Towards Density-optimal DNA Storage with Practical Bijective Coding

Jiasen Li[1], Mingkai Dong[1], Zhengwei Qi[1], Haibing Guan[1*]

[1]School of Computer Science, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China, 200240, China

*hbguan@sjtu.edu.cn

Abstract

DNA storage is a promising solution to data explosion due to its high density and longevity. However, existing coding methods struggle to efficiently avoid error-prone sequences while retaining usable ones, leading to low density and high copy numbers. Here we show Siyuan Code, a DNA storage codec that achieves theoretically optimal density. Our method maps binary data to nearly all suitable low-error sequences in a one-to-one manner, using our data structure to catalog and avoid error-prone patterns. This allows operation under very strict biochemical constraints with minimal redundancy. Experiments demonstrate a storage density of 1.66 bits per nucleotide and a capacity of 41 exabytes per gram of DNA, with perfect data retrieval using only six copies per strand. Encoding and decoding speeds reach approximately 1 megabyte per second on commodity hardware, comparable to early USB flash drives. These results bring DNA storage closer to practical deployment.

1 Introduction

The information industry is producing data at an unprecedented scale owing to the widespread adoption of the Internet, cloud computing, smartphones, and the Internet of Things (IoT).

Projections suggest that by 2025, the world will generate 163 zettabytes (ZB) of data, necessitating the equivalent storage capacity of about 16 billion 12-terabyte (TB) hard drives. Current storage media are facing significant challenges in accommodating zettabyte-scale data due to their limited storage density. DNA has emerged as a highly promising medium for storing zettabyte-scale data, owing to its ultra-high density [1, 2]. In addition, DNA boasts a longevity that spans centuries and requires minimal maintenance costs [3]. These compelling advantages have spurred numerous research initiatives aimed at developing fundamental methodologies for data storage in DNA.

DNA storage encodes information as multiple unordered DNA sequences, each of which is made up of bases (`A`,`T`,`C`,`G`). Each sequence can have multiple molecule copies in order to ensure successful data loading. As depicted in Figure 1a, the DNA storage process can be delineated into five steps. Initially, in the *Encoding* step, digital data is converted into nucleotide sequences through in-silico processes. Subsequently, these nucleotide sequences are synthesized into physical DNA strands, either in vivo or in vitro, during the *Synthesis* step. The synthesized strands are then stored in containers in the *Storage* step. To recover the stored digital data, the DNA is sequenced to retrieve the nucleotide sequences in the *Sequencing* step. Finally, these sequences are translated back into digital data during the *Decoding* step.

However, practical implementation faces significant challenges. Some DNA strand patterns are prone to high error rates because the chemical properties of DNA can lead to sequence-specific error rates during synthesis and sequencing [4, 5]. For example, homopolymers such as `AAAA` or `TTTT` are more error-prone during synthesis and sequencing processes. MESA classifies error-prone strands into three main types, namely strands with homopolymers, strands with unbalanced GC content, and strands with undesired motifs [4]. To deal with the error, error avoidance and/or error correction should be adopted. Intra-strand error avoidance is more important than intra-strand error correction because error-prone sequences are not worth error correction, and intra-strand error avoidance can work with inter-strand error correction. Most current methods are derived from those used in digital data storage and communication, integrating error correction codes such as the Reed-Solomon code [6, 7] and Fountain code [8, 9, 10, 11]. Current methods [12, 13, 14, 15, 16, 17, 18, 19, 20] attempt

to circumvent these error-prone and unsuitable sequences (i.e., error-prone strands that are bad for DNA coding) either through filtering-based methods that exclude unsuitable strands after random generation, as with Fountain code [15], or through manual designs [12, 13] that deliberately use only low-error and suitable patterns (i.e., low-error strands that are good for DNA coding; note that a low-error strand describes the objective property, while a suitable strand describes the reasonable choice in DNA coding). Filtering-based methods suffer from either endless repetition of generation and rejection caused by strict constraints or higher error rates followed by higher copy numbers and lower density caused by loose constraints. Filtering-based methods could thus be inefficient, unstable, high in strand error rates and low in density (Supplementary §A.2.1). Manual designs require substantial effort to adapt to changes in synthesis and sequencing technology and often fail to utilize all suitable strands, leading to low density. Also, manual designs are often not compatible with customized patterns or artificial nucleotides (Supplementary §A.1.3). Neither approach maintains all suitable sequences while avoiding all unsuitable ones, so neither fully leverages the potential density of DNA storage with optimal efficiency.

As explained above, despite the theoretical potential for high-density DNA storage, practical and achievable densities are significantly constrained by current techniques [21, 22, 23, 24, 25, 26]. The current methods face the aforementioned challenges in error avoidance, and this limits DNA storage from realizing its full potential.

We contend that the challenge of DNA storage encoding is fundamentally a constrained channel coding problem within the DNA storage context, which necessitates an information-theoretic approach [27]. Constrained channel coding refers to the scenario where the channel exhibits lower error rates under specific patterns, thus motivating the design of codes that primarily utilize strands free from error-prone patterns.

We are not satisfied with classical constrained channel coding approaches, as they typically evolve into manually designed DNA storage coding when applied to this domain [13, 28]. They often employ the segmented approach, where each segment enumerates an encoding set. This can lead to constraints that are either overly strict or overly lenient at the boundaries between segments and the segmented enumeration is slow. Such approaches are highly manual and often overly conservative in design, ensuring only that output DNA sequences

satisfy constraints without guaranteeing the potential to utilize all constraint-compliant sequences. Inspired by enumerative coding in data compression in information theory [29], we seek to establish a one-to-one mapping between all usable DNA sequences in the channel and binary data, thereby ensuring no compliant strand is overlooked. However, the prohibitive complexity of direct enumerative coding makes it impractical. To retain its advantages while overcoming this limitation, we adopt a counting-based encoding approach in place of enumeration.

To address this challenge, we re-evaluate the foundational principles of data storage and propose Siyuan Code, an innovative open-source encoding method designed for DNA-based data storage. Siyuan Code diverges from traditional manually designed or filtering-based methods. Instead, it assigns a unique sequence number to each suitable nucleotide sequence with an acceptable error rate, effectively mapping digital data one-to-one to a specific DNA sequence within a user-defined range, as shown in Figure 2a. By keeping all suitable sequences and excluding all unsuitable ones, the method inherently achieves theoretically optimal information density through its one-to-one property. To circumvent the impracticality of enumerating and storing all feasible sequences, we employ our Trie automaton to identify unsuitable patterns and utilize dynamic programming to streamline our counting-based mathematical construction in encoding and decoding processes. In terms of time complexity, our linear time algorithm ensures that the number of plus/minus/comparison computations in one strand is proportional to strand length in our computation. As a result, changing the strand length or the changing the constraints hardly affect the coding speed of our algorithm, as shown in Figure 3bc.

Our Trie automaton data structure is inspired by the classic trie tree data structure [30, 31] and the Aho-Corasick algorithm [32]. Dynamic Programming (DP) serves as a versatile and widely applicable classic computational method[31]. Siyuan Code employs our Trie automaton—inspired by the Aho-Corasick algorithm—to achieve a practical whole-strand bijective mapping for DNA storage. This approach enables theoretically optimal density, a capability that previous encoding methods could not attain.

Unlike filtering-based methods, which face a dilemma between loose constraints (high error rates) and strict constraints (endless retries), Siyuan Code operates deterministically

and can adopt arbitrarily strict constraints without breakdown. Stricter constraints yield lower single-strand error rates and reliable operation with very low copy numbers. These combined advantages are key to realizing the substantially high storage density reported for Siyuan Code.

Through computational in-silico simulations and chemical in-vitro experiments with text and image files, Siyuan Code delivers information density and storage capacity at competitive levels: 1.66 bits per nucleotide (bits/nt) and 41 exabytes per gram (EByte/g), respectively, with an encoding and decoding speed of $\sim$1 megabytes per second (MB/s). This performance is 63$\times$ faster and 192$\times$ denser than the Fountain Code and 435$\times$ faster and 18$\times$ denser than the Yin-yang Code. This encoding and decoding speed is at the level of the full speed USB1.1 flash drive (1.5 MB/s = 12 Mbps). This speed matches the range of early practical storage devices, yet we achieve it on commodity hardware. When compared to the often cited but highly idealized theoretical limit of 455 EByte/g proposed by Church et.al [33], the storage capacity of our method is notable because it approaches a comparable order of magnitude. Their calculation assumes 2 bits per nucleotide with zero overhead for indexing, error correction, or redundancy - conditions that are practically unattainable, making their limit a loose upper bound.

Siyuan Code offers several additional benefits: (1) it allows for the easy customization of patterns to reflect the nuances of various synthesis, storage, and sequencing technologies; (2) it supports the integration of artificial nucleotides, such as 5-methylcytosine and 5-hydroxymethylcytosine, alongside the canonical nucleotides A, C, G, and T; (3) as a deterministic algorithm, it runs stably regardless of data types with no risk of unsuccessful encoding or decoding caused by post-filtering; (4) it provides the flexibility to incorporate diverse encoding and error-correction techniques as interchangeable sub-modules, akin to building with Lego; and (5) it is open source. With the advantages above, Siyuan Code could potentially be the starting work of a series of works on whole-strand one-to-one codings.

Positioning and Contributions. This work achieves a full-chain contribution, from foundational component innovation to complete system validation, which is concretely manifested across the following four layers:

(1) **Component Innovation of Siyuan Code Based on Classical Ideas:** To address the specific constraints of DNA storage, we conduct targeted improvement and theoretical substantiation of classical ideas. The first core component is our Trie automaton as a data structure. Building upon the idea of the Aho-Corasick algorithm [32], innovations such as the press edge ensure strictly non-amortized time complexity for state transitions, accompanied by theoretical proofs for its construction method and properties. The second core component, based on this automaton, uses the idea of dynamic programming [31] to solve the "suitable DNA strand" counting problem, serving as the DP component. The third core component builds upon the DNA counting solved by the aforementioned DP, utilizing the idea of enumerative coding [29] to construct an encoder/decoder component that performs encoding and decoding based on DNA counts.

(2) **Construction of the Core Algorithm:** Based on the three components above, we propose and deeply integrate dynamic programming counting and bijective encoding into the Trie automaton, designing and implementing a deterministic bijective encoding core that approaches the theoretical limit under complex constraints. This forms the algorithmic heart of the entire system.

(3) **Engineering Realization of a Complete Codec System:** To equip the core algorithm with error-correction and practical functionality, we construct a complete system layer, as shown in Figure 2b. This integrates components such as intra-strand error correction, index addressing, and inter-strand error-correction schemes as pluggable modules, thereby engineering the theoretical bijective encoding into a robust and practical codec capable of processing arbitrary files and resisting biochemical errors.

(4) **Systematic Validation and Performance Benchmarking:** We validate the method's scalability and theoretical performance through in-silico experiments. Furthermore, via rigorous third-party in-vitro experiments under real biochemical conditions, we verify all key claims of the system: achieving perfect data recovery with a low copy number and high physical storage density.

In summary, the contribution of this work lies in proposing a complete DNA storage coding system—innovated and optimized across its underlying data structures and ideas, core algorithm, system implementation, and experimental validation—thereby establishing

a solid theoretical and experimental benchmark for its efficiency, reliability, and high density.

We consider the second layer, the construction of the core algorithm (as shown in Figure 2a), to be the most critical focus. This core algorithm can be viewed as both the integration and engineering application of the first-layer sub-components, and as the theoretical foundation for the third layer.

Detailed theoretical analyses of components are provided in Supplementary §B (together with the comparisons with the classic Aho-Corasick algorithm). Furthermore, given the high degree of integration between our first-layer components and classical ideas, we provide additional clarification regarding the specific boundaries between these classical ideas and our innovations in Supplementary §E.16.

2 Results

2.1 A general overview of Siyuan Code

Siyuan Code constructs a codebook by assigning non-negative integers to DNA sequences that exhibit low error rates, effectively creating a high-density encoding scheme. The codebook deliberately excludes sequences with high-error-rate motifs, such as homopolymers longer than two nucleotides, GC content outside the 40–60% range, and other subsequences known to increase error rates, including problematic primers or structural features.

To encode binary data, the data is segmented into blocks, each of which is translated into a decimal number that corresponds to a binary string. This decimal number is then used as an index to retrieve a specific DNA sequence from the codebook, with the X-th sequence representing the binary string of the integer X. Decoding involves locating the DNA sequence within the codebook to find its index, which is then converted back into a binary string.

However, this direct approach is impractical owing to the exponential growth of suitable sequences with increasing sequence length, a challenge illustrated in Supplementary Figure 13 and Supplementary Figure 19. Maintaining an exhaustive codebook and performing lookups within it would be prohibitively time-consuming and space-intensive. To mitigate this, we

must devise more efficient methods for codebook management and sequence retrieval, as shown in Figure 2a.

To overcome the challenge posed by the vast number of suitable sequences, Siyuan Code employs a Trie structure to efficiently catalogue all patterns unsuitable for DNA storage. These patterns, which do not increase exponentially with sequence length, significantly reduce the required storage space compared to storing suitable sequences directly. As shown in Figure 1c, Siyuan Code enhances the Trie by transforming it into a Trie automaton. This structure, coupled with dynamic programming techniques, substantially accelerates the encoding and decoding processes. This advanced Trie automaton dynamically computes values that facilitate the quick determination of suitable sequences, thereby streamlining the traversal through the Trie structure during the counting-based construction with Dynamic Programming in the encoding and decoding phases. The algorithm is detailed in §4 with its formal proof in Supplementary §B.

As shown in Table 1, coding potential differs from net information density. Coding potential represents an extremely high information density under special settings that disregard error correction and real-world feasibility. For example, Fountain Code removes all redundancy and the index (random seed) when calculating coding potential. As a result, coding potential represents only a theoretical maximum without feasibility requirements. With the density given by coding of previous works, the system does not need to work with error correction. Our 2.0 coding potential is in the setting that we input nothing to the Trie and make zero limitation to the strands, and at the same time we remove the index and error correction and believe the accuracy of DNA synthesis and DNA sequencing. With our extreme setting, our method can simply degenerate into quaternary conversion. Our coding potential setting is very close to the settings of the previous coding potentials. Also, we think that coding potential is too tricky and urge not to use this scoring criterion to evaluate the density of codings. We will show the superiority of our method with other criteria.

As shown in Table 1, net information density differs from actual physical density. Net information density accounts for errors and includes important components such as the index or random seed to ensure real-world feasibility. However, this density scoring criterion can still be manipulated. A sufficiently large copy number for each strand can decrease any

strand error to nearly zero. We do not exploit such manipulation and instead provide a solution with a very low copy number for each strand. With this low copy number, our method remains effective. We acknowledge that net information density can be somewhat misleading. For fair comparison, both copy number and strand length must be controlled. Only simulation can converge all the previous methods with the same setting in copy number and strand length. We will show the superiority of our method with the consideration of copy number, by comparing in-silico net information density and in-vitro physical density (reported results). The copy number is important when decreasing the physical density and a high physical density can save the increasingly precious and costly space in the potential large scale and very long term storage.

Here, we distinguish between in-vitro density and in-vivo density. In-vitro density refers to the actual density of dried DNA powder. It is the data size divided by the powder weight. If each strand has multiple copies, the average copy number of the strands influences the powder weight. In-vitro density excludes the weight of the container because the increase in information amount does not depend on the number of containers, and the amortized weight of the container converges to nearly zero percent as the data amount increases. Theoretically, more than 3^{100} different DNA strands with index can be stored in one huge container. The index length can be decided according to the amount of information to store. In-vivo density does not represent actual density for several reasons. First, multiple cells exist and each cell contains at least one copy of the data. A single living cell would not be sufficiently robust owing to programmed cell death [34]. However, in-vivo density counts the data only once. Second, cells used for in-vivo density calculations cannot be dried into powder, and the weight of water and other cellular components is not negligible [35]. Additionally, a cell such as *E. coli* can store only a limited amount of information [36], and the cost of cells cannot be amortized. This weight is not counted in in-vivo density calculations. Third, only approximately 5–10% of the DNA can be edited while keeping the cell alive [37], and plasmid capacity is much lower than chromosome capacity. The weight of non-editable base pairs is not counted in in-vivo density.

To roughly estimate the limits of the in-vivo physical density, we do the following calculation. A *E.coli* cell has about 4.6 million base pairs [36]. Even if all the bases are used

for storage, causing troubles in cell survival, a cell can only store less than 1.09 MB data ($\frac{4.6\times10^6 \text{ bases}\times 2.0\text{bits/base}}{8\text{bits/byte}\times 1024\text{bytes/KB}\times 1024\text{KB/MB}}$). To get a theoretical bound for in-vivo storage, the immoderate coding without error correction or detection in vivo presents exactly 2 bits with one base pair. The weight of a living cell (10^{-12}g for *E.coli* [35]) results in an actual density of less than 0.9 EByte per gram, which is impractical because there is no extra copy of the cell. It is a rough estimation of the theoretical upper bound of the in-vivo storage and the actual upper bound should be much lower. If 10% of the base pairs are used in coding to keep the cell alive [37], the theoretical upper bound of the in-vivo storage density drops to 0.09 EByte per gram. This upper bound is far smaller than the given so-called in-vivo density.

2.2 In-Silico Experiments

We evaluate Siyuan Code with in-silico experiments and compare it to prior work, including Fountain Code [15], Rotation Code [12], and Yin-yang Code [16]. Recent suboptimal works are not taken into comparison. For example, Aeon Code [38] has a lower density than the previous state-of-the-art method Yin-yang Code [16] in information density, as shown in Table 1. Detailed descriptions and theoretical comparisons of these works are given in Supplementary §A. To compare different coding methods, we use the input data from the Fountain Code paper, consisting of a zip bomb, a Kolibri OS image, a pioneer SVG picture, a book of Shannon, and a small video named *the arrival of a train*. The total data size is 2,145,290 bytes (2.04 MB). Siyuan Code is implemented in Python 3.8 and evaluated with a 2.6GHz six-core i7-10750H (Windows 10 WSL2). Note that the actual running time may vary depending on hardware specifications; however, all performance comparisons were conducted on the same hardware platform. The in-silico experiments are conducted via a custom simulator, DNASim (Supplementary §C), which calculates the overall error rate of encoding systems with error corrections. DNASim utilizes MESA [4] to estimate the error rate of a single input DNA sequence with given parameters. DNASim implements a random DNA error simulator to compare the density of the methods at the same error rate and the error rate at the same density.

We emphasize that the central reported results — the physical density of 41 EB/g and the perfect-retrieval copy number of 6 — are obtained from in-vitro experiments, not from

simulation. The simulator is used primarily for comparative analysis and design guidance, and all simulation-derived results are explicitly labeled as such throughout the manuscript.

It is possible to achieve unreasonably high density when error rate is not considered and successful decoding is disregarded. Therefore, we compared **density under different system error rates** to avoid unreasonable exploitation of the evaluation criteria. The system error rate is the probability that the system can not perfectly retrieve the information. As system error rates are key factors affecting the density, we compare the storage density of Siyuan Code to prior methods under different system error rates. Figure 3a shows curves depicting the relationship between net information density and system error rates. The x-axis quantifies the net information density, defined as the ratio of the number of bits effectively stored to the total number of nucleotides used, including those allocated for error correction and indexing purposes. The y-axis represents the system error rate.

For each encoding method, we adjusted the level of redundancy applied during data encoding to generate a curve demonstrating the interplay between net information density and system error rates. As expected, introducing more redundancy diminishes the error rate but at the expense of storage density. This inverse relationship is evident as higher net information density correlates with increased system error rate. Notably, Siyuan Code (`SYC-150-dup`) outperforms other methods (`FC-150-dup` and `YYC-150-dup`) by achieving the highest net information density while maintaining the lowest system error rate, striking a better balance between data storage density and reliability.

To deepen our understanding of the performance characteristics of Siyuan Code, we conducted evaluations varying the copy number (duplication). In practice, to enhance the reliability of DNA storage, each sequence is typically synthesized several times or amplified into multiple copies through Polymerase Chain Reaction (PCR). This copy number redundancy allows each unique DNA strand to be sequenced multiple times (achieving a larger sequencing depth), facilitating the accurate reconstruction of the original DNA sequence with the aid of error detection codes embedded within the strands. In Figure 3a, the curves labeled with "-dup" represent scenarios where this replication technique is employed to augment net density. The results indicate that Siyuan Code with a strand length of 150 nucleotides and duplication (denoted as `SYC-150-dup`) surpasses the performance of Siyuan Code with

the same strand length but without duplication (denoted as `SYC-150`). Both configurations of Siyuan Code significantly outperform prior encoding methods by a considerable margin. Furthermore, in the case of `SYC-150-dup`, 1.66 bits/nt in Table 1 is sufficient for perfect information retrieval.

Strand length is also a critical factor influencing storage density. Our observations reveal that as the strand length is extended from 50 to 200 nucleotides (`SYC-50` to `SYC-200`), there is a corresponding increase in storage density at comparable system error rates. This trend can be attributed to the fact that the overhead associated with fixed-length indexing and checksums constitutes a larger proportion of the total sequence in shorter strands. Consequently, longer strands, which dilute this overhead across more nucleotides, benefit from improved storage density.

Computational efficiency: Post-filtering methods face inefficiency caused by numerous rejection and retry loops under strict constraints when attempting to achieve low error rates with small copy numbers. Otherwise, post-filtering methods with loose constraints would increase either the logical redundancy or the copy number redundancy to gain successful decoding and both would lose high storage capacity. Figure 3b and c demonstrate the computational efficiency of Siyuan Code and other DNA encoding methods. We used the aforementioned 2.04 MB file from the Fountain code paper as the input and evaluated the performance of encoding and decoding. In Figure 3b, we run different encoding methods with various strand lengths under the condition that 4-homopolymers are forbidden and show the running time. Siyuan Code outperforms its counterparts significantly, being 63× and 79× faster than the Fountain code with a 4-homopolymer restriction in encoding and decoding, respectively. Compared to Yin-yang code with the same restriction, Siyuan Code is 435× and 106× faster in encoding and decoding, respectively. For very long strands, see Supplementary §E.11.

Notably, as strand length increases, there is a noticeable upward trend in running time for Fountain Code. Filtering methods such as Fountain code suffer from the need to reject numerous unsuitable strands, leading to a rapid increase in running time with longer strand lengths. This occurs because the number of unsuitable strands rises exponentially as length increases. Assuming that every k bases have a probability e of containing an unsuitable

substrand, a strand of length n would have a cumulative probability of at least $(1-e)^{\frac{n}{k}}$ to encounter such a substrand. This results in an exponentially decreasing likelihood of avoiding unsuitable substrands in the strand as n grows. Consequently, an exponentially large number of strands can lead to reject-and-retry cycles, leading to prohibitively long running time. As synthesis technology keeps advancing to allow for longer strands, filter-based methods struggle to keep pace due to their inability to efficiently process the increased strand length.

