_{[n+1,n+3k+3]} \neq 1^{k+1}0^{k+1}1^{k+1}$ but $\boldsymbol{z}_{[n+k+2,n+3k+3]} = 0^{k+1}1^{k+1}$, then $\boldsymbol{z}_{[1,n]} \in D(\boldsymbol{x})$ can be generated from $\boldsymbol{x}$ by a $k$-burst substitution. Thus, we can decode $\boldsymbol{x}$ from $\boldsymbol{z}_{[1,n]} \in D(\boldsymbol{x})$ and $h_F(\boldsymbol{x})$.
  - Otherwise, the $k$-substring edit only affect the buffer, $\boldsymbol{x}$ is error-free. ■

The simplest burst-substitution-detecting function is a parity-checking function. Given $\boldsymbol{r_x}$, let $L_1 = |\boldsymbol{r_x}| > k$. Then we partition $\boldsymbol{r_x}$ into $T = \lceil L_1/(k+1) \rceil$ blocks of length $(k+1)$, i.e., $\boldsymbol{w}_j$ for $j \in [T]$, where additional 0s are appended if the last block has less than $(k+1)$ binary symbols. Then the error-detecting function $\mathcal{E}_1 : \Sigma_2^{L_1} \to \Sigma_2^{2k+2}$ appends $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ in the binary form (each with length $(k+1)$) following $\boldsymbol{r_x}$, where $\boldsymbol{\gamma}_1 = \left(\sum_{j=1}^{\lceil T/2 \rceil} \boldsymbol{w}_{2j-1}\right) \bmod 2^{k+1}, \boldsymbol{\gamma}_2 = \left(\sum_{j=1}^{\lfloor T/2 \rfloor} \boldsymbol{w}_{2j}\right) \bmod 2^{k+1}$.

The final construction is shown below.

**Construction 18.** *Given a constant $k$, $\boldsymbol{b}_x = 1^{k+1}0^{k+1}1^{k+1}$, we have a construction $\mathcal{C}_N$ as*

$$\mathcal{C}_N = \{\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N, \boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n)\},$$

*where $\boldsymbol{r_x} = (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ and $\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$ are in binary form generated by $\mathcal{E}_1(\boldsymbol{r_x})$ in each codeword $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N$.*

**Theorem19.** *The error-correcting code $\mathcal{C}_N$ in Construction 18 can correct a $k$-substring edit with the redundancy of asymptotically $2\log N + o(\log N)$ bits, where $N = n + 2\log n + o(\log n)$.*

*Proof:* We first prove that the error-correcting code $C_N$ can correct a $k$-substring edit. Given $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in C_N$ and $\boldsymbol{z} \in D(\boldsymbol{w})$. Let $k' = |\boldsymbol{w}| - |\boldsymbol{z}|$.

- If $k' \neq 0$, the buffer $\boldsymbol{b_x}$ helps distinguish whether the strict $k$-substring edit affects $\boldsymbol{x}$ by Lemma 16. If $\boldsymbol{x}$ is error-free, we are done. If $\boldsymbol{x}$ suffers a substring edit, then $\boldsymbol{r_x}$ is error-free and $\boldsymbol{x}$ can be recovered from $\boldsymbol{z}_{[1,n-k']} \in D(\boldsymbol{x})$ and $(\boldsymbol{\beta}, \alpha)$.
- If $k' = 0$. Then $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ help recover $\boldsymbol{x}$ based on Lemma 17.

Next, we discuss the redundancy of the code $\mathcal{C}_N$.

Based on (1), the size of the error-correcting code in Construction 18 is $||\mathcal{C}_N|| \geq 2^{n-1}$. Furthermore, given a codeword $\boldsymbol{c} = \boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$, the lengths of $\boldsymbol{x}$, $\boldsymbol{b_x}$, $\boldsymbol{r_x}$, and $\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$ are $n$, $3k+3$, $2\log n + 16\log\log n + 2k + o(\log n)$, and $2k+2$, respectively. Therefore, the total length of $\boldsymbol{c}$ is $N = n + 2\log n + 16\log\log n + 7k + o(\log n)$. Then the redundancy of the code in Construction 18 is asymptotically $N - \log||\mathcal{C}|| = 2\log n + 16\log\log n + 7k + o(\log n) = 2\log N + o(\log N)$, where $\log n = \log N + o(\log N)$. ■

### D. Time complexity

This subsection presented the polynomial time complexities in both encoding and decoding algorithms.

We start with the time complexity of the encoder. Given a constant $k$, each codeword is generated by four steps:

- First, given $\boldsymbol{u} \in \Sigma_2^*$, an $(\mathcal{P}, \delta)$-dense string $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ can be generated by Algorithm 2 from Wang et.al [12]. The time complexity is polynomial with respect to $n$.
- Second, append a buffer $\boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$ with time complexity $O(1)$ .
- Third, produce $\boldsymbol{r_x}$ to correct a $k$-burst edit by applying the encoders of the linear Fire code [23], the error-locating code in Lemma 9, and the modified syndrome compression codes. For a constant $k$, the encoders of Fire code [23] and the error-locating code have polynomial time complexity with respect to $n$. Furthermore, based on [14], [16], for a constant $k$, the time complexity of the modified syndrome compression is also polynomial with respect to $n$.