Figure 3c presents the running times for various encoding methods with different homopolymer constraints at a fixed strand length of 200 nucleotides. The x-axis represents the length of the forbidden homopolymer sequence, and the y-axis shows the running time. The figure demonstrates that encoding time increases as constraints become tighter. Fountain and Yin-yang codes fail to function when 3-homopolymers are forbidden. These filtering-based methods are particularly challenged by these stringent conditions, as they must reject a high number of unsuitable strands. The rate of unsuitable sequences accumulates along the length of the strand, leading to a compounded growth in errors that can severely compromise reliability.

Single-strand error rate: The high efficiency independent of constraint stringency enables Siyuan Code to adopt strict constraints and achieve a low single-strand error rate. Single strand error rate can again increase the average amount of information stored per strand, as shown in Supplementary §E.1. Figure 3d illustrates the single-strand error rates associated with various encoding methods. The single-strand error rate measures the likelihood of a strand being inaccurately decoded back into its original binary form. In this evaluation, we employed the default settings for each encoding method, which means that the system error rates were not intentionally equalized for comparison. The Fountain and Yin-yang encoding methods exhibit single-strand error rates exceeding 60%, suggesting that significant redundancy is required to achieve system error rates that are within an acceptable range. In contrast, Siyuan Code demonstrates its intrinsic advantage by leveraging an understanding of DNA's constraints, achieving the lowest single-strand error rate of just 2.2%. This indicates that, even with a copy number of one, Siyuan Code maintains a high likelihood of accurate strand decoding. When the copy number is increased to five, as indicated by

the curves suffixed with "-dup", the single-strand error rate for Siyuan Code approaches zero. Meanwhile, the error rates for Fountain and Yin-yang codes decrease to 33.5% and 36.9%, respectively, showcasing the benefits of increased copy number and sequencing depth in reducing error rates.

In contrast to filtering-based encoding methods such as Fountain Code, Determinism and stability: Siyuan Code produces consistent outputs for identical inputs. This determinism benefits both debugging and result replication. Furthermore, determinism enables Siyuan Code to work with any positive integer as the maximum homopolymer length, not only the value of 2 shown in Table 1. Fountain Code, and other filtering-based methods, depend on a random seed and a pseudo-random number generator (PRNG) to transform information into DNA sequences. Either loss of the seed or changes in the PRNG can break the connection between the sequencing output and the original file, causing failures of decoding.

In addition to determinism, Siyuan Code can be stably applied to small inputs under strong constraints. Owing to randomness and filtering, existing filtering-based methods fail to produce correct strands for small files and sometimes enter infinite loops. As shown in Figure 3e, when encoding the first paragraph (see the text file in the git repository) of Wikipedia's self-introduction page (https://en.wikipedia.org/wiki/Wikipedia) with a 4-homopolymer forbidden, the Yin-yang code has approximately a 23% chance of failing to find a pair of suitable blocks to form a suitable sequence. In other words, all strands that Yin-yang code generates fail to satisfy the constraints, causing the algorithm to become stuck in an infinite loop. This failure rate escalates to 100% when the restriction is tightened to forbid 3-homopolymers, rendering the encoding of nearly any file incorrect under such stringent constraints. In contrast, Siyuan Code operates successfully under these challenging conditions, demonstrating superior stability with both small inputs and stringent constraints.

2.3 In-Vitro Experiments

To assess the practical performance of Siyuan Code, we conducted a real-world test using a diverse set of data, including the logo of our university, a "Hello World" C++ program, a textual introduction to the university, and an Atari game video. This file collection was compressed into a 30KB file using 7-zip [39, 40] and then encoded with Siyuan Code. After

synthesizing the DNA strands, we sequenced them and applied Siyuan Code to decode the data. The result was a complete and accurate reconstruction of the original compressed file, with all components intact and error-free.

Experiment Parameters and Setup

In synthesis, each DNA strand is 170 nt in length, including the 18-nt forward primer and the 18-nt reverse primer. The forward primer is chosen as "CTCGTAGACTGCGTACCA", and the reverse primer is chosen as "GACGATGAGTCCTGAGTA" [41]. Figure 4d illustrates that the internal 134 nucleotides encode 32 bytes of data, including 27 bytes for file data, two bytes for indexing, and a three-byte checksum, which is the first 24 bits of the file data's SHA-1 results. The transformation of these 134 nucleotides and 32 bytes is decided by Siyuan Code, where the maximal allowed homopolymer length is two, and the allowed GC percentage is 20%–80%. The GC percentage is controlled by the law of large numbers by excluding all five-nucleotide subsequences whose GC content falls outside 20%–80%. Undesired motifs, including primers and subsequences (adjacent substrings of a DNA strand) known to have high error rates [4], are also excluded from the Siyuan Code codebook. We chose Reed-Solomon Code for error correction across strands, and the parameter is $k = 58$ and $n = 63$, i.e., five parity sequences are generated for every 58 sequences in an error correction group. The parameters listed above should be kept by the file owner to recover the data.

The DNA strands were synthesized using chemical chips [42, 24, 25], with each sequence replicated into six physical copies during synthesis. The strands were then dried and stored without PCR (Polymerase Chain Reaction). To prepare for sequencing, we amplified the DNA strands through PCR to facilitate multiple sequencing iterations. On average, each strand underwent approximately 510.8 sequencing passes. Subsequently, the resulting sequences were merged using a paired-end read merger [43] and repaired through error detection and correction algorithms.

Evaluation of Physical Density

About 95.3% of the sequences are measured correctly in the sequencing, and all sequences are recovered with the error correction. The physical information density can be estimated as follows. The average molecular weight of a base is 308.95 g/mol. As expected, the molecular weight of a strand is $79 + 308.95 \times 170 = 52600.5$ g/mol. The weight of the

strands is about $1253 \times 6 \times 52600.5/(6.022 \times 10^{23}) = 6.5668 \times 10^{-16}$ g. The information of the strands is 31127 bytes. The size before compression is 86808 bytes. The density is $31127/(6.5668 \times 10^{-16}) = 47.4 \times 10^{18}$ bytes/g ≈ 41 EByte/g. Considering the compression, the density is $86808/(6.5668 \times 10^{-16}) = 132.2 \times 10^{18}$ bytes/g ≈ 115 EByte/g.

Note that the 41 EByte/g is the real-world wet lab density. If we ignore the copy number 6 and consider the compression as a part of the code, the density can then become an eye-catching presentation with the 688 EByte/g. To avoid misunderstanding and avoid unfairness, we describe the different densities of our method in the Supplementary §E.10.

Figure 4a displays a comparative analysis of physical density across various storage methods, with the horizontal axis representing the methods and the vertical axis indicating the physical density. The comparison visually illustrates that Siyuan Code achieves a significantly higher density than previous state-of-the-art methods by one to two orders of magnitude. The concrete numbers can be seen in Table 1.

Comparison of Sequence Designs

Figure 4d contrasts the sequence designs of oligonucleotides. To encode the same 32 bytes of information within a single strand, Siyuan Code requires only 189 nucleotides (nt), whereas Yin-yang Code (YYC) and Fountain Code (FC) both require 200 nt. This highlights a substantial difference between previous methods and Siyuan Code. The latter does not prescribe a fixed strand length for each segment but instead specifies the volume of information to be stored per strand. Moreover, Siyuan Code is compatible with more cost-effective synthesis lengths, such as 170 nt in vitro (see §2.3).

For reference, we calculate the density of Yin-yang Code exclusive of fixed overhead. The in-vitro physical information density of Yin-yang Code stands at 2.25 exabytes per gram (EByte/g). Within each 200 nt strand, there is a 16 nt Reed-Solomon (RS) error-correcting code, a 16 nt index, and two primers, each 20 nt long. Excluding the two indexes, the density of Yin-yang Code increases to $2.25 \times \frac{200}{184} = 2.45$ EByte/g. When further removing the error-correcting code and primers, the density escalates to $2.25 \times \frac{200}{128} = 3.52$ EByte/g. This analysis reveals that the enhanced density of Siyuan Code is not solely due to reduced redundancy. It also benefits from a lower error rate and consequently a reduced average copy number per oligonucleotide sequence.

Error Analysis

The evaluated data of Siyuan Code in the vitro experiments is organized into twenty error correction groups. Each group comprises 58 data strands and five parity strands, resulting in an approximate redundancy rate of 8%.

Figure 4c presents the error rate, required redundancy rate, and actual redundancy rate for these groups. The error rate is determined by the proportion of strands that cannot be accurately read after employing the paired-end read merger technique. The required redundancy rate is the ratio of the minimal number of additional strands needed for group recovery with inter-strand error correction to the total number of data strands. The redundancy rate, which is not less than the maximal redundancy rate needed among the groups, is the actual redundancy rate in vitro experiments. The distribution of the required redundancy rate resembles a normal distribution, ensuring that redundancy is utilized efficiently and not excessively. The experiment results show that each group's actual redundancy rate is no less than the required redundancy rate, indicating that Siyuan Code can successfully decode all groups.

Figure 4b illustrates the distribution of edit distances between correctly synthesized strands and their closest sequenced counterparts. The horizontal axis represents edit distance, and the vertical axis shows the number of strands at each distance. Edit distances less than 5 typically result from sequencing errors, whereas distances greater than 15 indicate strand loss. Notably, our method achieves a molecule copy number as low as 6, significantly lower than previous efforts. The missing strands are likely due to limited sequencing depth and copy number, and we observe a decrease in the error rate with increased sequencing depth and copy number.

3 Discussion

Siyuan Code proposes a method for converting digital data into DNA sequences. While the basic idea is straightforward, Siyuan Code addresses the fundamental challenges of excessive memory and computation requirements. Siyuan Code provides several advantages over existing solutions. First, Siyuan Code establishes a one-to-one mapping from suitable strands

to binary data through the codebook, enabling convergence to the theoretical optimal density of DNA storage. Actually, the in-vitro results demonstrate that Siyuan Code achieves a physical density of 41 exabytes per gram (EByte/g). This can be contextualized against the often-cited but highly idealized theoretical limit of 455 EByte/g (Church et.al [33]), which assumes 2 bits per nucleotide with zero overhead and thus represents a loose upper bound for an essentially unattainable scenario. Second, the encoding and decoding procedures of Siyuan Code are deterministic and efficient. In contrast to previous methods such as Fountain Code and Yin-yang Code, which depend on randomness and reject-and-retry cycles and suffer from random encoding and decoding failures, Siyuan Code operates without randomness or unbounded reject-and-retry cycles. The experiment shows Siyuan Code runs at the speed of the same level as full speed USB1.1 flash drive. Third, this efficiency enables Siyuan Code to adopt strict constraints that enhance single-strand accuracy. With higher accuracy, the copy number can be substantially reduced. The reduction in copy number can increase the physical density. Siyuan Code can perfectly retrieve all the data with a small copy number, which means that it is more robust with larger copy numbers. Fourth, Siyuan Code is a highly customizable approach that permits modification of its one-to-one mapping through user-defined rules. This versatility allows Siyuan Code to be fine-tuned for enhanced precision and to adapt to various conditions, including emerging synthesis or sequencing technologies and the incorporation of artificial nucleotides.

Siyuan Code also has certain limitations. First, control over global GC content in Siyuan Code is subject to the law of large numbers, meaning that while GC content may approach 50% on average, there is a small but non-zero probability of deviation from this ideal. The deviation probability is so small (as proved in Supplementary §B.1.3 and shown by the experimental result in Supplementary Figure 4) that it could be ignored to some extent and Siyuan Code works well with sufficient redundancy. To avoid ambiguity, we would like to clarify: only the global GC content, which tends to approach 1/2 as strand length increases, is governed by the law of large numbers. In contrast, the local GC content is actively constrained by Siyuan Code (as explained in Supplementary §B.1.2). Alternatively, we recommend filtering strands after generation with Siyuan Code, which is more efficient than previous methods. This efficiency arises because rejecting sequences based solely on global

GC content, which naturally tends toward $\frac{1}{2}$, is statistically inexpensive, whereas other constraints (e.g., local motifs, homopolymers) do not resolve automatically and would otherwise require far more costly filtering. Crucially, Siyuan Code avoids the need to generate an exponentially large pool of unsuitable strands, making the process feasible within polynomial time even for long strands and stringent constraints. In contrast, earlier post-filtering techniques struggle with efficiency due to the vast number of unsuitable strands that need to be randomly generated and then discarded (as explained in Supplementary §E.8). Second, Siyuan Code does not account for inter-strand chemical reactions. However, this limitation is not expected to significantly impact sequencing results, as such reactions are unlikely to interfere with the process. Also, it can be solved with minor revision of Siyuan Code in dynamic data structure. Finally, the example application of Siyuan Code is based on the error pattern described by MESA [4] and therefore depends on the accuracy of that error model. However, other error models and simulators can be applied, and Siyuan Code is not inherently limited to a single error profile.

We admit that our testcases are not very large, but it is large enough to show the property and the advantage of Siyuan Code. The discussion of data scale can be found in Supplementary §E.5.

We plan to explore the application of Siyuan Code encoding in in vivo settings, as well as its adaptation to emerging sequencing technologies such as Nanopore, in future work. However, these aspects will not be elaborated in the current paper, as its primary focus and core contribution lie in validating the advantages of Siyuan Code encoding through in vitro methods.

4 Methods

4.1 Basic Ideas and the Strawman

We argue that DNA storage encoding is, at its core, a constrained channel coding problem, demanding an information-theoretic solution. Constrained channel coding arises when a channel demonstrates reduced error rates under specific sequence patterns, thereby guiding

code design toward strands that avoid error-prone motifs [27].

Instead of developing an approach to transform binary strings into DNA sequences via complex encoding rules, we enumerate all DNA sequences suitable for DNA storage and build a one-to-one mapping between binary strings and these suitable sequences. By looking up the mapping, we can easily transform between binary strings and DNA sequences. This is inspired by enumerative coding in data compression in information theory [29].

Figure 5 shows an illustration of the approach. Specifically, with a given length M, we enumerate all 4^M possible DNA sequences of length M and filter suitable sequences. Supposing we get Q suitable sequences, we map each of them to the number from 0 to Q-1 (both inclusive). Consequently, we get a one-to-one mapping of DNA sequences of length M and the numbers from 0 to Q-1. We use these numbers to present $N = \lfloor log_2 Q \rfloor$ binary bits. Note that since we build the mapping for an integer number of binary bits, not all Q states are used in the current method. A variation of the method can first transform binary data to Q-ary strings to fully utilize these Q states. As a result, we can get a DNA sequence of length M to present a binary string of length N.

The one-to-one mapping method is more straightforward than existing DNA coding methods. However, to reduce the offcut percentage in the mapping and satisfy DNA storage requirements, M must be sufficiently large. This requirement introduces several critical issues that hinder its use in storage applications. First, the number of permutations increases exponentially with M, and filtering suitable sequences from numerous candidates could require significant time. Second, storing the one-to-one mapping requires substantial storage space. Third, encoding and decoding methods require direct and inverse lookups in the mapping, which consume substantial time when the mapping is large. We thus propose Siyuan Code based on the one-to-one mapping method with several optimized designs to solve these issues.

4.2 Siyuan Code Method

Siyuan Code Core Algorithm (Figure 2a) originates from the one-to-one mapping concept and is optimized by the observation that although the number of suitable sequences can be large, the patterns of unsuitable sequences are simple and easy to represent and store. Siyuan Code leverages a Trie [31] structure to store all patterns of unsuitable sequences,

with which the one-to-one mapping of binary strings and suitable sequences can be derived via computation.

Patterns of Unsuitable Sequences. As stated previously, a suitable sequence must satisfy three rules.

1. No homopolymer longer than K;
2. GC content percentage between $\mathrm{GC}_{\min}$ and $\mathrm{GC}_{\max}$;
3. No undesired motifs in the sequence.

Accordingly, an unsuitable sequence should contain at least one subsequence matching one of the following patterns:

1. $K+1$ repeated nucleotides, e.g., "AAAA" and "TTTT";
2. Sequences of a given length (e.g., five) with GC content percentage less than GC_{min} or larger than GC_{max} (e.g., 20% and 80%);
3. Undesired motifs.

Most patterns are straightforward. For GC content percentage, we limit the pattern sequence length to five and relax the acceptable GC content percentage range to 20% to 80%. As shown and proved in Supplementary §B.1.3, the method largely excludes permutations that cause high error rates due to extreme GC rates in sequences. For permutations not filtered by the pattern, the errors can be corrected via our error-correcting algorithm, as our experiments show in §2.

Although the number of unsuitable sequence patterns is much smaller than that of suitable sequences, an efficient data structure is still required to store all patterns. As shown in Figure 1c, Siyuan Code stores all unsuitable patterns in a Trie [30] structure, where each pattern is represented by a path from the root to one of the annotated nodes. For example, in Figure 1c, `TTATA` is an unsuitable subsequence stored in the Trie. The subsequence is presented by the node indicated by the path `T`, `T`, `A`, `T`, `A`, successively from the root. As an optimization, for two unsuitable sequences, A and B, if A is a prefix of B, we can remove B

from the storage since all sequences containing B always contain A. Avoiding the existence of A will also eliminate the appearance of B in DNA sequences.

Automaton and Dynamic Programming. We build a Trie Automaton from the Trie to facilitate encoding and decoding. Specifically, in addition to the edges in the original Trie (*tree edges*), we calculate and add two additional edge types for each node, namely *press edges* and *fail edges*. We leave the automaton construction and proof in the appendix and emphasize its high efficiency in the appendix since the proof and analysis are exactly based on traditional methods.

With *press* edges and *fail* edges, we further facilitate encoding and decoding using dynamic programming. We define $\text{Path}(v)$ as the sequence represented by the path from the root node to node v. We define $\text{child}_c(v)$ as the child node of v by following the edge (i.e., nucleotide) c. We define $F(v)$ to be the node so that $\text{Path}(F(v))$ is the longest suffix of $\text{Path}(v)$ in the tree. $F(v)$ is the same as the traditional "fail link" or "suffix link" of the AC Algorithm [32].

$$F(\text{child}_c(v)) = \begin{cases} \text{root} & F_c^*(v) \text{ is } \emptyset \\ \text{child}_c(F_c^*(v)) & otherwise \end{cases} \tag{1}$$

$F_c^*(v)$ is defined to be the shortest sequence of F acting on the node v until it has child c. If such a sequence does not exist, return $\emptyset$.

$$F_c^*(v) = \begin{cases} \emptyset & v \text{ is root} \\ F(v) & \text{if } \text{child}_c(F(v)) \text{ exists} \\ F_c^*(F(v)) & \text{otherwise} \end{cases} \tag{2}$$

For any nucleotide c (i.e., $c \in \{ATCG\}$), we define $P_c(v)$ to be the node that $\text{Path}(P_c(v))$ is the longest suffix of $\text{Path}(v) + c$ in Trie. $P_c(v)$ can be calculated with:

$$P_c(v) = \begin{cases} child_c(v) & \text{if } child_c(v) \text{ exists} \\ \text{root} & \text{if } v \text{ is root and } child_c(v) \text{ does not exist} \\ P_c(F(v)) & otherwise \end{cases} \tag{3}$$

With *fail edge* $F(\cdot)$ and *press edge* $P_c(\cdot)$, we further calculate an auxiliary array for each node. Specifically, for each node v, Siyuan Code calculates $\mathrm{PC}(v, t)$, the number of paths of length t starting from node v without visiting any unsuitable node. An unsuitable node is a leaf of a Trie tree, which presents an unsuitable subsequence stored in the Trie. An unsuitable endpoint is a node that visits an unsuitable node pressing the right side primer from the end point node.

By dynamic programming and the Trie automation, we can calculate $\mathrm{PC}(v, r)$ for all nodes in the Trie and all $0 \leq r \leq m$:

$$\mathrm{PC}(v, r) = \begin{cases} 0 & v \text{ is unsuitable} \\ 1 & v \text{ is not unsuitable and } r = 0 \\ \sum_c \mathrm{PC}(P_c(v), r-1) & v \text{ is suitable and } r > 0 \end{cases} \tag{4}$$

The first two cases are intuitive; the last case is based on an observation that a suitable path of length $r > 0$ from v can be seen as the combination of two parts: an edge following the nucleotide c and a path of length $r - 1$ from $P_c(v)$. Thus, the number of suitable paths of length $r > 0$ from v can be obtained by enumerating the first edge (nucleotide c) and summing the number of suitable paths of length $r-1$. Note that it does not matter whether $P_c(v)$ is an suitable node, because $\mathrm{PC}(P_c(v), r-1) = 0$ if $P_c(v)$ is unsuitable. A small revision to PC shown in the appendix can be taken in order to take the primer into consideration and the idea is the conditional counting of PC under the condition that the reverse primer is added to the sequence.

Encoding. In Siyuan Code, encoding transforms any binary input to DNA sequences using Trie Automaton and dynamic programming array (the path count array, i.e., $\mathrm{PC}(v, r)$).

The core procedure in encoding is to find the $K + 1^{th}$ lexicographical smallest suitable

sequence of length L. Let $\mathrm{Enc}(v, k, l)$ be the $k+1^{th}$ smallest suitable path of length l from v.

$$\mathrm{Enc}(v, k, l) = c + \mathrm{Enc}(P_c(v), k - \sum_{i<c} \mathrm{PC}(P_i(v), l-1), l-1), \tag{5}$$

where $i, c \in \{ATCG\}$ and c is the largest nucleotide such that $P_c(v)$ is a suitable node and $\sum_{i<c} \mathrm{PC}(P_i(v), l-1) \leq k$. The calculation is not difficult. At node v, we enumerate the next nucleotide in the sequence in lexicographical order. For a nucleotide i, there are $\mathrm{PC}(P_i(v), l-1)$ suitable paths of length $l-1$. In other words, there are $\mathrm{PC}(P_i(v), l-1)$ suitable sequences of length l, starting with i from the node v. If the $\mathrm{PC}(P_i(v), l-1)$ is larger than k, then i is the right nucleotide to pick; otherwise, we subtract $\mathrm{PC}(P_i(v), l-1)$ from k and try the next nucleotide. The output of the encoder should be $\mathrm{Enc}(\mathrm{root}, K, L)$ if there is no primer. A small revision to the process shown in the appendix can be taken in order to take the primer into consideration and the idea is the conditional counting finding the $K+1^{th}$ under the condition that the forward primer and the reverse primer are added to the sequence.

Decoding. Decoding is the reverse process of encoding. Given a nucleotide sequence, it counts the number of suitable sequences lexicographically smaller than that sequence. Let $\mathrm{Dec}(v, s, l)$ be number of suitable sequences lexicographical smaller than the sequence $s = (c_1, c_2, \cdots, c_l), c_i \in \{ATCG\}$ from v.

$$\mathrm{Dec}(v, s, l) = \sum_{i<c_1} \mathrm{PC}(P_i(v), l-1) + \mathrm{Dec}(P_{c_1}(v), s\prime, l-1), \tag{6}$$

where $i \in \{ATCG\}$ and $s\prime = (c_2, \cdots, c_l)$. The calculation is straightforward. At node v, all suitable sequences of length l that are smaller than s consist of two categories: those starting with c_1 and those starting with i where $i < c_1$. The number of sequences starting with c_1 is calculated by $\mathrm{Dec}(P_{c_1}(v), s\prime, l-1)$ where $s\prime$ is the subsequence after the first c_1. The number of sequences starting with nucleotide smaller than c_1 is calculated by summing $\sum_{i<c_1} \mathrm{PC}(P_i(v), l-1)$ for $i \leq c_1$. The output of the decoder is $\mathrm{Dec}(\mathrm{root}, S, L)$, where S is the given sequence and L is its length if there is no primer. A small revision to the process

shown in the appendix can be taken in order to take the primer into consideration and the idea is the conditional counting under the condition that the forward primer and the reverse primer are added to the sequence.

Chunking and Redundancy. Before encoding, Siyuan Code treats the binary message as a binary sequence, which is first divided into chunks for DNA synthesis. To preserve the order of chunks, each chunk is attached with an index sequence indicating the chunk order, as shown in Figure 2b.

In our in-silico experiments, each slice consists of 256-bit data, 16-bit index, and 8-bit SHA checksum. In our in-vitro experiments, each slice consists of 216-bit data, 16-bit index, and 24-bit SHA checksum. Although the encoding of Siyuan Code already excludes strands with high error rates during synthesis, storage and sequencing, errors might still occur with a non-zero probability. Thus, Siyuan Code leverages *intra-slice error detection* and *inter-slice error correction* to tolerate accidental errors. Note that the parameters above can be user-defined, so the index can be longer for larger files.

As shown in Supplementary Figure 9, 8-bit checksum value is appended to each slice. The checksum covers both data and index so that any error in the slice will cause checksum failures. The intra-slice error detection helps the sequencing with more depth, and is used as a sign of whether the data in the slice can be considered corrupted for inter-slice error correction.

As errors in different slices can be considered independent, we adopt erasure code for inter-slice error correction. Taking Reed-Solomon code as an example, 12 redundant slices are added for every 100 slices. Note that only the data part is protected by the erasure code, and redundant slices has their own checksum covering their index and the parity data.

The Complete Procedures of Encoding and Decoding. As shown in Figure 2b, the encoding procedure:

1. Data is cut into chunks;

2. Redundancy slices are added to support inter-slice error correction;

3. The index is inserted into slices to preserve slice orders;

4. Checksum is calculated for intra-slice error detection;

5. Each binary sequence is translated to DNA sequences.

The decoding procedure is basically the opposite:

1. Select strands and decode them into binary slices. Note that multiple duplicates of slices are kept to decrease the error rate;

2. Remove binary slices with the checksum failures;

3. Sort strands using the index;

4. Recover missing or corrupted slices using inter-slice error correction;

5. Get the stored data by concatenating the data part in order.

Statistics & Reproducibility

No statistical method was used to predetermine sample size. No data were excluded from the analyses. The experiments were not randomized. The investigators were not blinded to allocation during experiments and outcome assessment. The core coding scheme presented in this work is a deterministic algorithm whose theoretical properties are established analytically. The in-vitro experiments (DNA synthesis, storage, and sequencing conducted by GenScript) serve as experimental validation of the coding scheme's practical feasibility. All results are reproducible using the publicly available code and data (see Code Availability and Data Availability).

Data Availability

The experiments are conducted and verified by a third-party organization (GenScript). The sequenced results (fastq) by GenScript with the stored files are publicly available at:

https://github.com/DNASiyuanCode/SiyuanCodeSequencing

https://gitea.sjtu-edu.top/SiyuanCode/SiyuanCodeSequencing

The sequencing data have also been deposited in Zenodo 10.5281/zenodo.21357059 [44]. The data and code can also be accessed via the Code Ocean capsule. Source data are provided with this paper.

Code Availability

The Python implementation of Siyuan Code is publicly available at:

https://github.com/DNASiyuanCode/SiyuanCode

https://gitea.sjtu-edu.top/SiyuanCode/SiyuanCode

The code has also been deposited in Code Ocean 10.24433/CO.4952245.v1 [45].

References

[1] MR Wallace. Molecular cybernetics: the next step? *Kybernetes*, 7(4):265–268, 1978.

[2] Andy Extance. How dna could store all the world's data. *Nature*, 537(7618), 2016.

[3] Carter Bancroft, Timothy Bowler, Brian Bloom, and Catherine Taylor Clelland. Long-term storage of information in dna. *Science*, 293(5536):1763–1765, 2001.

[4] Michael Schwarz, Marius Welzel, Tolganay Kabdullayeva, Anke Becker, Bernd Freisleben, and Dominik Heider. Mesa: automated assessment of synthetic dna fragments and simulation of dna synthesis, storage, sequencing and pcr errors. *Bioinformatics*, 36(11):3322–3326, 2020.

[5] Michael G Ross, Carsten Russ, Maura Costello, Andrew Hollinger, Niall J Lennon, Ryan Hegarty, Chad Nusbaum, and David B Jaffe. Characterizing and measuring bias in sequence data. *Genome biology*, 14:1–20, 2013.

[6] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[7] Ufuk Demir and Ozlem Aktas. Raptor versus reed solomon forward error correction codes. In *2006 International Symposium on Computer Networks*, pages 264–269. IEEE, 2006.

[8] Michael Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–271. IEEE Computer Society, 2002.

[9] John W Byers, Michael Luby, and Michael Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected areas in Communications*, 20(8):1528–1540, 2002.

[10] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[11] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

[12] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, jan 2013.

[13] Robert N Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J Stark. Robust chemical preservation of digital information on dna in silica with error-correcting codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, 2015.

[14] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649, 2016.

[15] Yaniv Erlich and Dina Zielinski. Dna fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.

[16] Zhi Ping, Shihong Chen, Guangyu Zhou, Xiaoluo Huang, Sha Joe Zhu, Haoling Zhang, Henry H Lee, Zhaojun Lan, Jie Cui, Tai Chen, et al. Towards practical and robust dna-

based data archiving using the yin–yang codec system. *Nature Computational Science*, 2(4):234–242, 2022.

[17] David Buterez. Scaling up dna digital data storage by efficiently predicting dna hybridisation using deep learning. *Scientific Reports*, 11(1):20517, 2021.

[18] Meinolf Blawat, Klaus Gaedke, Ingo Hütter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W. Pruitt, and George M. Church. Forward Error Correction for DNA Data Storage. *Procedia Computer Science*, 80:1011–1022, January 2016.

[19] SM Hossein Tabatabaei Yazdi, Han Mao Kiah, Eva Garcia-Ruiz, Jian Ma, Huimin Zhao, and Olgica Milenkovic. Dna-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 1(3):230–248, 2015.

[20] SM Tabatabaei Yazdi, Yongbo Yuan, Jian Ma, Huimin Zhao, and Olgica Milenkovic. A rewritable, random-access dna-based storage system. *Scientific reports*, 5(1):1–10, 2015.

[21] Jerrod J Schwartz, Choli Lee, and Jay Shendure. Accurate gene synthesis with tag-directed retrieval of sequence-verified dna molecules. *Nature methods*, 9(9):913–915, 2012.

[22] Yaniv Erlich, Kenneth Chang, Assaf Gordon, Roy Ronen, Oron Navon, Michelle Rooks, and Gregory J Hannon. Dna sudoku—harnessing high-throughput sequencing for multiplexed specimen analysis. *Genome research*, 19(7):1243–1253, 2009.

[23] Sriram Kosuri and George M Church. Large-scale de novo dna synthesis: technologies and applications. *Nature methods*, 11(5):499–507, 2014.

[24] Sriram Kosuri, Nikolai Eroshenko, Emily M LeProust, Michael Super, Jeffrey Way, Jin Billy Li, and George M Church. Scalable gene synthesis by selective amplification of dna pools from high-fidelity microchips. *Nature biotechnology*, 28(12):1295–1299, 2010.

[25] Nikolai Eroshenko, Sriram Kosuri, Adam H Marblestone, Nicholas Conway, and George M Church. Gene assembly from chip-synthesized oligonucleotides. *Current protocols in chemical biology*, 4(1):1–17, 2012.

[26] Joshua Sampson, Kevin Jacobs, Meredith Yeager, Stephen Chanock, and Nilanjan Chatterjee. Efficient study design for next generation sequencing. *Genetic epidemiology*, 35(4):269–277, 2011.

[27] Robert Gallager. *Information Theory and Reliable Communication*. Springer, Vienna, 1972.

[28] Daniella Bar-Lev, Itai Orr, Omer Sabary, Tuvi Etzion, and Eitan Yaakobi. Scalable and robust DNA-based storage via coding theory and deep learning. *Nature Machine Intelligence*, 7(4):639–649, April 2025. Publisher: Nature Publishing Group.

[29] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 2nd edition, 2006.

[30] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[31] Charles Eric Leiserson, Ronald L Rivest, Thomas H Cormen, and Clifford Stein. *Introduction to algorithms*, volume 3. MIT press Cambridge, MA, USA, 1994.

[32] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[33] George M Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628, 2012.

[34] Boaz Sat, Ronen Hazan, Tova Fisher, Hanita Khaner, Gad Glaser, and Hanna Engelberg-Kulka. Programmed Cell Death in Escherichia coli: Some Antibiotics Can Trigger mazEFLethality. *Journal of Bacteriology*, March 2001. Publisher: American Society for Microbiology.

[35] Uri Moran, Rob Phillips, and Ron Milo. SnapShot: key numbers in biology. *Cell*, 141(7):1262–1262.e1, June 2010.

[36] Frederick R. Blattner, I. I. I. Guy Plunkett, Craig A. Bloch, Nicole T. Perna, Valerie Burland, Monica Riley, Julio Collado-Vides, Jeremy D. Glasner, Christopher K. Rode, George F. Mayhew, Jason Gregor, Nelson Wayne Davis, Heather A. Kirkpatrick,

Michael A. Goeden, Debra J. Rose, Bob Mau, and Ying Shao. The Complete Genome Sequence of Escherichia coli K-12. *Science*, September 1997. Publisher: American Association for the Advancement of Science.

[37] Julius Fredens, Kaihang Wang, Daniel de la Torre, Louise F. H. Funke, Wesley E. Robertson, Yonka Christova, Tiongsun Chia, Wolfgang H. Schmied, Daniel L. Dunkelmann, Václav Beránek, Chayasith Uttamapinant, Andres Gonzalez Llamazares, Thomas S. Elliott, and Jason W. Chin. Total synthesis of Escherichia coli with a recoded genome. *Nature*, 569(7757):514–518, May 2019. Publisher: Nature Publishing Group.

[38] Marius Welzel, Peter Michael Schwarz, Hannah F. Löchel, Tolganay Kabdullayeva, Sandra Clemens, Anke Becker, Bernd Freisleben, and Dominik Heider. DNA-Aeon provides flexible arithmetic coding for constraint adherence and error correction in DNA storage. *Nature Communications*, 14(1):628, February 2023.

[39] Abraham Lempel and Jacob Ziv. Compression of two-dimensional data. *IEEE transactions on information theory*, 32(1):2–8, 1986.

[40] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for lempel-ziv encoding. In *Algorithm Theory—SWAT'96: 5th Scandinavian Workshop on Algorithm Theory Reykjavík, Iceland, July 3–5, 1996 Proceedings 5*, pages 392–403. Springer, 1996.

[41] Daxing Wen and Chunqing Zhang. Universal multiplex pcr: a novel method of simultaneous amplification of multiple dna fragments. *Plant methods*, 8:1–9, 2012.

[42] Emily M LeProust, Bill J Peck, Konstantin Spirin, Heather Brummel McCuen, Bridget Moore, Eugeni Namsaraev, and Marvin H Caruthers. Synthesis of high-quality libraries of long (150mer) oligonucleotides by a novel depurination controlled process. *Nucleic acids research*, 38(8):2522–2540, 2010.

[43] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. Pear: a fast and accurate illumina paired-end read merger. *Bioinformatics*, 30(5):614–620, 2014.

[44] Jiasen Li, Mingkai Dong, Zhengwei Qi, and Haibing Guan. Siyuancodesequencing — sequencing data for towards density-optimal dna storage with practical bijective coding, 2026. DOI: 10.5281/zenodo.21357059.

[45] Jiasen Li, Mingkai Dong, Zhengwei Qi, and Haibing Guan. Siyuan code — towards density-optimal dna storage with practical bijective coding. Code Ocean, 2026. DOI: 10.24433/CO.4952245.v1.

Acknowledgements

This work was supported by the National Key R&D Program of China (2023YFC2705504), and Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University. This work was partially supported by SJTU Kunpeng&Ascend Center of Excellence. We appreciate Fei Wang for discussions on the basic knowledge of DNA synthesis and sequencing.

Funding

This work was partially supported by SJTU Kunpeng&Ascend Center of Excellence (to H.G.).

Author Contributions

J.L. designed Siyuan Code (including the mapping algorithm). J.L. wrote the code of the software program of Siyuan Code. J.L. conducted simulation and data analysis. J.L. conducted the sequencing data analysis. J.L. drafted the manuscript. J.L. contacted GenScript for DNA synthesis and sequencing service. J.L. conducted the theoretical proof. M.D. contributed to the literature review in the early stage of this work. H.G. supervised the study, named the algorithm Siyuan Code, reviewed the manuscript and provided revision suggestions. Z.Q. read the manuscript and provided revision suggestions. All authors read and approved the final manuscript.

Competing Interests

The authors declare no competing interests.

Tables

Table 1: Comparison among popular methods

	NGDIS [33] (Church et al.)	PHCLMIS [12] (Goldman et al.)	RCPDIS ECC [13] (Grass et al.)	Fountain Code [15] (Erlich et al.)	Yin-yang Code [16] (Ping et al.)	Aeon Code [38] (Welzel et al.)	DNAFormer [28] (Bar-Lev et al.)	Siyuan Code (This work)
Error correction strategy	No	Repetition	RS	Fountain	RS	Fountain+CRC	RS+DL	RS
Robustness against excessive errors	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Coding potential ($bits \cdot nt^{-1}$)	1	1.58	1.78	1.98	1.95	N/A	N/A	2
Net information density ($bits \cdot nt^{-1}$)	0.83	0.33	1.14	1.57	N/A	1.15	1.6	1.66
Physical density achieved (In vitro) ($EBytes \cdot g^{-1}$)	0.00128	0.00225	0.025	0.214	2.25	Not Given	Not Given	41
GC content (%) of sequences	2.5–100	22.5–82.5	12.5–100	40–60	40–60	40-60	45-55	42.9 - 57.6
Maximum homopolymer length (nt)	3	1	3	4	4	3	4	2
Customized maximized homopolymer length (nt)	≥ 1	$= 1$	≥ 3	≥ 4	≥ 4	≥ 3	=4	≥ 1

Figure Legends

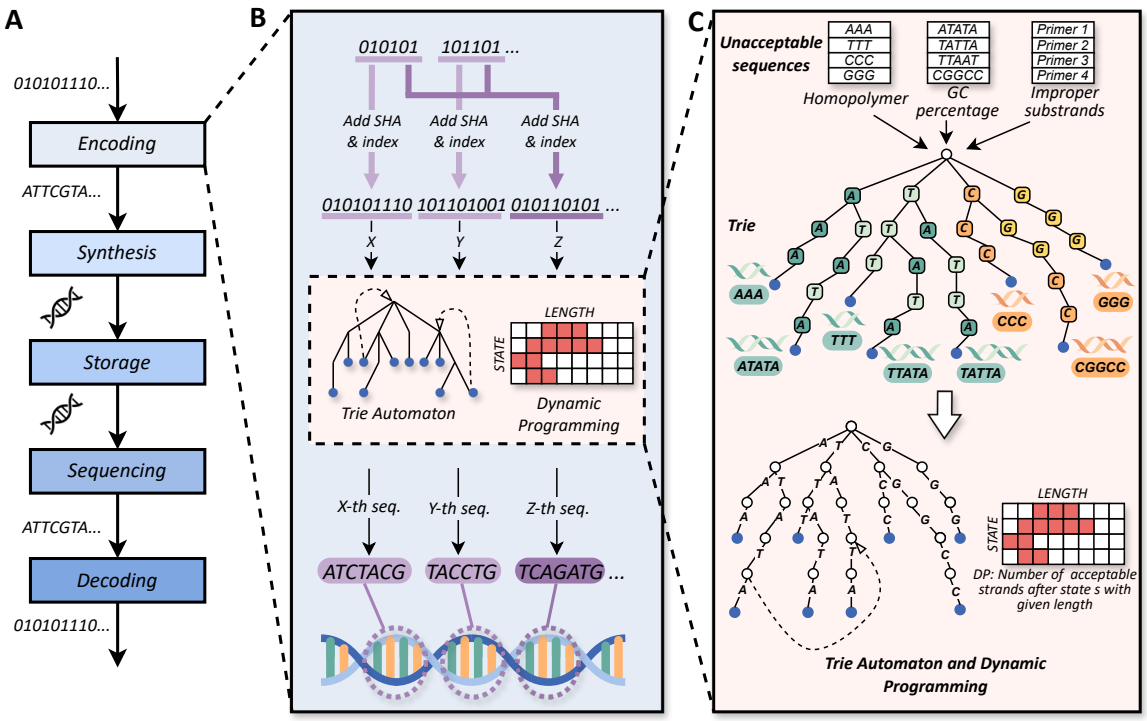


Figure 1: **Overview illustration of Siyuan Code.** a, Five stages of DNA storage and how Siyuan Code encodes binary data to DNA sequences. b, Encoding stage. c, The mathematical core of bijective coding.

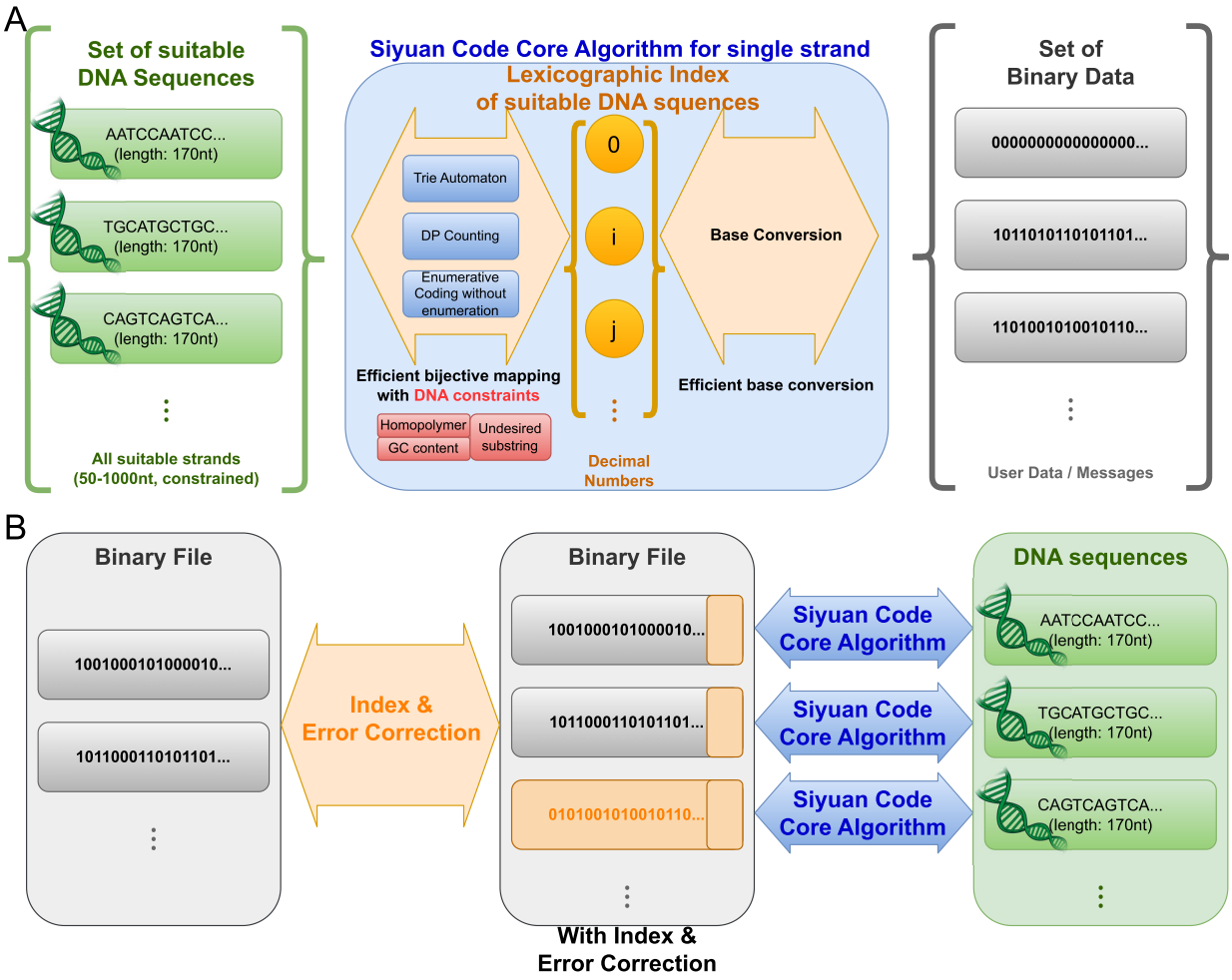


Figure 2: **Simplified Conceptual illustration for the core principle of Siyuan Code.** a, Siyuan Code Core Algorithm is a bijective mapping between the set of suitable DNA sequences and the set of binary data. The components are Trie Automaton, DP counting and counting-based enumeration-less bijective enumerative coding. b, Siyuan Code employs Siyuan Code Core Algorithm with error correction and index plugins to deal with file-to-DNA coding. Icons are in the Public Domain.