- Forth, append two parity blocks $\gamma_1$ and $\gamma_2$ with time complexity $O(n)$.

Therefore, the time complexity of the encoder is polynomial with respect to $n$.

The decoder consists of detecting the parity check bits, decoding the Fire code [23], locating the error in an interval, and decoding the modified syndrome compression codes. Similarly, for a constant $k$, the time complexity of the decoder is also polynomial with respect to $n$. Since $N = n + 2\log n + o(\log n)$, the time complexities of both the encoder and decoder are polynomial with respect to $N$.

## V. Conclusion

A $k$-substring edit will have many localized errors such as burst insertions/deletions/substitutions as a special case. Based on two datasets, the statistical hypothesis tests and the codes reaching the GV bounds show that the substring edits are common errors and substring-edit-correcting codes can achieve a lower redundancy compared to insertion/deletion-error-correcting codes. Therefore, this paper constructs error-correcting codes to correct a $k$-substring edit with the redundancy of asymptotically $2\log n$ and polynomial time complexities in both encoding and the decoding algorithms. In addition to the ongoing work, numerous unsolved challenges remain, including the development of systematic codes for correcting substring edits and the creation of error-correcting codes with approximately $\log n$ bits of redundancy.

## References

[1] 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. 1–6.

[2] P. Fire, *A class of multiple-error-correcting binary codes for non-independent errors*. Stanford University, 1959, vol. 55.

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

[4] W. Zhou, S. Lin, and K. Abdel-Ghaffar, "Burst or random error correction based on Fire and BCH codes," in *2014 Information Theory and Applications Workshop (ITA)*. IEEE, 2014, pp. 1–5.

[5] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. Abdel-Ghaffar, "Codes for correcting three or more adjacent deletions or insertions," in *2014 IEEE International Symposium on Information Theory*. IEEE, 2014, pp. 1246–1250.

[6] 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.

[7] C. Schoeny, F. Sala, and L. Dolecek, "Novel combinatorial coding results for DNA sequencing and data storage," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 511–515.

[8] A. Lenz and N. Polyanskii, "Optimal codes correcting a burst of deletions of variable length," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 757–762.

[9] R. Bitar, S. K. 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.

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

[11] S. Wang, Y. Tang, R. Gabrys, and F. Farnoud, "Permutation codes for correcting a burst of at most $t$ deletions," in *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2022, pp. 1–6.

[12] S. Wang, Y. Tang, J. Sima, R. Gabrys, and F. Farnoud, "Non-binary codes for correcting a burst of at most t deletions," 2022. [Online]. Available: https://arxiv.org/abs/2210.11818

[13] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3360–3375, 2020.

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

[15] Y. Tang, H. Lou, and F. Farnoud, "Error-correcting codes for short tandem duplications and at most $p$ substitutions," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1835–1840.

[16] J. Sima, R. Gabrys, and J. Bruck, "Syndrome compression for optimal redundancy codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 751–756.

[17] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," *Nature Biotechnology*, vol. 34, no. 5, pp. 518–524, May 2016.

[18] A. Doricchi, C. M. Platnich, A. Gimpel, F. Horn, M. Earle, G. Lanzavecchia, A. L. Cortajarena, L. M. Liz-Marzán, N. Liu, R. Heckel *et al.*, "Emerging approaches to dna data storage: Challenges and prospects," *ACS nano*, vol. 16, no. 11, pp. 17 552–17 571, 2022.

[19] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "Deepsimulator1. 5: a more powerful, quicker and lighter simulator for nanopore sequencing," *Bioinformatics*, vol. 36, no. 8, pp. 2578–2580, 2020.

[20] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, Oct. 1991, pp. 228–238.

[21] C. R. Mehta and N. R. Patel, "IBM SPSS exact tests," *Armonk, NY: IBM Corporation*, pp. 23–24, 2011.

[22] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[23] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.

## Appendix A
## Alternative interpretation of "INS-TEST"

Section III-B interprets the "INS-TEST" model in the probabilistic edit process by considering each insertion independently. In this section, we introduce another interpretation of considering *distributing $k$ balls into $N$ bins* by considering $k$ uniform balls. According to the proposed edit process in Section III-B, the probability of observing $w$ non-empty bins can be shown as

$$\Pr\left(W = w|k, N\right) = \frac{\binom{N}{w}\cdot\binom{s-1}{w-1}}{\binom{N+s-1}{s}}, \tag{9}$$

where $\binom{N+s-1}{s}$ denotes all the cases of distributing $s$ balls into $N$ bins, and $\binom{N}{w}\cdot\binom{s-1}{w-1}$ denotes all the cases that $w$ bins are not empty

| | RUNS-TEST | INS-TEST | SUBDEL-TEST |
|---|---|---|---|
| Bash | 97.2% | 100% | 97.6% |
| DNA | 95.7% | 94.6% | 76.1% |

Table III: Fraction of sequences rejecting the null hypothesis at $p$-value threshold of $5\%$ with new "INS-TEST" interpretation.

Based on eq (9), we follow the same process in Subsection III-D3 to do the "INS-TEST". The $p$-value is defined as the probability of seeing at most $w$ non-empty bins, i.e., $\Pr\left(W \le w|s = K_i, N = n+1\right)$, where the pmf of $W$ is given by (9). The second column of Table III still shows strong evidence for rejecting the uniform edit process for two datasets.