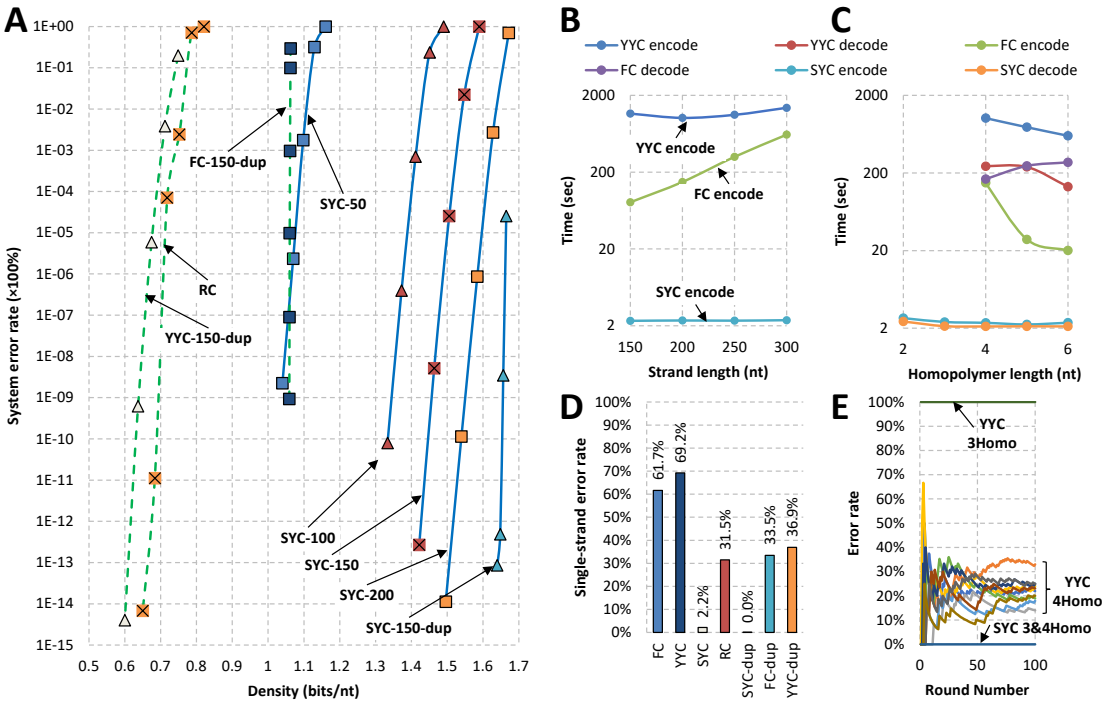


Figure 3: **In-silico Experiments. FC: Fountain Code; RC: Rotation Code; SYC: Siyuan Code; YYC: Yin-yang Code. The number after the code name is the DNA strand length, and the "-dup" suffix indicates duplication (copy number=5), i.e., replication is employed during sequencing; thus, SYC-150-dup stands for the Siyuan Code with 150 DNA strand length and duplication.** a, The relations between system error rate and the density for different coding methods. The bottom-right corner denotes the highest density with the least error rates. b, The encoding/decoding time with different DNA strand lengths when 4-Homopolymer is forbidden. c, The encoding/decoding time with different lengths of forbidden homopolymer with DNA strand length 200. d, The single-strand error rates of different methods. e, The failure rate when encoding small files (Wikipedia's self-introduction). The curves are the system failure rate with different random seeds under different number of rounds. In each curve, the rounds are the multiple tries with the same random seeds. All panels in this figure can be reproduced using the simulation code. The Docker environment is available on Code Ocean.

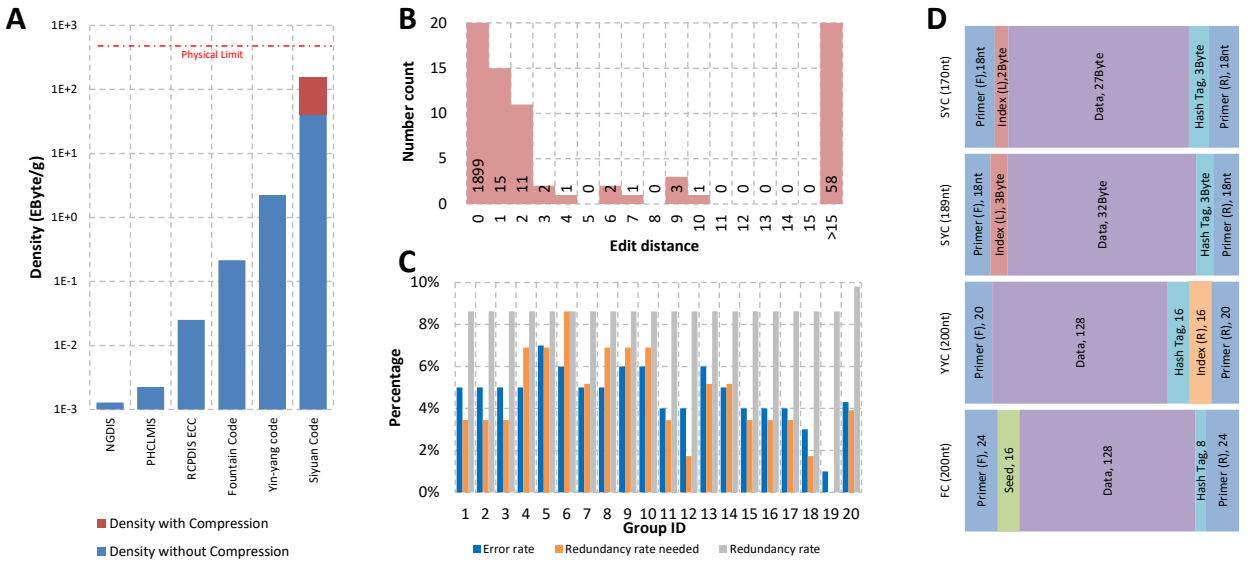


Figure 4: **In-vitro Experiments and sequence design.** a, The physical density of all the methods in comparison. b, The statistics of edit distance from the input strands to the closest sequenced strands. c, The error rate, the redundancy rate needed and the redundancy rate of each group during error correction after sequencing. Group: the group refers to the Reed-Solomon data block. Error rate: the ratio of errored strands in each group. Redundancy rate: the actual redundancy given in each group. Redundancy rate needed: the minimal proportion of extra strands needed in decoding correctly. d, The sequence design of oligo generated by Siyuan Code, YYC and Fountain code. Each rectangle represents a DNA strand. For all panels in this figure, the data pertaining to our work can be reproduced using the provided test code and sequencing data. Panel d is a schematic illustration and does not involve experimental data, though its proportions are accurate.

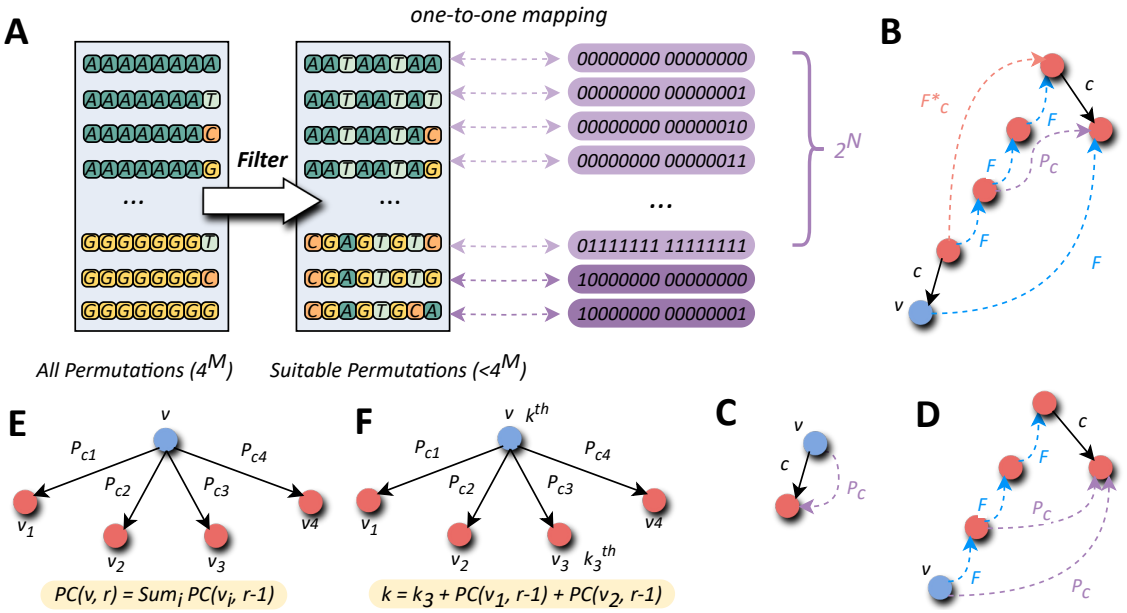


Figure 5: **The basic idea and its improvements with the data structure.** a, The strawman and the basic idea is that some suitable strands are picked from all the strands, and then mapped one-to-one to the integers from 1 to 2^N. b, When deriving the Fail Edge (the F function) of vertex v, the parent vertex is taken into consideration. Walk through the F edge from the parent vertex until the edge has a c child. Then the c child is the Fail of v. With the help of F_c^* or P_c, $O(1)$ time is needed to derive Fail Edge. The blue point is the vertex v. The red points are the relevant points. c, When deriving the Press Edge (the P_c function) of vertex v, if V has the c child, the P_c of the vertex is the c child of vertex v. d, When driving the Press Edge (the P_c function) of vertex v, if v does not have the c child, walk through Fail Edge until it has a c child. The P_c of v is the c child. With the help of F and P_c, the Press Edge can be efficiently derived. e, The example of PC value calculation. The number of suitable strands starting from vertex v can be the sum of the numbers from all its immediate successors. f, The principle of encoding and decoding. When the next DNA base from vertex v is unknown, calculate the numbers of its successors and see which successor contains the k^{th} suitable strand.

Supplementary Material: Towards Density-optimal DNA Storage with Practical Bijective Coding

Jiasen Li[1], Mingkai Dong[1], Zhengwei Qi[1], Haibing Guan[1*]

[1]School of Computer Science, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China, 200240, China

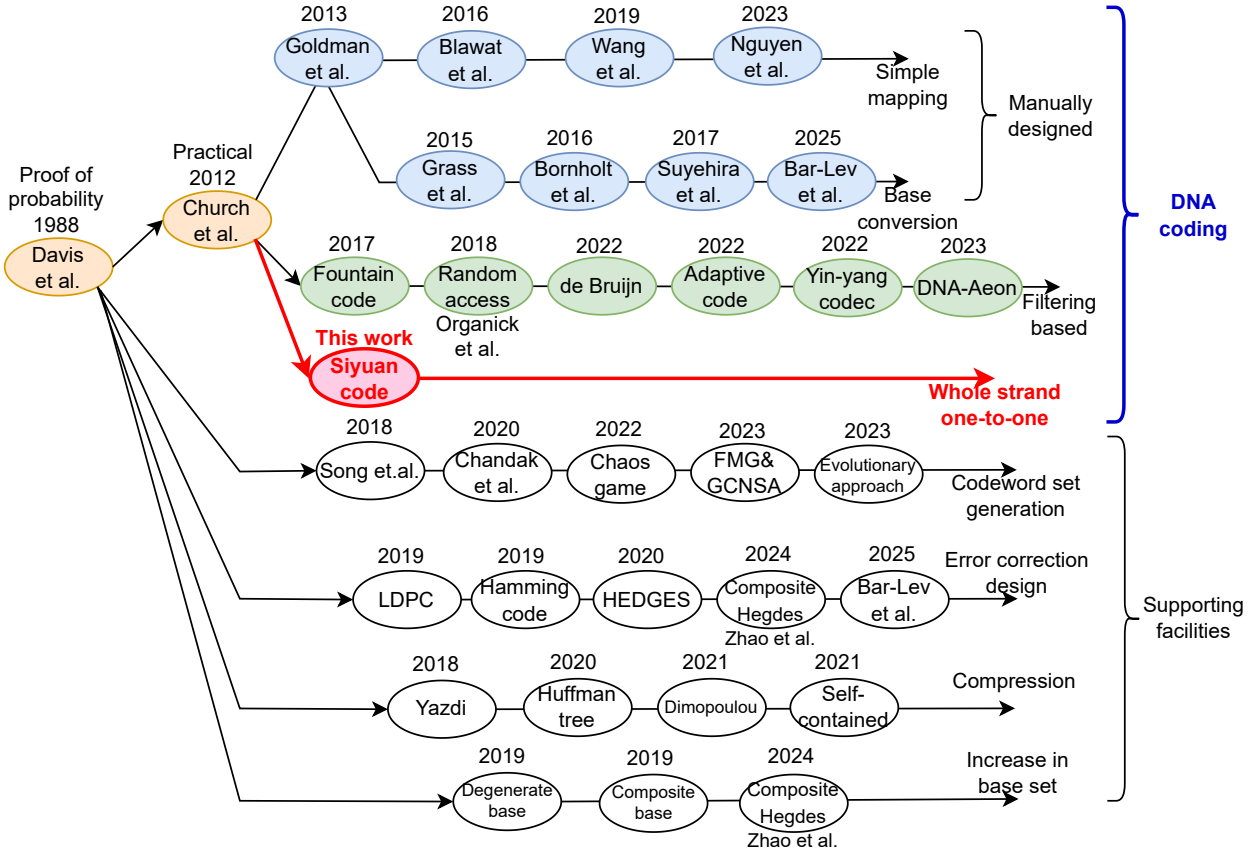
*hbguan@sjtu.edu.cn

A Existing Work of DNA Storage

After the possibility of storing information in DNA is proved in 1988 [1], a large number of attempts at DNA storage have been made as shown in Supplementary Figure 1. Many storage researches have been proposed to fit the storage with DNA. These researches can be classified into two big types, DNA coding and supporting facilities. DNA coding is to study how to encode and decode. Supporting facilities emphasize some techniques to help DNA coding design. Two main types of DNA coding are manually designed DNA coding and filtering-based DNA coding. One is based on manpower design and the other is based on random generating and filtering.

A.1 Manually Designed DNA Coding

Some early works on DNA storage restrict the coding manually to avoid unsuitable error-prone sequences. For example, Church's coding [2] is designed to map A/C to 0 and G/T to 1. As each bit can be presented by two kinds of nucleotides, it is easy to encode the data to



Supplementary Figure 1: The context map of the related work

avoid homopolymers and control the GC percentage. The obvious disadvantage of Church's coding is the low storage density (1 bit/nt), which makes it unsuitable for large-scale data storage considering the high cost of DNA synthesis.

Goldman's coding [3] is built on a rule that the meaning of each nucleotide depends on its previous nucleotide in the sequence, and no two consecutive nucleotides are the same. Such a rotating coding avoids homopolymers by design and matches the ternary since three nucleotides are available for each position in the sequence. Consequently, the encoding procedure of Goldman's coding, or rotation code, converts binary strings to ternary and encodes to nucleotide sequences via the rotated coding table. Though it has a higher density than Church's coding, Goldman's coding is still over-conservative and several available sequence patterns are excluded by the fixed coding rule.

A.1.1 Base conversion

A type of manually designed DNA coding is based on base conversion. Taking Goldman et al.'s effort [3] as an example, Goldman's method first converts the binary information into ternary and then converts each ternary bit into a base. Such manually designed methods convert the information into k-base form and give a mapping from $\{0 \cdots k-1\}$ to a small segment of strands.

Grass et al. [4] maps 47 sequences of length 3 to integers from 0 to 46. The coding rule only avoids four-homopolymers which further utilizes the coding space. Bornholt et al. [5] enables compression with a Base 3 Huffman code when converting binary strings to ternary sequences. Suyehira et al. [6] maps 39 sequences of length 3 to integers from 0 to 15. The coding rule is not fixed because each hexadecimal digit corresponds to at least 2 different sequences of length 3.

Bar-Lev et al. [7] employed a deep learning approach to correct insertion and deletion errors, which complemented Reed-Solomon (RS) coding used for substitution error correction. They adopted a partitioned codeword set generation strategy, where a predefined binary-to-DNA block lookup table was constructed through enumeration. This ensured that the minimum edit distance between any possible block codewords was no less than 3, thereby enabling error correction using conventional information-theoretic coding methods. Their error avoidance can be regarded as variant of the approach by Grass et al. [4]. Their block lookup table can be regarded as a manually designed method, because the error pattern is not customizable like filtering-based methods. Manually designed methods need manual work in lookup table generating. Also, it is impossible for their method to fully account for error avoidance between two adjacent blocks—either a conservative approach leads to under-utilization of low-error strands, or an aggressive strategy results in the use of error-prone strands. Although longer block lengths can help mitigate imperfections and overhead between adjacent blocks, they also introduce significant storage costs, thus preventing the use of excessively long blocks. That is one of the reasons why their net information density without constraints is still not large enough.

A base conversion with larger blocks in base conversion utilizes more coding space. How-

ever, the designing of the big blocks with manual design is impractical and manpower wasting. Small blocks either waste too much coding space or accidentally include unsuitable patterns because the suitable pattern is not simply the replication of k-base digits. In fact, the suitability depends on adjacent blocks. Also, a base conversion method cannot avoid specific patterns unless it uses unfixed mapping. When a base conversion uses unfixed mapping, it wastes at least half of the coding space and at least one bit for each small block in base conversion.

A.1.2 Simple mapping

A type of manually designed DNA coding does not do the base conversion. Take Church's coding [2] as an example, Church's method simply maps each base to a binary bit. Such manually designed methods are more flexible and have nearly nothing in common except that they are manually designed.

Blawat et al. [8] maps 3 adjacent pairs of bits to 3 bases and maps the following pair of bits to 2 bases and avoids 7homopolymer. The mapping is not like base conversion because it does not map the same length of binary data to the same length of strand. Also, this method provides options for users to select the coding because binary information corresponds to multiple choices of strands in order to avoid unsuitable patterns. Wang et al. [9] maps the binary data to the strands with Markov process finite state transition. This method avoids homopolymers of length 3 with base A. Nguyen et al. [10] refers the constraint of the count of base in order to avoid secondary structure. Any substrand is not the reverse of another substrand.

However, simple mapping methods are based on delicate design and all their avoided pattern are all by manpower-wasting manual design and their designs are not accurate enough to preclude all unsuitable patterns while keeping all suitable patterns. Their inaccuracy can lead to a low encoding density.

A.1.3 Summary of Manually Designed DNA Coding

In summary, these manually designed coding methods leverage manually decided rules to avoid unsuitable sequences. These rules are usually easy to implement and fast to execute.

However, as the fixed rules usually preclude suitable sequences by accident, these methods cannot leverage all sequences suitable for DNA storage, resulting in low encoding density. Also, the fixed rules sometimes include some unsuitable sequences by accident when the rules are based on radical strategies, resulting in high redundancy in error correction and low density to fix the high error rate. Besides, it's difficult, if not impossible, for these methods to adjust customized rules, for example, the length of allowed homopolymers. To sum up, all the manually designed coding methods have worse density values and other properties than Yin-yang code [11] mentioned in Table 1. Only the state-of-the-art methods are ineluctable in comparison.

A.2 Filtering-based DNA Coding

Another category of DNA coding methods generates DNA sequences and filters these sequences to avoid unsuitable sequences for DNA storage. The sequence generation method is usually based on randomization, so new sequences can be generated for the same data when the last generated sequence is rejected.

Fountain code [12] is an advanced DNA coding method that relies on filtering. It randomly generates a pseudorandom matrix to encode data and filters the generated sequences for DNA storage. The filtering considers both homopolymers and GC percentage, i.e., sequences with unacceptably long homopolymers or biased GC content percentages are rejected. Fountain code is friendly to error correction since a portion out of all sequences is sufficient to make a successful decode. Thus, even if some generated sequences are filtered out, the remaining sequences are still sufficient for decoding. At the same time, the randomizations increase the computation complexity since the probability of generating enough suitable sequences might be too small to encode in a reasonable time.

Yin-yang code [11] also leverages randomization to generate sequence candidates and uses filters to select suitable sequences for DNA storage. Yin-yang code randomly chooses slice pairs to generate pseudorandom sequences via yang rules and yin rules to store information. Filters avoid 5-homopolymers and guarantee that the GC content percentage is within 40%–60% with appropriate free energy. The filtering rules are slightly looser than previous work, which contributes to a higher encoding density but requires a large sequence depth

($\geq$100$\times$) to tolerate single-strand errors. Yin-yang code is the first filtering-based codec that nearly eliminates the additional redundancy required by Fountain codes, which need extra redundancy to ensure successful decoding even without considering error correction.

Organick et al. [13] adds the random access function to the RS code and filters undesired patterns of strands and primers. De Bruijn [14] uses fountain code as the outer code and uses Bruijn graph to generate the inner code. However, the method brings together the disadvantages of both manually designed methods and filtering-based methods. Adaptive code [15] uses fountain code to encode the data payload with adaptive parameters. Aeon [16] uses arithmetic and fountain code. However, Aeon's density has not exceeded the density of Yin-yang code and the state-of-the-art filtering-based method in density is still Yin-yang code.

Also, Aeon [16] has some weaknesses. First, as a code aiming for robustness, Aeon does not provide a satisfying density value. Second, Aeon is not robust enough. On one hand, Aeon's arithmetic code can not deal with undesired motifs across the reverse primer. On the other hand, Aeon is so weak in robustness that a simple undesired motif set $\{AA, AT, AC, AG\}$ can ruin the arithmetic code by an infinite loop. This is an example of Aeon's local optimum leading to rather bad global results. By contrast, our robust method can work with malicious settings and complex settings because of our global optimality of our one-to-one mapping. Third, the arithmetic code uses numeral systems with variable bases. Each base can be a ternary or quaternary digit. As a result, the information content of each strand is not a fixed amount. The information source can not be simply segmented. A conservative binary length for each strand is wasteful when an aggressive binary length can lead to system crash.

Ceze [17] established a system to integrate DNA storage and to overview the process. David Buterex et al. [18] integrates efficient Deep Learning inference with the filtering step. However, the post-filtering DNA codings can not balance the efficiency and the constraint. For those methods, a large number and a large proportion of unsuitable strands should be randomly generated and filtered by the filtering step. With strict constraints (e.g., disabling 3-homopolymer), the running time of the post-filtering DNA codings is actually exponentially large by the strand length, which is unscalable and not practical. With loss constraints (e.g.,

disabling 6-homopolymer), the error rates of the accepted strands are too high, causing a low information density and high redundancy and duplication in DNA storage to reproduce the stored information.

A.2.1 Summary of Filtering-based DNA Coding

In summary, filtering-based DNA coding introduces randomness in the encoding and relies on filters to avoid sequences unsuitable for DNA storage. Filtering-based DNA coding randomly generates sequences and removes unsuitable ones until the remaining ones can be decoded. The filters present high compatibility and flexibility since updating the filtering rules is usually trivial. However, the filters strongly affect the single-strand error rates and encoding time. Strict filtering rules reduce the single-strand error rate, but more sequences should be generated and fed to the filter, which can significantly prolong the encoding time and cause low efficiency. The low efficiency is estimated in §2.2. A bad and unstable case (as shown in Figure 3e) is that nearly all the generated sequences are unsuitable and the regenerating either becomes an endless loop or fails. Loose filtering, on the other hand, incurs high single-strand error rates (as shown in Figure 3d), and thus more copies of a sequence need to be synthesized, lowering the density (as shown in Figure 3a and Figure 4a), and a higher sequencing depth is required, which increases the costs. Thus, for filtering-based DNA coding, it's difficult to achieve high density with fast execution time and the filtering-based methods are unstable and high in single-strand error rates.

A.3 Supporting facilities

Supporting facilities are not complete DNA coding systems. Supporting facilities emphasize some techniques to help DNA coding design as a part of the coding system. Some supporting facilities can be applied to existing DNA coding methods.

A.3.1 DNA Codeword Set Generation

The Codeword Set Generation generates a codeword set based on the enumeration of possible suitable strands. It requires less manpower than manually designing a codeword set in

manually designed methods and can potentially achieve larger capacity than manually designed sets. It has a very high non-polynomial time complexity and the codeword set itself is non-polynomial. It is too time-consuming to encode a whole strand of practical length as a single codeword. Most works of codeword set generation have a simulation result with a strand length of less than 30. Encoding parts of the strands one by one cannot solve the unsuitable patterns across the parts. Almost all the methods based on enumerating are not examined by in-vivo or in-vitro experiments. Song et al. [19], Chandak et al. [20], Chaos game [21], Cao et al. [22, 23] and evolutionary approach all try to generate the enormous codeword set and are not examined with in-vitro experiments.

A.3.2 Error correction

The error correction here does not refer to the coding that has an error correction function. In fact, it refers to the design of the error correction part that does not generate strands with specific patterns and strongly depends on error correction. Unfortunately, such methods are based on traditional error correction methods that do not use the property of the strands in error correction. LDPC code [24] uses binary parity check to get perform error correction. Quaternary Hamming code [25] can also be used for error correction with replacement error. HEDGES [26] uses greedy exhaustive search to get the correct hash value back and removes the error.

Zhao et al. [27] uses HEDGES to present the latest error correction method in DNA storage. They synthesized DNA strands and successfully sequenced and decoded them using nanopore sequencing (NPS). Their method can handle indel error rates of up to 15.9% and substitution rates of up to 7.8%. Their method utilizes four composite bases and four natural bases, namely M, K, S, W (composite bases) and A, C, G, T (natural bases). Each composite base represents a mixture of two natural bases: M (a mixture of A and C), K (a mixture of G and T), S (a mixture of G and C), and W (a mixture of A and T). The composite base is not represented by two consecutive nucleotides, but is determined by their relative frequency across multiple strands. For instance, if 50% of the strands contain an A at a specific position and the other 50% contain a C, that position is the composite base M. Besides the composite bases' consensus mechanism, their method also uses RS code and

5mer anchors in each strand to help error correction. However, it should be noted that while the random hash function employed in their method is conceptually elegant, it essentially serves as a randomized mapping from binary pairs to nucleotide bases, which may be equal to other random functions. Additionally, their approach requires an eight-fold increase in the number of DNA strands to emulate the defined virtual bases, resulting in significant sequence redundancy. Although the method demonstrates strong robustness under high-error-rate sequencing conditions, future improvements could focus on adopting alternative error avoidance strategies to enhance the net information density (bits/nt). Furthermore, the identification of virtual bases relies not on mere presence/absence detection but on the relative abundance ratios of the underlying nucleotides, which introduces a dependency on quantitative measurement accuracy and leaves room for further robustness enhancement by using the natural bases. It can give an error correction method for future attempts.

In the research aimed at enhancing the robustness of DNA storage systems, the DNA StairLoop [28] scheme proposed a novel error-tolerant coding framework for high-error-rate synthesis scenarios. Its core contribution lies in the design of a staircase inter-leaver structure and an iterative soft-decision decoding algorithm, which significantly improves the system's tolerance to random insertion, deletion, substitution errors, and high sequence dropout rates, enabling reliable data recovery in specific high-noise environments. Several key aspects of this scheme must be clarified. First, its design objective is inherently high error tolerance, rather than the active avoidance of error-prone sequence motifs (such as long homopolymers or extreme GC content) at the sequence level. Second, to achieve strong error tolerance and parallel decoding, the scheme employs low-rate codes and redundant structures; the logical storage density, calculated from its reported data, is approximately 0.42 bit/nt, indicating a relatively low storage density. Third, its superior error correction comes with significant computational overhead due to iterative decoding. The scheme explicitly addresses this by employing a highly parallel decoding architecture that scales to thousands of nodes to mitigate the time cost, indicating a trade-off between decoding capability and computational resources. Finally, its performance evaluation is primarily conducted under the premise of "equal error rates" highlighting its error correction advantage. However, in practical synthesis, coding schemes that directly circumvent error-prone motifs through biochemical

constraints (e.g., Fountain codes) often achieve lower initial error rates, potentially offering greater advantages at the system level. As the core objective of our work is high-density storage and error-avoidance coding - a direction not optimized for in StairLoop's design - its performance in high density and active error avoidance is not its intended strength. To ensure a fair comparison and avoid presenting it unfavorably in dimensions outside its design focus, and given the lack of officially optimized density parameters for StairLoop, our subsequent in-depth evaluations of density and error-avoidance capability will not include it in direct comparisons. This ensures that the assessment remains focused on schemes with similar objectives, enhancing comparability and practical relevance.

A.3.3 Compression in DNA Coding

Compression is necessary for non-lossy DNA storage because the file should be as small as possible to have a higher information density. However, the compression rate is highly related to the property of the compressed data. It is unfair to calculate the density with compression, especially if a zip bomb is stored as a part of the compressed file. For non-lossy DNA storage, the compression can be decoupled from the coding. Thus, the coding density should be compared without compression and the compression should be compared with traditional compression. Yazdi et al. [29, 30], Huffman tree [31], Dimopoulou et al. [32] and self-contained code [33] are all attempts at putting compression into the coding.

A.3.4 Increase in DNA Storage Base Set

Also, some researches expand the base in coding to achieve higher density [34, 35]. A lot of methods, especially the filtering-based methods can work well with the expansion in the base. However, manually designed methods should be redesigned with manpower in order to work with the larger set of bases.

A.3.5 Hardware

The development of DNA-based storage systems is intrinsically linked to advances in the underlying biotechnology. Current hardware platforms primarily rely on in situsynthesis on high-density oligonucleotide microchips for data writing and high-throughput next-

generation sequencing (NGS) platforms for data reading [36]. The specific limitations and error profiles of these synthesis and sequencing technologies directly dictate the requirements for robust encoding and decoding schemes.

Li et al. introduce DNA-DISK [37], an automated, end-to-end platform for DNA data storage that integrates enzymatic DNA synthesis, thermo-responsive encapsulation, and pyrosequencing on a single digital microfluidics (DMF) chip. This system achieves base-by-base writing with high fidelity using an engineered terminal deoxynucleotidyl transferase, solidifies DNA-in-agarose for on-chip storage, and enables accurate in-situ reading. By miniaturizing and automating the entire process within a benchtop device, DNA-DISK significantly reduces operational complexity, latency, and manual intervention. This work represents a major step towards practical DNA-based information storage, demonstrating a scalable, integrated, and user-accessible approach that could fundamentally accelerate the development of DNA as a viable medium for long-term, high-density data archiving.The platform's distinct error profile, predominantly deletions, necessitates the development of adaptive coding algorithms with customizable parameters. Such integrated and accessible devices are crucial for practical large-scale testing and commercialization of DNA storage, moving algorithms from theory to real-world application.

While high-throughput chip synthesis exhibits a higher intrinsic error rate compared to column-based or enzymatic methods, its unparalleled parallelism and low cost make it the most viable technology for storing exabyte- and zettabyte-scale datasets [36]. Therefore, a modern DNA storage encoding algorithm should be robust against the characteristic error profiles of chip synthesis, employing error-avoidance or error-correction schemes to compensate for physical medium imperfections, rather than relying on high-fidelity synthesis. Ideally, such an algorithm would also adapt to varying error patterns. For data retrieval, the decoding algorithm should be compatible with next-generation sequencing (NGS), which serves as the high-throughput counterpart to chip synthesis, enabling parallel reading of the immense oligonucleotide pools required for large-scale data recovery. The algorithm should also be optimized to handle substitution errors, which are more prevalent in NGS technologies.

B Mathematics and Algorithms in DNA encoding

B.1 Formal definition of the Problem

The problem is to find a DNA strand encoding to present the information. An encoding method with high density (few bases), low error rate, high stability and high encoding speed at the same time is optimal. With non-zero error rate for any DNA strand, a practical encoding should have error correction strategy to acquire a close-to-zero error rate with high redundancy. It is possible to sacrifice one of high density and low error rate to acquire another, so both of them should be taken into consideration.

B.1.1 Formal definition of an encoding

Too-long DNA strand is much more expensive and can lead to extremely low accuracy, so it is reasonable and necessary to assume that the information to be encoded is cut into several parts with length n. Let the set of information character be $\mathcal{A} = \{0, 1\}$. Let the set of DNA bases be $\mathcal{B} = \{A, T, C, G\}$. Any possible encoding should be a mapping from a string of $\mathcal{A}$ of length n to a string of $\mathcal{B}$ (a DNA strand can be regarded as a string of $\mathcal{B}$), i.e., $f : \mathcal{A}^n \to \mathcal{B}^*$. Let $\text{len}(\cdot) : \mathcal{A}^* \cup \mathcal{B}^* \to \mathbb{Z}^+$ be the function of the length of a string.

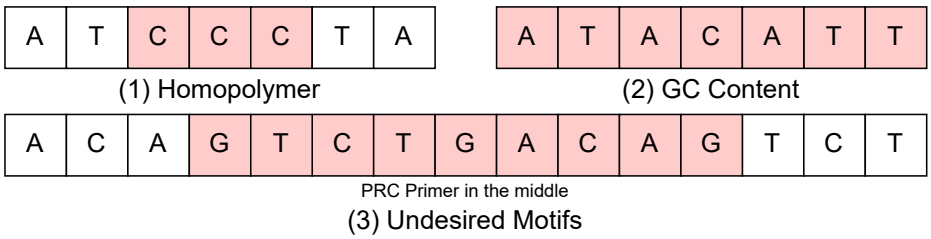
B.1.2 Definition of suitable output strands

According to MESA [38], there are 4 main kinds of errors. One of them is the secondary structure named Kmer, which is less than 10^{-4} and negligible. As a result, we can define a suitable strand to be a strand without the other 3 patterns, as shown in Supplementary Figure 2: 1) homopolymer longer than k. 2) GC content rate not between $\text{GC}_{min} \times 100\%$ and $\text{GC}_{max} \times 100\%$. 3) undesired motifs.

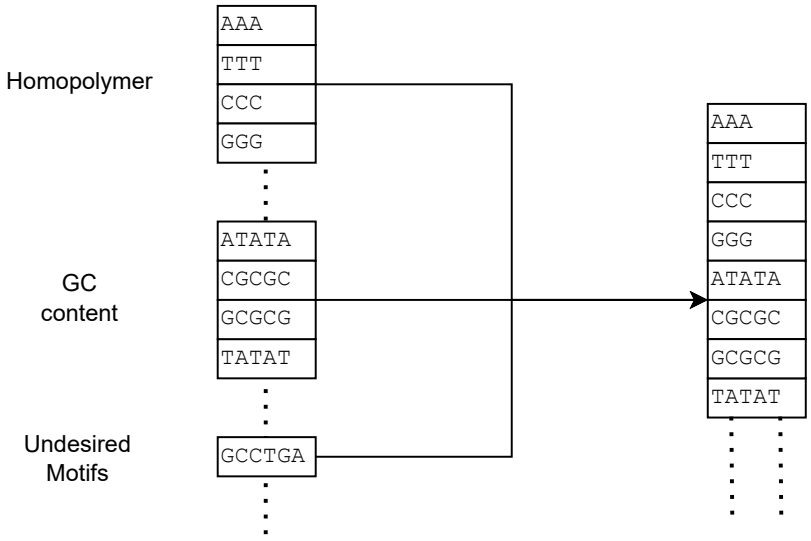
We define a DNA sequence to be suitable if:

1. No homopolymer longer than k;
2. GC rate between GC_{min} and GC_{max};
3. No undesired motifs in the sequence.

For a fixed-length sequence, we can build additional undesired motifs to present rules (1) and (2). For (1), we add a sequence of $k+1$ repeating bases for each character in $\mathcal{B}$, e.g., "AAAA", "TTTT". For (2), we deal with global GC content and local GC content separately. With the help of the law of big number, the global GC content rate can be close to it average. Dealing with the local GC content rate, it is easy to enumerate some bad short motifs with unacceptable local GC content rate like "ATATA" or "GCGCC" and mark them as undesired motifs, as shown in Supplementary Figure 3.

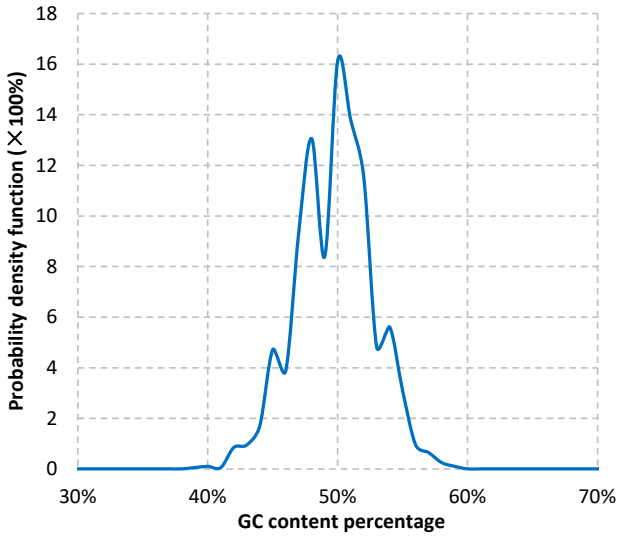


Supplementary Figure 2: Error patterns



Supplementary Figure 3: Building additional undesired motifs

In the case of Siyuan Code, the definition of suitable string is a string without a substring occurring in a given "unsuitable substring" set. S_k indicates a set of suitable strands of length k and $\mathcal{S}_k$ indicates a set of suitable strands of length less than or equal to k.



Supplementary Figure 4: The probability distribution of GC content percentage.

B.1.3 Rareness of global GC content error

The GC content percentage is a crucial metric for assessing the viability of DNA coding strategies. Both Fountain Code and Yin-yang Code aim to maintain the GC content within the range of 40% to 60%. Supplementary Figure 4 illustrates the GC content distribution for strands in silico and in vitro experiments encoded by Siyuan Code. The x-axis indicates the GC content percentage, while the y-axis represents the probability density function for strands with a given GC content percentage. The data shows that 99% of the strands have a GC content ranging from 42.9% to 57.6%, demonstrating that the law of large numbers is effectively applied in Siyuan Code. Importantly, deviations in GC content percentage have not impacted the decoding accuracy of Siyuan Code in experimental observations. In fact, the GC content percentage maintained by the law of large numbers is at least not worse than the competitors in Table 1.

To theoretically prove that the GC content error is low in a random strand because of the *law of large numbers* [39], we give an approximation of the distribution of the global GC content rate to prove the statement and to know the order of magnitude of global GC content error rate.

Assumption. We approximately assume that each strand is uniformly distributed be-

cause the unsuitable set is too large to enumerate in order to get the precise result. The approximate assumption is reasonable because the distribution of GC content rate does not too much depend on the unsuitable set (if undesired motifs are symmetric with respect to AT and CG) and we only need a rough estimation in order to show that the GC content error is rare.

Theorem B.1. *Rareness of global GC content error. Let $s \in \mathcal{B}^n$ be sample from $\mathcal{B}^n$. Let $s = (s_1, s_2, \cdots, s_n)$. Let E_i be the event that $s_i \in \{C, G\}$. Let X_i be 1 if E_i and 0 if not. Let $X = \frac{X_1+X_2+\cdots+X_n}{n}$. Then, the variance of X is $\frac{1}{4n}$.*

Note that X is the GC content rate. By the way, the GC content error is extreme if $X \in (0.35, 0.65)$.

Proof. The following is to prove that X far from 0.5 is not common. The expectation of X can be calculated as followed:

$$\begin{aligned} E(X) &= \frac{E(X_1)+(X_2)+\cdots+(X_n)}{n} \\ &= \frac{n \cdot E(X_i)}{n} \\ &= \frac{1}{2} \end{aligned} \tag{7}$$

According to the Independent Identically Distribution assumption of the bases in the strands, the variance of X can be calculated as followed:

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{Var}(\frac{X_1+X_2+\cdots+X_n}{n}) \\ &= \mathrm{Var}(\frac{X_1}{n}) + \mathrm{Var}(\frac{X_1}{n}) + \cdots + \mathrm{Var}(\frac{X_1}{n}) \\ &= \frac{n}{n^2}\mathrm{Var}(X_i) \\ &= \frac{1}{4n} \end{aligned} \tag{8}$$

□

Here the binomial distribution can be approximately treated as the normal distribution. Let ϵ be normal distribution ($\mu = 0, \sigma = 1$), and its cumulative density function be CDF($\cdot$).

$$
\begin{aligned}
P(|X-0.5|>c) &\approx P(|\epsilon|>c\cdot\sqrt{4n}) \\
&= 2P(\epsilon<-2c\cdot\sqrt{n}) \\
&= 2\cdot\mathrm{CDF}(-2c\cdot\sqrt{n})
\end{aligned} \tag{9}
$$

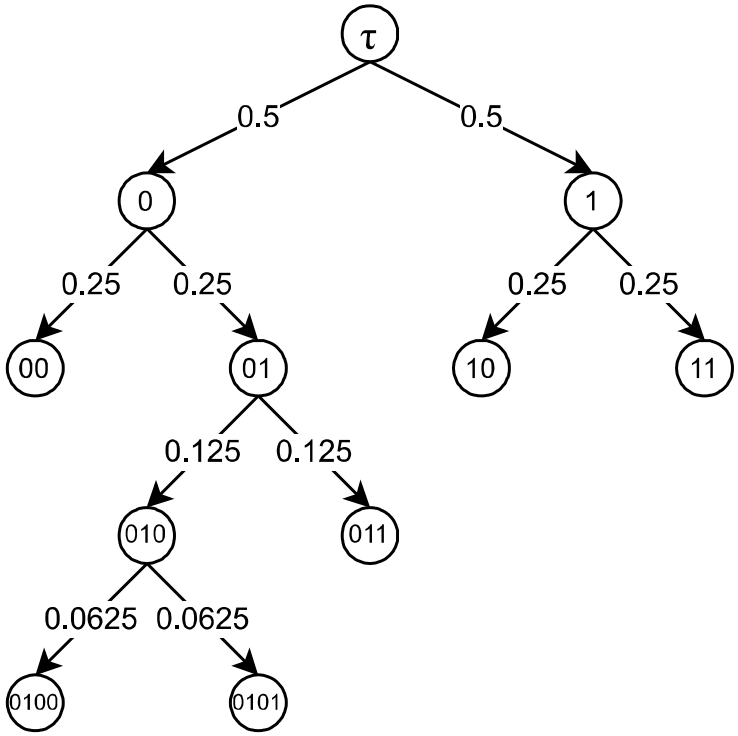
Let $n=100$, and $P(|X-0.5|>0.15)\approx 0.0027$. The error less than 1 percent can easily be covered by error correction with very low cost.

B.1.4 Uniform distribution of input binary

This section is to explain why the assumption of uniform input binary is reasonable.

If an input string has non-uniform probability distribution, it could be compressed using traditional compression methods in information theory like Huffman Coding.

An ideal Huffman Coding Tree should have equal weights on both sons of a node, indicating that probability of each bit after compression is independently, identically and uniformly distributed. Supplementary Figure 5 shows a Huffman coding tree. The leaves are the characters or the words to be compressed. The leaves can be considered a sub-tree with one



Supplementary Figure 5: An ideal Huffman Tree

vertex. Huffman Coding Tree of size n is generated by merging the smallest 2 sub-trees with the lowest probability and mark the new sub-tree probability as the sum of the 2 sub-trees. When all sub-trees have sons with equal probabilities, the optimal Huffman code achieves the best coding which is equal to the original coding.

After optimal compression, each bit in the compressed file uniformly independently identically distributed. The proof can be developed easily as follows. Assume that the probability of all possible input is known and taken into statistics. Let $hc_{(\mathcal{A}^n,p)}(\cdot)$ be the global Huffman coding on $\mathcal{A}^n$ and $p : \mathcal{A}^n \to \mathbb{R}^+ \cup \{0\}$ is the probability distribution of the input. According to the optimality of Huffman code, it has that:

$$\mathbf{E}_{X\sim(\mathcal{A}^n,p)}(\text{len}(hc_{(\mathcal{A}^n,p)}(X))) \leq H_{X\sim(\mathcal{A}^n,p)}(X) + 1 = \mathbf{E}_{X\sim(\mathcal{A}^n,p)}(-\log_2(p(X))) + 1 \tag{10}$$

According to Jensen's inequality,

$$\mathbf{E}_{X\sim(\mathcal{A}^n,p)}(-\log_2(p(X))) = -\sum_{x\in\mathcal{A}^n} p(x)\log(p(x)) \leq -\sum_{x\in\mathcal{A}^n} \frac{\log_2(1/|\mathcal{A}^n|)}{|\mathcal{A}^n|} = n = \text{len}(x \in \mathcal{A}^n) \tag{11}$$

and the equality holds when $\forall x \in \mathcal{A}^n$, $p(x) = \frac{1}{|\mathcal{A}^n|}$. To sum up, $\mathbf{E}_{X\sim(\mathcal{A}^n,p)}(\text{len}(hc_{(\mathcal{A}^n,p)}(X))) \leq \text{len}(x) + 1$. In other words, if X is far from uniformly independently identically distributed after compression, it is clear that adding another Huffman coding can further drop the expected length of the compressed file, indicating that the original compression is not optimal.

B.1.5 Definition of the density of an encoding

For a given mapping f, it is easy to define the expected length $\mathbf{E}(\text{len}(f)) = \sum_x \text{len}(f(x))P(x)$. $P(x)$ is uniform and $\text{len}(f(x))$ could be a variable. An optimal solution should be dense. If there is a suitable input not in the set of possible output $y_0 \notin f(\mathcal{A}^n)$ such that $\text{len}(y_0) < \max_x \text{len}(f(x))$ then there is a shorter encoding mapping some x to y_0.

The density should be defined as follows, and an optimal encoding should have a highest density.

$$d : f \mapsto \frac{\mathbf{E}_x(\text{len}(x))}{\mathbf{E}_x(\text{len}(f(x)))} \tag{12}$$

B.2 Siyuan Code Strategy

B.2.1 Optimal error avoidance DNA code definitions and settings

Note that this section is about the theoretical settings, but not the parameter settings.

A code can be written as $T : \mathcal{A}^n \to \mathcal{B}^*$. Here, $T(\mathcal{A}^n)$ should be a subset of the suitable strands. It is reasonable that the message has been compressed before coding, so it is assumed that each bit follows uniform distribution.

$$\forall x \in \mathcal{A}^n, Pr(x) = \frac{1}{2^n} \tag{13}$$

The expected density of an encoding $f : \mathcal{A}^n \to \mathcal{B}^*$ is defined as:

$$d : f \mapsto \frac{\mathbf{E}_x(\text{len}(x))}{\mathbf{E}_x(\text{len}(f(x)))} \tag{14}$$

$$d(T) = \frac{n}{\mathbf{E}_x(\text{len}(T(x)))} \leq 2 \tag{15}$$

Here, it is obvious that the input length can be the same because input can be cut into substrings with the same length.

Let the suitable strand set of length i be S_i. It is obvious that $\lim_{i\to\infty} \frac{|S_{i+1}|}{|S_i|} \in (2, 4)$. Let $T'_{k,n}$ be an one-to-one mapping from $\mathcal{A}^n$ to the suitable strand set $\mathcal{S}_k = \bigcup_{i=1}^{k} S_i$ of length less than or equal to k. According to Stolz Theorem, regard n as a function of k and the density converges to:

$$\lim_{k\to\infty} d(T'_{k,n}) = \lim_{k\to\infty} \frac{\mathbf{E}_x(\text{len}(x))}{\mathbf{E}_x(\text{len}(T'_{k,n}(x)))} = \lim_{k\to\infty} \frac{\sum_x \text{len}(x)}{\sum_x \text{len}(T'_{k,n}(x))} = \lim_{k\to\infty} \frac{n}{k} \tag{16}$$

Proof of $T'_{k,n}$ to be optimal can be found in the latter section. Let $T_{k,n}$ be an one-to-one mapping from $\mathcal{A}^n$ to the suitable strand set S_k of length k.

In the case of an optimal encoding $T'_{k,n}$, the given set of strand should represent as much information as possible, so let $n(k)$ be a function of k and n should be as large as possible:

$$|\mathcal{A}^n| = 2^n \leq |\mathcal{S}_k| = \sum_{i=1}^{k} |S_i| \leq |\mathcal{A}^{n+1}| = 2^{n+1} \tag{17}$$

Lemma B.2. *Encoding the same length n of information, it is possible to use $T_{k+1,n}$ instead of $T'_{k,n}$, because the code set of $T_{k+1,n}$ is larger than $T'_{k,n}$:*

$$|\mathcal{S}_k| = \sum_{i=1}^{k} |S_i| \leq |S_{k+1}| \cdot (\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots) < |S_{k+1}| \tag{18}$$

To be convenient, let T'_k be $T'_{k,n}$ with maximized length n and let T_k be $T_{k,n}$ with maximized length n. That is to say that make n as big as possible and name $T'_{k,n}$ as T'_k. Do the same to T.

Theorem B.3. $\lim_{k\to\infty} d(T'_k) - \lim_{k\to\infty} d(T_k) = 0$ *(An encoding with uniform code length can do as good as an optimal encoding with non-uniform code length.)*

Proof. The density gap from T to T' can be estimated as followed:

$$\text{LHS} = \lim_{k\to\infty} d(T'_k) - d(T_k) = \lim_{k\to\infty} d(T'_{k,n}) - d(T_{k+1,n}) = \lim_{k\to\infty} (\frac{n}{k} - \frac{n}{k+1}) \tag{19}$$

$$= \lim_{k\to\infty} \frac{n}{k(k+1)} = \lim_{k\to\infty} d(T') \cdot \lim_{k\to\infty} \frac{1}{k+1} = 0 \tag{20}$$

□

As a result, an encoding with uniform code length can do as good as an optimal encoding with non-uniform code length. Here, in this paper, we represent a special case of the Siyuan Code making the corresponding between the suitable strands set S_k with strand length k and the set of input data $\mathcal{A}^n$.

B.2.2 Definition: Siyuan Code using lexicographic order

The set S_k (or $\mathcal{S}_k$) is sorted with lexicographic order. Regard $\mathcal{A}^n$ as a set of non-negative integer. Let $T : \mathcal{A}^n \to \mathcal{B}^*, I \mapsto x \ s.t. |\{y \in S_k(\text{or } \mathcal{S}_k) | y < x\}| = i$ where i is the integer form of I. In other words, $T(I)$ is the $(i+1)^{th}$ smallest strand. In order to make alignment for DNA, S_k is preferred, with negligible cost added.

B.2.3 Proof of the optimum of T'

Theorem B.4. *T' is optimum.*

Proof. Proof by contradiction. Assume there is a better suitable encoding $f : \mathcal{A}^n \cup \mathcal{A}^{n-1} \cup \cdots \cup \mathcal{A}^1 \to \mathcal{S}_k$ with higher density : $\lim_{k\to\infty} d(f) > \lim_{k\to\infty} \frac{\log_2 |\mathcal{S}_k|}{k}$, then

$$\lim_{k\to\infty} d(f) = \lim_{k\to\infty} \frac{\mathbf{E}_x(\text{len}(x))}{k} \geq \lim_{k\to\infty} \frac{\log_2 |\mathcal{S}_k|}{k}(1+\epsilon) \tag{21}$$

So $\exists k, x \ s.t.$

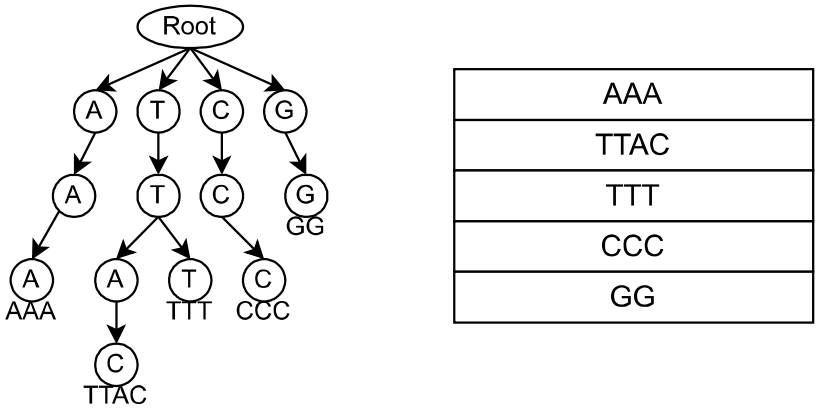
$$|\mathcal{A}^n \cup \mathcal{A}^{n-1} \cup \cdots \cup \mathcal{A}^1| \geq 2^{\text{len}(x)} \geq 2^{(\log_2 |\mathcal{S}_k|)(1+\epsilon)} > |S_k|(1+\epsilon) \tag{22}$$

So the encoding f can not be decoded correctly because there exist 2 different input mapping to the same strand. □

B.3 Siyuan Code Data Structure

B.3.1 Trie Tree

A Trie tree is a k-ary search tree, storing strings with finite length. Any string in the Trie tree is stored as a path from the root to the leaf, as shown in Supplementary Figure 6. To be formal, we will define the Trie tree in our case.



Supplementary Figure 6: Trie Tree

For any strand $s = (s_1, s_2, \cdots, s_k)$, its substring $t = (s_i, s_{i+1}, \cdots, s_k)$ $(\forall i \in \{1, \cdots, k+1\})$ is defined as its suffix ($t \in \text{suffix}(s)$) and its substring $t = (s_1, s_2, \cdots, s_i)$ $(\forall i \in \{0, \cdots, k\})$ is defined as its prefix ($t \in \text{prefix}(s)$). Specially, according to the definition, the empty string τ is its suffix $(s_{k+1}, s_{k+2}, \cdots, s_k) = \tau$ and its prefix $(s_1, s_2, \cdots, s_0) = \tau$. For example, τ, (A), (AT), (ATC), $(ATCG)$ are the prefix substrings of $(ATCG)$.

Let S be the set of suitable strands and $\overline{S}$ be the set of unsuitable strands. Let $\mathcal{I}$ be the set of undesired motifs or the set of unsuitable substrings, and it is reasonable to assume that each string in $\mathcal{I}$ is not a substring of another string in $\mathcal{I}$, because removing the long string and keep the substring in $\mathcal{I}$ can keep the property of $\mathcal{I}$. According to the assumption, $\overline{S}$ is the set of strands with a substring in the set $\mathcal{I}$.

$$\overline{S} = \{v | \exists u \text{ is a subset of } v, u \in \mathcal{I}\} \tag{23}$$

Siyuan Code Trie tree is defined as $G = (V, E)$, where V is its vertices set and G is its edges set. Unsuitable substrings $\mathcal{I}$ are stored in the Trie tree. V is a subset of $\mathcal{B}^* = \{A, T, C, G\}^*$, and each vertex $v \in V$ is actually a string in $\mathcal{B}^*$. V is a defined as the set of all prefix of the unsuitable strands.

$$V = \{v | \exists u \in \mathcal{I}, v \text{ is prefix of } u\} \tag{24}$$

For each unsuitable strand $s = (s_1, s_2, \cdots, s_k) \in \mathcal{I}$, V includes all its prefix substrings.

$$\{(s_1, s_2, \cdots, s_{k'}) | k' \in \{0, 1, \cdots, k\}\} \subseteq V \tag{25}$$

For each $v \in V$, let $v = (s_1, s_2, \cdots, s_k)$. It is obviously that $\forall v \neq \tau$, $v' = (s_1, s_2, \cdots, s_{k-1}) \in V$. Define v as the son of v' and define v' as the parent of v. Let $\text{par} : V - \{\tau\} \to V, v \mapsto v'$ be the parent function. Let $e = (v', v) \in E$. In other words, edge is pointing to a vertex from its prefix shorter by one character. Also, it can be written as $v \in \text{child}(v')$ or $v = \text{child}_{s_k}(v')$.

To avoid confusion, we would like to clarify that the strings and the vertices are the same pointer. In implementation, the strings in the graph are stored as $O(1)$ ***space vertices. In mathematical definition, the vertices are strings. Do not attach the content of the string to the vertex in algorithm implementation.***

Lemma B.5. *It is obvious that the Trie tree can be constructed efficiently, within $O(L)$ time and $O(L)$ space, where $L = \sum_{i \in \mathcal{I}} \text{len}(i)$ indicates the total length of $\mathcal{I}$. [40]*

Supplementary Lemma B.5 follows from previous works. Note that Supplementary Lemma B.5 is used but not proposed in our Siyuan Code.

B.3.2 Determining an unsuitable strand

It is obvious that u is a substring of $v \Leftrightarrow \exists w$ s.t. $u \in \text{suffix}(w)$ and $w \in \text{prefix}(v) \Leftrightarrow \exists w$ s.t. $u \in \text{prefix}(w)$ and $w \in \text{suffix}(v)$. (By definition)

Considering a process, a strand is read into the buffer one character (base) by one character (base). Each character read is a step. It is obvious that the buffer string is an increasing prefix of the input strand. After reading each character, if the buffer string w contains a suffix u in $\mathcal{I}$, then the strand contains $u \in \mathcal{I}$ as a substring.

It has been assumed that no string in $\mathcal{I}$ is a substring of another string in $\mathcal{I}$, otherwise just remove the longer string.

Theorem B.6.

$$\textit{The input strand } v \textit{ contains unsuitable substring} \tag{26}$$

$$\Leftrightarrow \textit{There exists some prefix } w \textit{ of the input } v \textit{ that contains unsuitable suffix } u \in \mathcal{I}$$

$$\Leftrightarrow \exists \textit{ prefix } w \textit{ of the input strand } \exists u \in \mathcal{I} \textit{ s.t } u \in \text{suffix}(w)$$

$$\Leftrightarrow \exists \textit{ prefix } w \textit{ of the input strand, let } u' \textit{ be the longest suffix of } w \textit{ in } V, \exists u \in \text{suffix}(u') \cap \mathcal{I}$$

$$\Leftrightarrow \exists \textit{ prefix } w \textit{ of the input strand, let } u' \textit{ be the longest suffix of } w \textit{ in } V, u' \in \mathcal{I}$$

Proof.

The input strand v contains unsuitable substring

$\Rightarrow$ There exists some prefix w of the input v that contains unsuitable suffix $u \in \mathcal{I}$

($\exists w, u$, such that w is a prefix of v and u is a suffix of w and $u \in \mathcal{I}$)

$\Rightarrow \exists$ prefix w of the input strand $\exists u \in \mathcal{I}$ s.t $u \in \text{suffix}(w)$

$$\Rightarrow \exists \text{ prefix } w \text{ of the input strand, let } u' \text{ be the longest suffix of } w \text{ in } V,\ \exists u \in \text{suffix}(u') \cap \mathcal{I} \ (1)$$

$$\Rightarrow \exists \text{ prefix } w \text{ of the input strand, let } u' \text{ be the longest suffix of } w \text{ in } V,\ u' \in \mathcal{I} \ (2)$$

$$\Rightarrow \text{The input strand contains unsuitable substring}$$

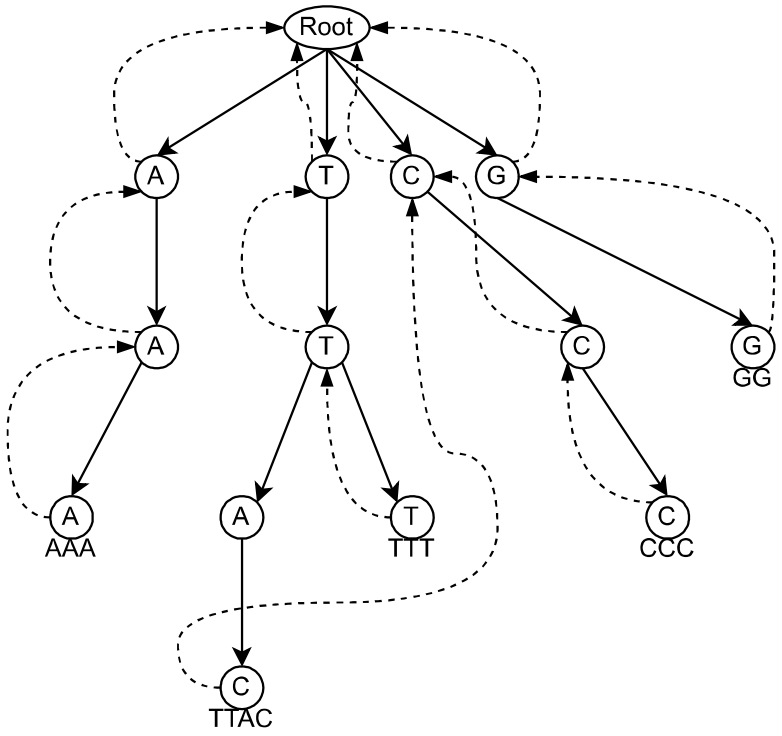
(1)and (2) are obvious and explained as follows. (1) Let the longest suffix of w in V be u', so any suffix of w in V is not longer than u' and also suffix of u'. Note that u is a suffix of w in $\mathcal{I} \subseteq V$. This means that $u \in \mathcal{I} \subseteq V$ should be suffix of u'. (2) Also, if suffix $u(\neq u')$ of u' is in $\mathcal{I}$, then u' should be unsuitable. Note that $u' \in V$, so there is a longer string in $\mathcal{I}$ containing u and u' as substrings, so the assumption $u \neq u'$ is wrong. □

Corollary B.6.1. *So the necessary and sufficient condition of the input strand being unsuitable is that at some step, the longest suffix of the buffer in V is in $\mathcal{I}$.*

According to the proposition given above, it is easy to construct a naive algorithm that determines whether a strand is suitable or not. Input the strand one base by one base into the buffer, and after each base input, see whether the longest suffix of the buffer string is in $\mathcal{I}$. Finding the longest suffix of the buffer string can cost $O(L)$ time. Total time complexity is $O(nL)$, which is polynomial but slow. In the next section named AC Algorithm, an acceleration will be introduced [41] to speed up suitability determination, in order to acquire $O(n)$ time complexity.

B.3.3 AC Algorithm

AC Algorithm [41] is sometimes called AC Automaton for its fail edge transition. AC algorithm is an efficient method for locating the longest suffixes of the buffer at all the steps appearing in the vertices V of the tree. With the suffixes and the "necessary and sufficient condition" in Supplementary §B.3.2, one can efficiently determine whether a set of small strings appears in a long string as substrings. For example, a strand is read by the AC algorithm, and it determines whether it is an unsuitable strand containing unsuitable substrings by checking whether the longest suffixes are suitable. Thus, with the trivial reduction of substring checking to longest suffixes finding, only the longest suffixes of the buffer at all steps are what we need from AC Algorithm.



Supplementary Figure 7: AC Algorithm

As shown in Supplementary Figure 7 The key of AC Algorithm is its fail edge F. Fail edge points to the longest suffix except itself of a given vertex.

$$F : v \mapsto \begin{cases} \arg\max_{u \text{ s.t. } u \in V, u \neq v, u \in suffix(v)} \text{len}(u) & v \neq \tau \\ \tau & v = \tau \end{cases} \tag{27}$$

With the help of fail edge, finding the longest suffix of the buffer string is much easier. After a character c is added into the back of the buffer string w, the buffer string becomes $w + c$. The longest suffix v of w in V has been calculated in advance, and the task is to calculate the longest suffix $v' + c$ of $w + c$ in V.

It is obvious that v' should be a suffix of w, so v' is not longer than the longest suffix v of w. Thus $v' \in \text{suffix}(v) \cap V$. Thus $v' \in \{v, F(v), F(F(v)), \cdots\}$. Thus reasonable solution to the task is to start with $v' := v$, and set $v' := F(v')$ until $v' + c \in V$.

Thus, AC Algorithm preprocession can be described as shown in Supplementary Algorithm 1 and usage of AC Algorithm can be described as shown in Supplementary Algorithm 2.

Supplementary Algorithm 1 AC Algorithm Preprocessing

$\mathcal{V} = [v_1, v_2, \cdots, v_n] \leftarrow$ bfs order of V

Initialize fail[$\cdot$]

root $\leftarrow \tau$,fail[root] $\leftarrow root$

for $v \in \mathcal{V}$ **do**

  for $c \in \{A, T, C, G\}$ s.t. child$_c(v)$ exists **do**

    $v' \leftarrow$ fail[v]

    while $v' \neq root$ and child$_c(v')$ does not exist **do**

      $v' \leftarrow$ fail[v']

    end while

    if child$_c(v')$ exists **then**

      fail[child$_c(v)$] $\leftarrow$ child$_c(v')$

    else

      fail[child$_c(v)$] $\leftarrow$ root

    end if

  end for

end for

return fail[$\cdot$]

Supplementary Algorithm 2 AC Algorithm for longest suffix finding

Initialize Suffix[$\cdot$] = []

$s \leftarrow$ root $= \tau$

for $c \in$ Str **do**

  while $s \neq \tau$ and child$_c(s)$ does not exists **do**

    $s \leftarrow$ fail[s]

  end while

  if child$_c(s)$ exists **then**

    $s \leftarrow$ child$_c s$

  end if

  Suffix[$\cdot$] = Suffix[$\cdot$] + [s]

end for

return Suffix[$\cdot$]

It can be proved that the AC Algorithm can speed up the determining and acquire a time complexity of $O(L)$ in fail edge construction and $O(n)$ in determination. We do not give the proof of AC Algorithm's time complexity because it was proved by Alfred et.al [41]. Our automaton and encoding is inspired by AC Algorithm.

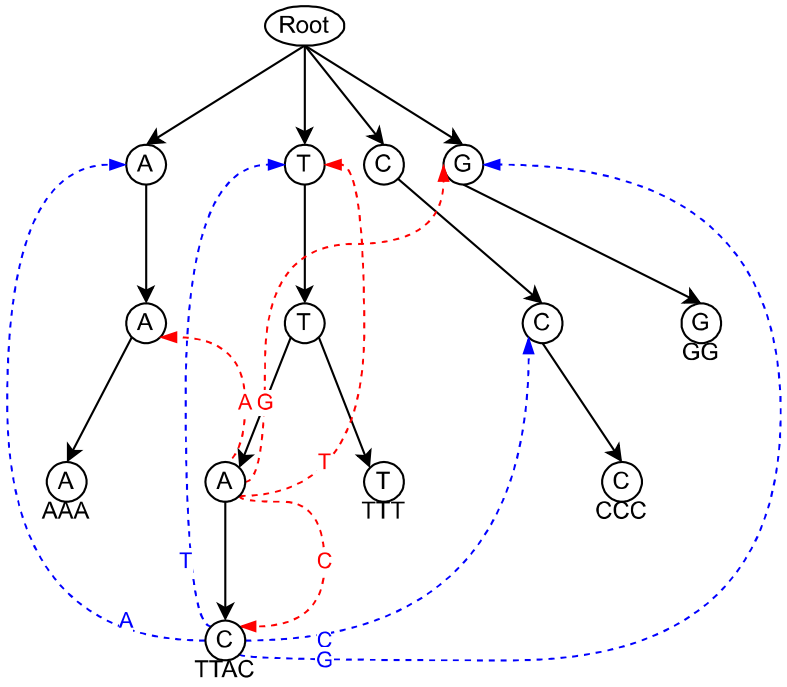
B.3.4 Trie Automaton

In this paper, Trie automaton, as shown in Supplementary Figure 8, as an extension of AC Algorithm, will be used to support the suitable string counting problem in the next section. Press edge calculates the longest suffix of a buffer string after inserting character c at its back.

$$P_c : v \mapsto \arg\max_{u \in V \text{ s.t. } u \in \text{suffix}(v+c)} \text{len}(u) \tag{28}$$

$P_c(v)$ is the longest suffix of $v + c$ in V. To be convenient, $P_{c_1,c_2,c_3,\cdots,c_k}$ is an abbreviation of $P_{c_k} P_{c_{k-1}} \cdots P_{c_1}(v)$.

With the help of Press edge, the AC Algorithm can be further accelerated as shown in Supplementary Algorithm 4. After a character c is added into the back of the buffer list



Supplementary Figure 8: Trie Automaton

w, the buffer string becomes $w + c$. The longest suffix v of w in V has been calculated in advance, and the task is to calculate the longest suffix $v' + c$ of w+c in V. With the help of P_c, just let $v' = \text{par}(P_c(v))$.

The construction of P_c edge with F edge is simpler and more static. The press edge can be calculate as followed:

$$P_c(v + c') = \begin{cases} \text{child}_c(v + c') & \textbf{if } \text{child}_c(v + c') \text{ exists} \\ P_c(F(v + c')) & \textbf{if } \text{child}_c(v + c') \text{ does not exist} \end{cases}$$

$$P_c(\text{root}) = \begin{cases} c & \textbf{if } \text{child}_c(\text{root}) \text{ exists} \\ \text{root} & \textbf{otherwise} \end{cases}$$

$$F(v + c') = P_{c'}(F(v)) \textbf{ if } v \neq \text{root}$$

$$F(\text{root}) = \text{root}; F(c') = \text{root} \tag{29}$$

Theorem B.7. *Well-definedness of the formula*

Proof. It is obviously by definition that $\text{len}(P_c(v + c')) \leq \text{len}(v + c') + 1$ and $\text{len}(F(v + c')) \leq$

$\text{len}(v + c') - 1$. BFS search can be applied when calculating the press edge and fail edge. When calculating $v + c'$, the press and fail edge of any string not longer than $\text{len}(v)$ has been already calculated. $F(v)$ has been already calculated and $\text{len}(F(v)) \leq n' - 1$. Thus $P_{c'}(F(v))$ has been calculated and $\text{len}(P_{c'}(F(v))) \leq n'$. Thus $P_c(P_{c'}(F(v)))$ has been calculated and the formula given above is well-defined. □

Theorem B.8. *The correctness of the formula*

Proof. 1. On one hand, $P_{c'}(F(v))$ is a suffix of $v + c'$ shorter than $v + c'$, so it is not longer than then $F(v + c')$. On the other hand, $\text{par}(F(v + c'))$ is a suffix of v shorter than v, so it is not longer than $F(v)$. Thus $\text{len}(F(v + c')) = \text{len}(P_{c'}(\text{par}(F(v + c'))) \leq \text{len}(P_{c'}(F(v)))$. Thus $P_{c'}(F(v))$ is the longest suffix of $v + c'$ shorter than $v + c'$, and thus $F(v + c') = P_{c'}(F(v))$.

2. When the child c of $v+c'$ exists, the result of $P_c(v+c')$ is trivial. When it does not exist, $P_c(v+c')$ should be not longer than $v+c'$. On one hand, $P_c(F(v+c'))$ is a suffix of $v+c'+c$, so it is not longer than the longest suffix $P_c(v + c')$. On the other hand, $\text{par}(P_c(v + c'))$ is a suffix of $v + c'$ shorter than $v + c'$, so $\text{par}(P_c(v + c'))$ is not longer than $F(v + c')$. Thus $\text{len}(P_c(v + c')) = \text{len}(P_c(\text{par}(P_c(v + c')))) \leq \text{len}(P_c(F(v + c')))$. Thus $P_c(F(v + c'))$ is the longest suffix of $v + c' + c$, and thus $P_c(v + c') = P_c(F(v + c'))$. □

Supplementary Algorithm 3 Trie Automaton Construction

Initialize $\text{press}_{(\cdot)}[\cdot], \text{fail}[\cdot]$
Preprocess $press$ and $fail$ for $root$ and its children.
for $v \in$ bfs order of V **if** $v \neq root$ **do**
  for $c' \in \{A, T, C, G\}$ s.t. $\text{child}_{c'}(v)$ exists **do**
    $v' = \text{child}_{c'}(v)$
    $\text{fail}[v'] = \text{press}_{c'}[\text{fail}[v]]$
    for $c \in \{A, T, C, G\}$ **do**
      if $\text{child}_c(v')$ exists **then**
        $\text{press}_c[v'] = \text{child}_c(v')$
      else
        $\text{press}_c[v'] = \text{press}_c[\text{fail}[v']]$
      end if
    end for
  end for
end for
return $\text{press}_{(\cdot)}[\cdot], \text{fail}[\cdot]$

Supplementary Algorithm 4 Trie Automaton for longest suffix finding

Initialize $\text{Suffix}[\cdot] = []$
$s \leftarrow \text{root} = \tau$
for $c \in \text{Str}$ **do**
  $s \leftarrow \text{press}_c[s]$
  $\text{Suffix}[\cdot] = \text{Suffix}[\cdot] + [s]$
end for
return $\text{Suffix}[\cdot]$

Thus the Trie Automaton construction can be described as shown in Supplementary Algorithm 3 and the Trie Automaton usage can be described as shown in Supplementary Algorithm 4.

To avoid confusion, we would like to clarify that the strings and the vertices are the same pointer. In implementation, the strings in the graph are stored as $O(1)$ space vertices. In mathematical definition, the vertices are strings. Do not attach the content of the string to the vertex in implementation.

Theorem B.9. *Time and space complexity of Supplementary Algorithm 3 are $O(L)$ and $O(L)$.*

Time and space complexity of Supplementary Algorithm 4 are $O(n)$ and $O(n)$.

Proof. In Supplementary Algorithm 3, the outermost for-loop (line 3-15) iterates about $|V| \leq L + 1$ times, while the inner loop (line 4-14) iterates constant times, therefore the overall time complexity is $O(L)$. Here, L is the total length of all the constraints and V is the vertex number of the trie tree. The inner loop (line 4-14) iterates constant times because the for loops in line 4 and line 7 have constant size.

In Supplementary Algorithm 3, the space complexity is $O(L)$. The main costs are the $\text{press}_{(\cdot)}[\cdot], \text{fail}[\cdot]$ arrays.

In Supplementary Algorithm 4, the outermost for-loop (line 3-6) iterates n times, while the inner loop (line 4-5) operates in constant time, therefore the overall time complexity is $O(n)$. Here, n is the length of the input string where we want to find the longest suffix.

In Supplementary Algorithm 4, excluding the output, the input and the data structure in Supplementary Algorithm 3, there is only $O(1)$ extra space needed. Note that the cost of $\text{press}_{(\cdot)}[\cdot], \text{fail}[\cdot]$ are calculated in the Supplementary Algorithm 3.

In Supplementary Algorithm 4, excluding the input and the data structure in Supplementary Algorithm 3, there is only $O(n)$ extra space needed for output. □

B.4 Siyuan Code Algorithm

Theorem B.10. *A string being unsuitable is equivalent to a string containing an unsuitable substring pattern and equivalent to a string as a* press *sequence of which the path visits unsuitable vertex ($\mathcal{I}$) in the Trie automaton.*

Proof. According to Supplementary §B.3.2 and Supplementary Corollary B.6.1 and the property of the press edge, a string being unsuitable is equivalent to a string containing an un-

suitable substring pattern and equivalent to a string as a press sequence of which the path visits unsuitable vertex ($\mathcal{I}$) in the Trie automaton:

$$s = (s_1, s_2, \cdots, s_n) \text{ is unsuitable} \tag{30}$$

$$\Leftrightarrow \exists i, j \text{ s.t. } s' = (s_i, s_{i+1}, \cdots, s_j) \in \mathcal{I} \tag{31}$$

$$\Leftrightarrow \exists j \text{ s.t. } P_{s_j}(\cdots(P_{s_1}(\tau)\cdots) \in \mathcal{I} \tag{32}$$

$$\Leftrightarrow \{\tau, P_{s_1}(\tau), P_{s_2}(P_{s_1}(\tau)), \cdots, P_{s_n}(\cdots(P_{s_1}(\tau)\cdots)\} \cap \mathcal{I} \neq \emptyset \tag{33}$$

□

As a result, any path on press edge with length n on the graph without visiting the unsuitable substring set $\mathcal{I}$ is corresponding to a suitable strand of length n. Under generalized scenarios, at beginning state $v = (t_1, t_2, \cdots)$ of the automaton, the press path $(s_1, s_2, \cdots, s_n)$ without visiting unsuitable vertices can be corresponding to a suitable strand $(t_1, t_2, \cdots) + (s_1, s_2, \cdots, s_n) = (t_1, t_2, \cdots, s_1, s_2, \cdots, s_n)$.

Definition. Thus, a suitable path in the automaton can be defined as a press path without visiting the unsuitable substring set $\mathcal{I}$: $(v_1, v_2, \cdots, v_l)$ $(\{v_1, v_2, \cdots, v_l\} \cap \mathcal{I} = \emptyset)$.

In this section, this property will be used to provide a solution of the counting problem finding the $(k+1)^{th}$ smallest suitable strand. This counting problem can be a natural mapping function mentioned in previous sections.

B.4.1 Dynamic programming and definition of the variables

Siyuan Code is using DP to give a sufficient method counting the $(k+1)^{th}$ smallest suitable strand. DP is a generally used math method that solves a difficult problem using the solution of several easier problems.

Note that there is a one-to-one corresponding between a Trie automaton path from τ without visiting $\mathcal{I}$ and a suitable strand. As a result, counting the strand equals counting the Trie automaton path. Thus, DP(v) is defined as the number of suitable path of length l from vertex v:

$$\begin{aligned} \mathrm{DP}: \quad & V \otimes \mathbb{Z}^+ \cup \{0\} \to \mathbb{Z}^+ \cup \{0\} \\ & (v, l) \mapsto \{(v_1, v_2, \cdots, v_l) | \{v_1, v_2, \cdots, v_l\} \cap \mathcal{I} = \emptyset\} \end{aligned} \tag{34}$$

Note that 2 paths visiting the same set of vertices but different set of press edge are considered as different paths.

B.4.2 Derive the DP value

Theorem B.11. $\mathrm{DP}(v, 0) = \begin{cases} 0 & v \in \mathcal{I} \\ 1 & v \notin \mathcal{I} \end{cases}$ *and* $\mathrm{DP}(v, l) = \begin{cases} \sum_c \mathrm{DP}\left(P_c(v), l-1\right) & v \notin \mathcal{I} \\ 0 & v \in \mathcal{I} \end{cases}$

Proof. DP value can be derived easily with the help of the Trie Automaton. The boundary value is that:

$$\mathrm{DP}(v, 0) = \begin{cases} 0 & v \in \mathcal{I} \\ 1 & v \notin \mathcal{I} \end{cases} \tag{35}$$

A suitable path of length $l > 0$ from v can be seen as 2 stages: pressing character c and then press a sequence of length $l-1$ from $P_c(v)$. Thus, the number of suitable path of length $l > 0$ from v is equal to the sum of the number of suitable path of length $l-1$ from all the possible direct pressing result of v:

$$\mathrm{DP}(v, l) = \begin{cases} \sum_c \mathrm{DP}\left(P_c(v), l-1\right) & v \notin \mathcal{I} \\ 0 & v \in \mathcal{I} \end{cases} \tag{36}$$

Note that it does not matter whether $P_c(v)$ is a suitable state, because $\mathrm{DP}\left(P_c(v), l-1\right) = 0$ if $P_c(v)$ is unsuitable. □

B.4.3 Conditional revision to the DP value

In order to calculate the conditional counting result, in other words, the $(i+1)^{th}$ smallest suitable sequence under the condition that the forward and reverse primers are added, some revisions are taken to the definition and the deriving of DP value.

The revised definition of the DP value is the number of suitable paths of length l from vertex v such that the path adding the right primer is still suitable. The transfer equation

is not modified and the boundary value is revised as follows:

$$\mathrm{DP}(v,0) = \begin{cases} 0 & \exists r \in prefix(primerR) \text{ s.t. } P_r(v) \in \mathcal{I} \\ 1 & \forall r \in prefix(primerR)\ P_r(v) \notin \mathcal{I} \end{cases} \tag{37}$$

The pseudo code is described in Supplementary Algorithm 5.

Theorem B.12. *The time complexity of* DP *is* $O(nL)$ *and the space complexity is also* $O(nL)$ *if all the operations are* $O(1)$.

Proof. The time complexity is obvious because the dominant part of the code contains two nested for loops (ignoring the very inner constant time part in the loop), with the outer loop executing n times and the inner loop iterating at most $L+1$ times. The space complexity is also $O(nL)$, because the DP array is $O(nL)$ and the press edges and fail edges are $O(L)$. □

B.4.4 Encode using Siyuan Code DP value

Encoding is to give a mapping from $\mathcal{A}^n$ to $\mathbb{Z}$ and to $\mathcal{B}^m$. The first stage is trivial, using the Conversion Between Decimal and Binary. The stage part is to find the $i+1^{th}$ lexicographical smallest press edge path or strand.

Definitions. $\mathrm{Enc}(v,i,l)$ is defined as the $i+1^{th}$ smallest suitable press path of length l from v. The output of the encoder should be $\mathrm{Enc}(P_{\mathrm{primerL}}(\tau),i,m)$. It is the $i+1^{th}$ smallest suitable press path of length l under the condition that adding the reverse primer primerR is still suitable from the forward primer primerL.

Theorem B.13. *Correctness of Supplementary Algorithm 6.*

Proof. The problem can be recursively conquered. When calculating $\mathrm{Enc}(v,i,l)$, try $c=c_0$ at first (line 9).

If i is smaller than the total number $\mathrm{DP}(P_{c_0},l-1)$ of suitable strand from $P_{c_0}(v)$, then the first bit of $\mathrm{Enc}(v,i,l)$ should be c_0 (line 13), and the rest of the $\mathrm{Enc}(v,i,l)$ should be $\mathrm{Enc}(P_{c_0},i,l-1)$ (line 14).

If i is not smaller than $\mathrm{DP}(P_{c_0},l-1)$, then just decrease i by $\mathrm{DP}(P_{c_0},l-1)$ and try c_1 instead of c_0(line 16). □

The pseudo code is described in Supplementary Algorithm 6.

Theorem B.14. *The time and space complexities of encoding are both $O(n)$ if all the operations are $O(1)$.*

Proof. The time complexity is obvious because the outer for loop iterates n times, while the inner loop maintains constant time complexity, resulting in an overall time complexity of $O(n)$.

Note that the DP value can be preprocessed and shared among different strands' encoding. Thus, the DP value preprocessing deriving time complexity $O(nL)$ can be amortized by as many strands as possible. The actual encoding time complexity is still $O(n)$.

The total space complexity including the data structure in Supplementary Algorithm 5 is $O(nL)$, but the space additional to the DP preprocessing is only the $O(n)$ Strand[$\cdot$] array. □

Supplementary Algorithm 5 DP value calculation (with conditional revision)

for $v \in V$ **do**
  ok $\leftarrow True$, $u \leftarrow v$
  if $u \in \mathcal{I}$ **then**
    $ok \leftarrow False$ {Make sure the suitability of the node }
  end if
  for $c \in primerR$ **do**
    $u \leftarrow P_c(u)$
    if $u \in \mathcal{I}$ **then**
      $ok \leftarrow False$ {Make sure the suitability of the path with the reverse primer }
    end if
  end for
  if ok **then**
    DP$[v, 0] \leftarrow 1$
  else
    DP$[v, 0] \leftarrow 0$
  end if
end for
for $l \in \{1, \cdots, n\}$ **do**
  for $v \in V$ **do**
    tmp $\leftarrow 0$
    for $c \in \{A, T, C, G\}$ **do**
      tmp $\leftarrow$ tmp + DP$(P_c(v), l-1)$ {Derive DP with transition equation }
    end for
    if $v \in \mathcal{I}$ **then**
      DP$[v, l] \leftarrow 0$ {Mark 0 if not suitable }
    else
      DP$[v, l] \leftarrow$ tmp
    end if
  end for
end for

Supplementary Algorithm 6 Encode using DP (with conditional revision)

This algorithm is to calculate the $i+1^{th}$ lexicographic order strand of length len
DP is derived using Supplementary Algorithm 5
Initialize Strand[·]
Input len, i
$v \leftarrow P_{\text{primerL}}(\text{root})$ {The starting node is the forward primer }
for $j \in \{0, 1, \cdots \text{len} - 1\}$ **do**
    depth $\leftarrow$ len $- j - 1$
    $skip \leftarrow$ false
    for $c \in \{A, T, C, G\}$ **do**
        if Not $skip$ **then**
            if $i <$ DP[Press$_c[v]$, depth] **then**
                $skip \leftarrow$ true
                Strand$[j] \leftarrow c$
                $v \leftarrow$ Press$_c[v]$
            else
                $i \leftarrow i -$ DP[Press$_c[v]$, depth]
            end if
        end if
    end for
end for
return Strand[·]

B.4.5 Decode using Siyuan Code DP value

Decoding is to give a mapping from $\mathcal{B}^m$ to $\mathbb{Z}$ and to $\mathcal{A}^n$. The second stage is trivial using Conversion Between Decimal and Binary. The first stage is to count the number of suitable lexicographical smaller press edge path or strand than the given press edge path.

Definitions. Dec(v, s) is define as the number of suitable lexicographical smaller press edge path than $s = (s_1, s_2, \cdots, s_j)$ from v. The output of the decoder should be Dec$(P_{\text{primerL}}(\tau), s)$

Theorem B.15. *Correctness of Supplementary Algorithm 7.*

Proof. The problem can be recursively conquered.

When calculating $\mathrm{Dec}(v, s)$ $s = (s_1, s_2, \cdots, s_j)$, it can be divided into 2 stages. One is to calculate the smaller path with different first character, which can be calculated using DP value. The other is to calculate the smaller path with the same first character, which is a smaller Dec problem.

$$\mathrm{Dec}(v, s) = \sum_{c \text{ s.t. } c<s_1} DP(P_c(v), j-1) + Dec(P_{s_1}(v), s') \ , \ s' = (s_2, \cdots, s_j) \tag{38}$$

□

The pseudo code is described in Supplementary Algorithm 7.

Theorem B.16. *The time and space complexities of decoding are both $O(n)$ if all the operations are $O(1)$.*

Proof. The time complexity is obvious because the outer for loop iterates n times, while the inner loop maintains constant time complexity, resulting in an overall time complexity of $O(n)$.

The total space complexity including Supplementary Algorithm 5 is $O(nL)$, but the space additional to the DP preprocessing is only the $O(n)$ Strand[·] array. □

Supplementary Algorithm 7 Decode using DP (with conditional revision)

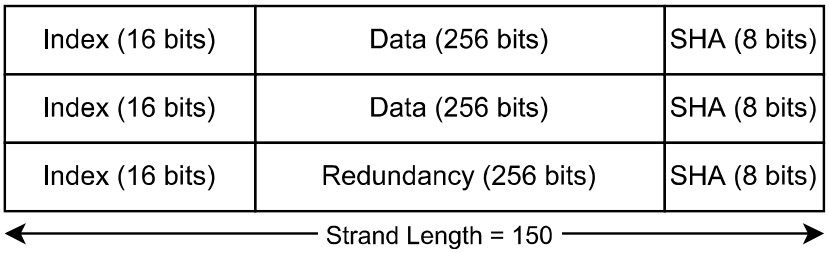
Input Strand$[\cdot], len$
$i \leftarrow 0$
$v \leftarrow P_{\text{primerL}}(\text{root})$ {The starting node is the forward primer }
for $j \in \{0, \cdots, \text{len} - 1\}$ **do**
  depth $\leftarrow \text{len} - j - 1$
  $skip \leftarrow false$
  for $c \in \{A, T, C, G\}$ **do**
    if Not $skip$ **then**
      if $c = \text{Strand}[j]$ **then**
        $v \leftarrow \text{Press}_c[v]$
        $skip \leftarrow$ true
      else
        $i = i + \text{DP}[\text{Press}_c[v], \text{depth}]$
      end if
    end if
  end for
end for
return i

B.5 An error correcting DNA coding with Siyuan Code

This section is to introduce the whole system and how to add error correction into the system.

B.5.1 Slicing

In Siyuan Code, each binary message is regarded as a binary sequence, and the sequence should be cut into slices if it is too long for DNA synthesis. After DNA sequencing, the slices become out of order. In order to deal with the out-of-order problem, each slice is attached with an index of the slice, so that it is possible to get the information back in order.



Supplementary Figure 9: An example of the structure of Siyuan Code

Here, each slice shown in Supplementary Figure 9 as an example contains 35 bytes in which the index contains 2 bytes.

B.5.2 Inter-slice error correction

In Siyuan Code, each strand has a very high accuracy, but the accuracy is not 1. Thus, error correction is very necessary. Siyuan Code uses inter-slice error correction instead of intra-slice error correction.

Intra-slice error correction does not work well. The error transition of a single strand can be regraded as:

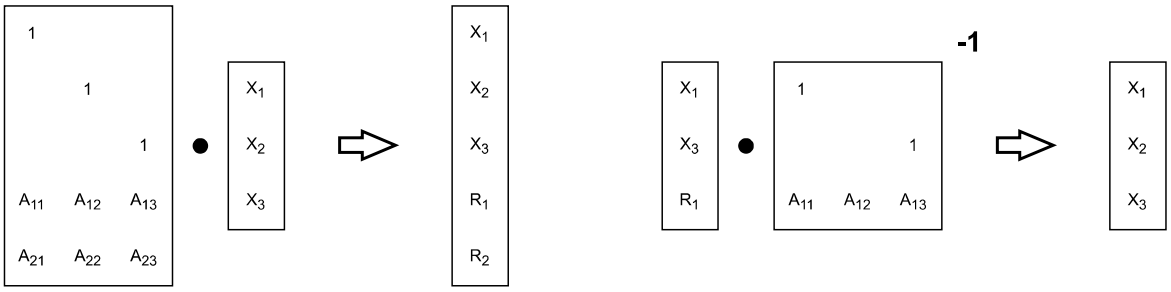
$$\begin{aligned} Err: \quad & \mathcal{B}^* \otimes \mathcal{B}^* \to \mathbb{R} \\ & \left(s = (s_1, s_2, \cdots, s_{m_s}), t = (t_1, t_2, \cdots, t_{m_t})\right) \\ & \mapsto Pr\left(\text{errored data is } t | \text{source is } s\right) \end{aligned} \tag{39}$$

If intra-slice error correction is added, it should consider the error transition probability when decoding, by simply maximizing the probability in maximum likelihood estimation:

$$\begin{aligned} & \arg\max_{\text{input } x} Pr(\text{input is } x | \text{output is } y) \\ = \quad & \arg\max_{\text{input } x} Pr(x) Err(x, y) \end{aligned} \tag{40}$$

The error transition remains an open problem when identifying the low accuracy DNA strand is much easier. Thus, the previous knowledge of the biology does not support customized intra-slice error correction. As a result, the Siyuan Code uses the inter-slice error correction instead of intra-slice error correction.

Inter-slice error correction works well. In inter-slice error correction, each slice is sym-



Supplementary Figure 10: RS code example

metric, so the error probability of the slice could be considered as independent identically distribution, which is common in the digital computer. As a result, the previous erasure code designed for digital computer can easily work with the inter-slice error correction.

Take RS code (shown in Supplementary Figure 10) as an example. Here, RS code slices are added into the binary slices before the index is added into each slice. This kind of erasure code adding does not interrupt the encoding steps. The error correcting can be regarded as a mapping from binary data to binary data with RS code.

As shown in the graph, more strands are added in, so that the error of a small number of strands does not cause the error of the whole system. The slices with error can be corrected with the correct slices. The scope of the error correcting does not contain its index or hash value, because the RS code or MDS code error correction can be only based on the data, and the redundancy slice can have its own index and hash value as shown in Supplementary Figure 9.

B.5.3 Intra-slice error detection

Intra-slice error detection can be added to determine whether a single strand contains error, without the help of other strands. There are 2 advantages of the error detection. First, when doing sequencing, more than 1 strands are read, and error detection can help finding the correct strands. Second, with error detection, the error correction step can simply just assume there are only missing data or correct data.

As shown in Supplementary Figure 9, 1 byte is designed to store a hash value, like high digits of MD5 or SHA value. The scope of the error detection contains all the information

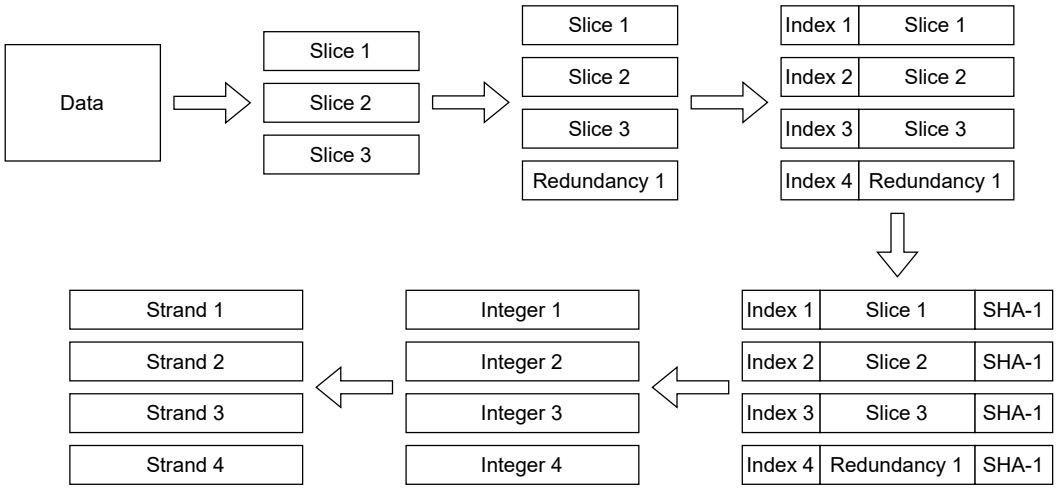
in the slice, including the index. The redundancy slices also have their own hash values.

B.5.4 Multi-time sequencing

With the help of Intra-slice error detection, a common trick in sequencing of DNA coding system is the multi-time sequencing. In other words, each strand is sequenced for multiple times and then filtered according to hash value. The probability that all the strands are wrong is very low. At the same time, the hash value attached to the slice of information can help determine errored strands. With the hash value, the overall error of a single strand is actually exponentially small.

B.5.5 Process of encoding and decoding

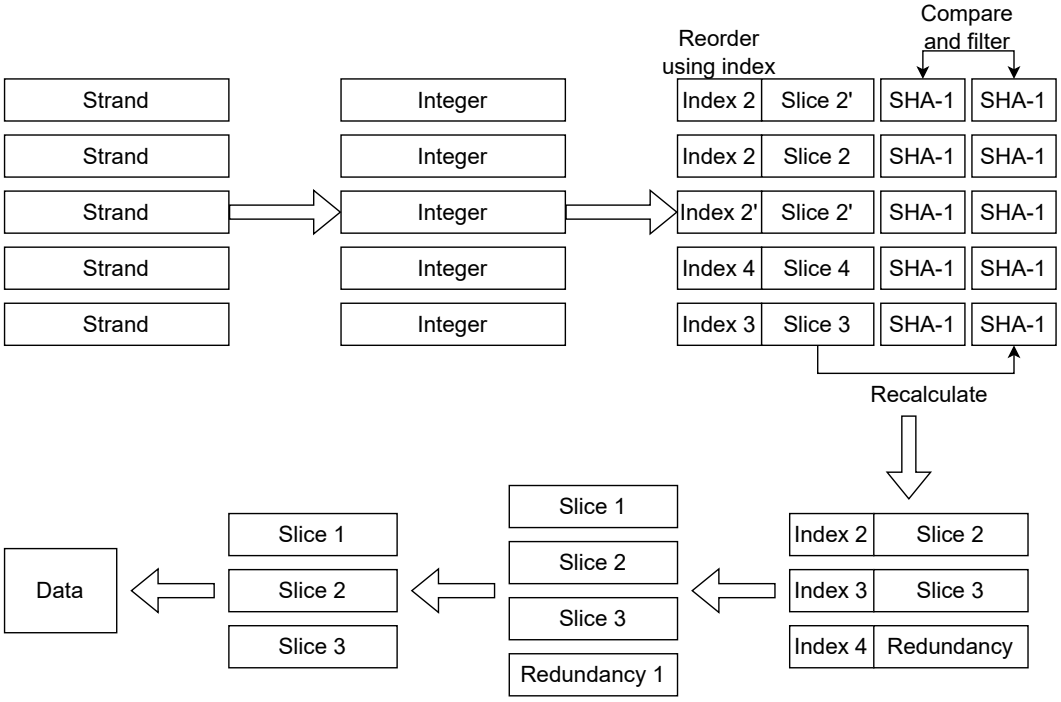
The encoding stage (shown in Supplementary Figure 11): First, the data is cut into slices. Second, the redundancy slices are added in to support inter-strand error correction. Third, the index is inserted into the slices to support reordering of the strands. Fourth, the hash value is calculated to support intra-strand error detection. Fifth, the binary data with error correction and error detection is turned into the DNA strand without using any unsuitable substring and primer is added.



Supplementary Figure 11: Siyuan Code Encoding Process

The decoding stage (shown in Supplementary Figure 12): First, decode all the strands, or decode the specific strands with the primer. Multiple duplicates of strands are kept to

decrease the error rate. Second, the binary slices are filtered using the hash value with the help of intra-strand error detection. Third, the slices are ordered using the index, and the index is removed. Fourth, the missing slices are recovered using the erasure code (error correcting code) with the help of inter-strand error correction. Fifth, the data are joint together.

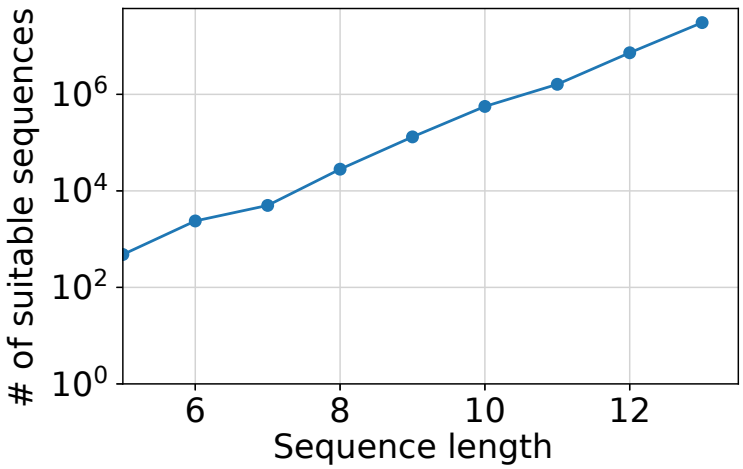


Supplementary Figure 12: Siyuan Code Decoding Process

C DNASim Simulator

C.1 Bio basis

In the previous section, our method can avoid the unsuitable error-prone strands with the optimal density. In this part, we propose a simulator for storage named DNASim. The error rates of the strands are taken into account using a random DNA error simulator in order to compare the density of the methods at the same error rate and the error rate at the same density.



Supplementary Figure 13: Suitable sequence numbers with different sequence lengths.

DNASim is based on MESA [38]. The MESA can estimate the error rate of a single input strand using the parameters. In this project, the DNASim is used in order to calculate the overall error rate of the system with error correction. The error simulator calculating the overall error rate is named as DNASim.

To simplify the problem, the DNASim has assumed that a strand cannot affect the transition probability of the other strands. In fact, the probability of a strand affecting others is negligible [38].

C.2 Formalization of multiple-strand coding system with error correction

Let the strands stored be $X_1, X_2, X_3, \cdots$ and the strands have already included the error correction strands. Note that the strands can include the unsuitable strands mentioned in the previous section. Let the measured result of the strands be $Y_1, Y_2, Y_3, \cdots$. Let the error transition be a function from the correct strands $\{X_1, X_2, \cdots\}$ and the strands with error $\{Y_1, Y_2, \cdots\}$ to the transition probability p.

$$(\{X_1, X_2, \cdots\}, \{Y_1, Y_2, \cdots\}) \mapsto p \tag{41}$$

Let the error correction system be a function that finds out the correct strands from the

strands with error with the maximal probability.

$$\begin{aligned} & \mathit{decode}(\{Y_1, Y_2, \cdots\}) \\ = \; & \arg\max_{\{X_1, X_2, \cdots\}} \Pr\left(\{X_1, X_2, \cdots\} \mid \{Y_1, Y_2, \cdots\}\right) \\ = \; & \arg\max_{\{X_1, X_2, \cdots\}} \Pr\left(\{Y_1, Y_2, \cdots\} \mid \{X_1, X_2, \cdots\}\right) \\ & \cdot \Pr\left(\{X_1, X_2, \cdots\}\right) \end{aligned} \tag{42}$$

Unfortunately, the transition probability of a strand $\Pr(Y_i|X_i)$ is unknown based on the previous researches including MESA. The MESA research only gives an estimation of the error rate of a single strand, which is $\Pr(X_i \neq Y_i)$. As a result, a strand with errors cannot give any further information that helps determine the input value without manual guess of the transition probability, which is unreliable. The DNA error simulator can simply remove the errored strands.

It is reasonable because known coding systems of DNA does not have intra-strand error correction, except for the clustering of DNA strands, which can be regarded as a single measurement with another error rate, which works with the abstraction mentioned above. Thus the errored strands can be removed in the simulator.

C.3 Error Probability Types

1. Probability of error occurrence in any of the strands

   The error occurrence in any of the strands means if any of the strands has error, then the error is taken into account to be the overall error. The event of the element-wise-or error is defined as

$$E_{\text{element wise } \cup} = \{X_1 \neq Y_1\} \cup \{X_2 \neq Y_2\} \cup \cdots \tag{43}$$

   Such error is useful in traditional cases, but not useful here. Errors in DNA coding is corrected by the error correction in the coding. A minor change would be corrected

and it would not destroy the system, so the element-wise-or error rate is not a good estimation.

2. Probability of failure in decoding

   The failure in decoding means the probability of the error correction failure, which means the decoding strand is not the source strand. The event of the failure error is defined as

$$E_{\text{failure}} = \{decode(\{Y_1, Y_2, \cdots\}) \neq \{X_1, X_2, \cdots\}\} \tag{44}$$

   Such error is the case of the DNA error, which should be simulated by the simulator.

3. Probability of failure in all strands

   The error occurrence in any of the strands means if all of the strands have error, then the error is taken into account to be the overall error. The event of the element-wise-and error is defined as

$$E_{\text{element wise} \cap} = \{X_1 \neq Y_1\} \cap \{X_2 \neq Y_2\} \cap \cdots \tag{45}$$

   Such error is not useful at all.

Among all the errors above, it is obvious that $E_{\text{element wise} \cap} \subseteq E_{\text{failure}} \subseteq E_{\text{element wise} \cup}$. And the E_{failure} is the error rate that matters, and the failure error rate is to be estimated.

A simple implementation of the error rate estimation is to simply remove the strands randomly according to the simulator. Encode, add error and decode for many times and calculate the proportion of failures. Such sampling method is too slow for high precision.

C.4 Estimation of E_{failure}

A simple but slow implementation of E_{failure} estimation is just sampling the errors. In other words, you encode, add error and decode for many times and calculate the proportion of failures. A faster and precise estimation should be introduced. In this section, an estimation of the overall failure error rate $\Pr(E_{\text{failure}})$ is given.

Let Cnt_i be 1 if $X_i = Y_i$ and 0 if $X_i \neq Y_i$ or Y_i is missing. In most cases, the decoding needs a specific number of correct strands. For example, in RS code (n, k), if there are k strands given, then the n strands can be decoded correctly. In other words, let Cnt' be the minimal number of the strands needed during decoding, which is k in RS code (n, k). Then the error event should be

$$E_{\text{failure}} = \{\sum_i \mathrm{Cnt}_i < \mathrm{Cnt}'\} \tag{46}$$

$\Pr\{\sum_i \mathrm{Cnt}_i < \mathrm{Cnt}'\}$ is hard to estimate directly and efficiently if the Cnt_i distribution varies from different i.

Let $E_1, E_2, \cdots$ be independent identically distributed random 0/1 variables, such that the sum of the error rates are the same $\sum_i \Pr\{E_i = 1\} = \sum_i \Pr\{\mathrm{Cnt}_i = 1\}$. It is obvious that

$$Pr\{\sum_i \mathrm{Cnt}_i < \mathrm{Cnt}'\} \approx Pr\{\sum_i E_i < \mathrm{Cnt}'\} \tag{47}$$

In fact, when $\Pr\{E_i = 1\} > \frac{\mathrm{Cnt}'}{n}$ is a constant independent from i, the error rate is bounded by the estimation, and the estimation is actually a binomial distribution.

$$Pr\{\sum_i \mathrm{Cnt}_i < \mathrm{Cnt}'\} \leq Pr\{\sum_i E_i < \mathrm{Cnt}'\} \tag{48}$$

If the number of strands is not too large, it can be estimated that

$$Pr(E_{\text{failure}}) \approx \sum_{0 \leq j < \mathrm{Cnt}'} \binom{n}{j} (Pr\{E_i = 1\})^j (Pr\{E_i = 0\})^{n-j} \tag{49}$$

The estimation is efficient enough for grouped error correction, where n is not larger than 1000. If n is larger than 1000000, the binomial coefficient is hard to derive, a more efficient method will be given in the next part.

C.5 Normal distribution approximation

If the number of strands is too large, the binomial distribution can be approximate to a normal distribution, and the $\Pr(E_{\text{failure}})$ can be derived from the Cumulative Distribution Function of normal distribution, and the Cumulative Distribution Function can be looked up in the CDF table of normal distribution. A more accurate approximation can be obtained by adding the variance of each Cnt_i.

$$\sum_i E_i \sim N(n \cdot Pr\{E_i = 1\},\ n \cdot Pr\{E_i = 1\} \cdot Pr\{E_i = 0\}) \tag{50}$$

C.6 Duplicate Sequencing

In the case of duplicate sequencing, a DNA strand is sequenced for more than 1 time. With the help of the intra-strand error detection, it is possible to tell whether a result of a sequencing is correct.

The probability of correct blind guess against the error detection can be ignored if the hash value is long enough. By expectation, $\sum_i \Pr\{\text{Cnt}_i = 0\}$ is the number of blind guesses. If the hash value has k bits, the expected number of correct blind guess is $\frac{\sum_i \Pr\{\text{Cnt}_i=0\}}{2^k}$, which can be rather small.

When the correct blind guess is ignored, and the error probability can be reduced exponentially by simply filtering the wrong results. If a strand is sequenced for t time, the new error rate of this strand becomes $\Pr\{\text{Cnt}_i = 0\}^t$.

D Implementation

This section is to provide an overview structure of the code. The Siyuan Code is implemented in Python. The code API is named the BioSys. BioSys can encode the strands in an end-to-end way. The BioSys uses 3 main parts of submodule. The Single-Strand Trie Coder named Trie, the inter-strand error correction named erasure_code, the indexing and intra-strand error detection named safe_hash_algo, the strand evaluation named simulator, the controller named dna_seq and the config management.

D.1 Single-Strand Trie Coder

The Single-Strand Trie Coder is implemented in the Trie module. The data structures of the automaton include the Node and the Trie tree with press value. This part is to provide the function of converting between an integer and a strand. The 'add strand' is to add an unsuitable strand into the strand list. The 'calculatePress' is to calculate the press edge in the Trie coder. The 'get_index' and 'get_strand' are the 2 main functions converting between index and strand. The automaton is constructed and the coding is implemented using the algorithm described in §4. This part is the main contribution of our work.

D.2 inter-strand error correction

The inter-strand error correction is implemented in the erasure_code module and is implemented based on Python error correction libraries. By default, the error correction is to use the RS code method to give the error correction function.

In the standard RS code, k bits can be recovered from n bits. The $n-k$ bits are the redundancy. The standard RS code can correct at most $n-k$ errors. In this problem, the inter-strand error correction picks k strands and adds $n-k$ redundancy strands. The i^{th} bits of the n integers presenting the n strands can be thought the same as the n bits in the standard RS code.

D.3 Reed–Solomon Erasure Code

The RS code module in the system uses the standard pip library 'reedsolo' to implement the function of RS code. Actually, the module can be easily replaced by any other error correction code or erasure code.

D.4 The indexing and the intra-strand error detection

The indexing and the intra-strand error detection is implemented in the safe_hash_algo module. The indexing is very trivial, adding a binary string presenting the index of the binary string. The hash is implemented using the python standard hash library. The method takes

some bits in the SHA-1. The bits are to examine whether the binary string is correct.

D.5 Strand evaluation

The strand evaluation is implemented in the simulator module. The module evaluates the DNA strands and calculates the error rate of each single strand based on the MESA (https://github.com/umr-ds/mesa_dna_sim). Also, the module helps calculate the system decoding error rate with the single strand error rate based on the method mentioned previously in our paper.

D.6 Multi-Strand Controller System with Error Correction

The BioSys in the dna_seq module is to control the modules mentioned above to achieve the function of end-to-end Siyuan Code coding with error correction between multiple strands and a binary file.

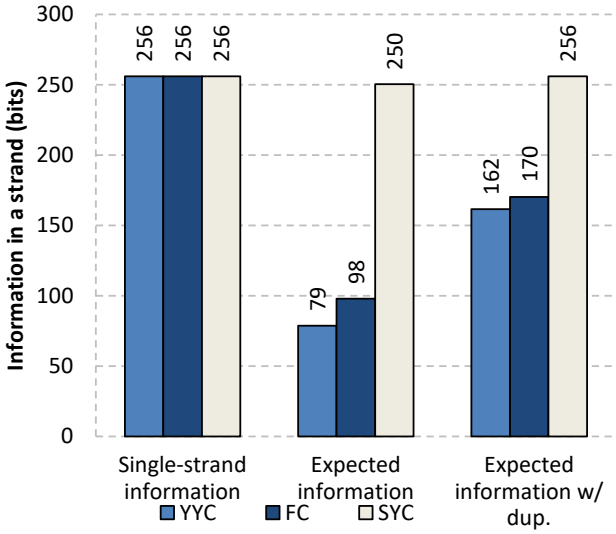
The encoding part receives the config and the input file as input. The decoding part receives the config and the DNA strands as input. The config includes the good strands, the binary length in partitioning, the hash length, the index length, the RS code k and n value and the file length as input.

There is an example of config management in the 'tests' folder. It helps operate the test_with_simulator and feeds the config into it. Also, it helps in evaluating the Strands generated by the BioSys.

E Misc

E.1 In-silico comparison of per-strand information

The strict constraint does not hurt the information density. Here we analyze the amount of information of a strand. Supplementary Figure 14 displays the per-strand information density of different encoding methods for strands of 150 nucleotides, including indexing and hash values. As shown in the figure, Siyuan Code, Yin-yang Code, and Fountain Code each have the capacity to encode 256 bits of information within a single strand. However,



Supplementary Figure 14: The information contained in a single DNA strand of length 150.

not all strands can be decoded successfully, necessitating an adjustment of the expected information yield to account for the single-strand error rate, ϵ. Therefore, the effective information content recoverable from a strand is $256 \cdot (1 - \epsilon)$ bits.

The figure reveals that, on average, a single strand encoded by Siyuan Code can be expected to yield about 250 bits of information, factoring in the error rate. By comparison, the expected information retrievable from a single Fountain Code strand is around 98 bits, and from a single Yin-yang Code strand, approximately 79 bits.

E.2 Number of copies in synthesis

The copy number of the synthesis is highly related to the synthesis cost, storage cost and the physical density of the DNA storage system. In the synthesis step, multiple copies of a strand are synthesized in order to ensure at least one correct copy of each strand. However, the storage cost increases linearly with the number of copies, while too few copies before PCR amplification leads to a high error rate in synthesis. Storage cost refers to the expense of leasing adequate long-term infrastructure to house future large-scale DNA storage powder. The synthesis cost increased slowly with number of copies if PCR was performed before storing, but it is still wasting the escalating space cost. Advancing biotechnologies will precipitate a steep decline in DNA synthesis costs, whereas non-renewable spacetime

resource expenditures will undergo persistent appreciation.

Let the total number of strands be N and let the number of copies of a strand in synthesis be k. In the example of Siyuan Code, k is chosen to be 6 and N is chosen to be 1253, with 42% redundancy in RS Code which is not used out. According to the result, a redundancy of 7.9% is needed.

Our method can work with a very low number of copies during the synthesis step, which can save a lot in synthesis and storing costs. With our method, the number of strands to store in storage can drop dramatically by $\sim 200\times$. The number of copies drops from ~ 1000 in Fountain code to 6 in Siyuan Code.

E.3 Reproducing the results

The synthesis and sequencing services are supported by GenScript. The equipment relative to the DNA microchips [42] is shown in Supplementary Figure 15.

The source code is publicly available at https://github.com/DNASiyuanCode/SiyuanCode and https://gitea.sjtu-edu.top/SiyuanCode/SiyuanCode, and has been deposited in Code Ocean 10.24433/CO.4952245.v1 [43].



Supplementary Figure 15: Genscript equipment



Supplementary Figure 16: Data stored in vitro

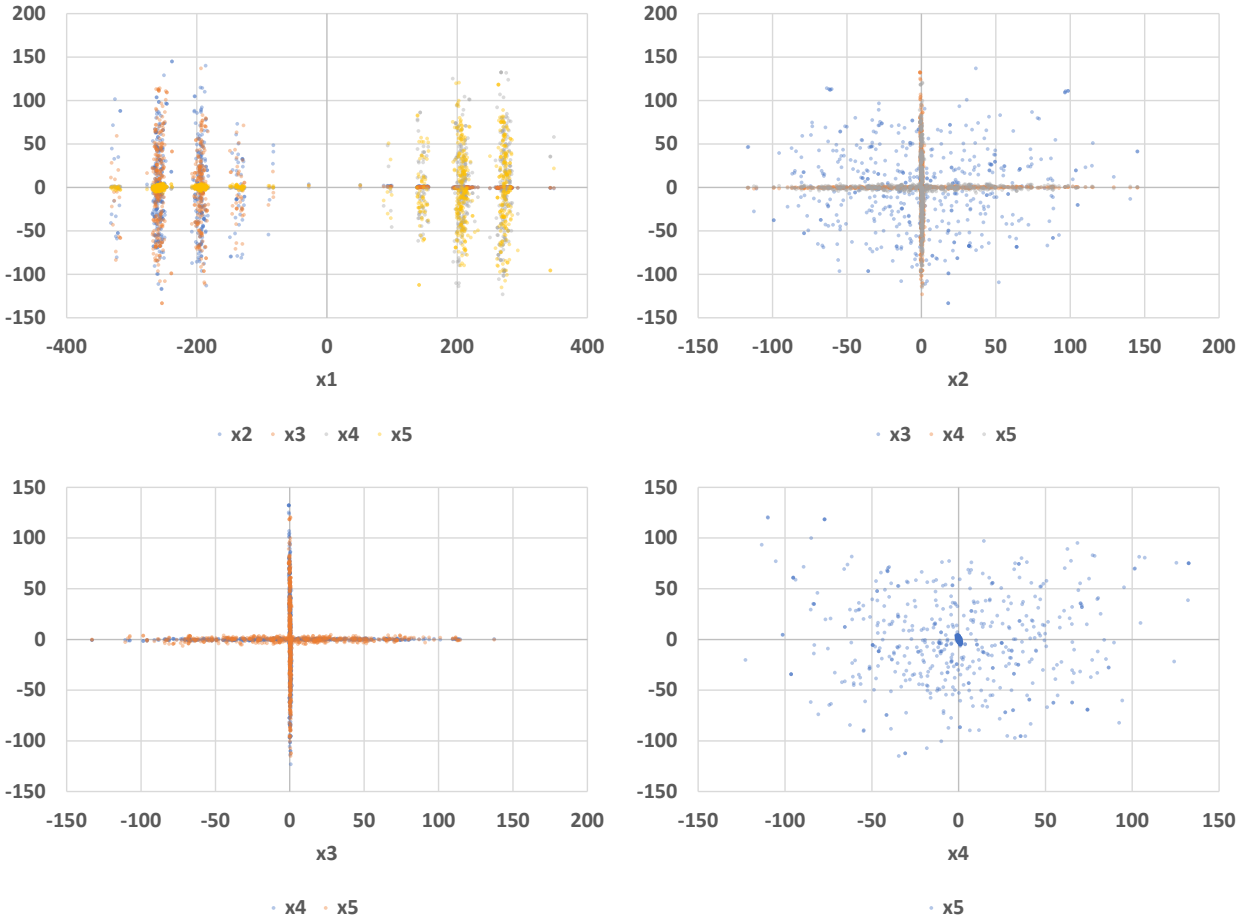
The in-vitro experiments are described as follows. The data is first compressed using 7-zip (seen at **tests/samples/SJTU_zip/samples3.7z**), and then encoded by the program (seen at **tea_bio_06.py**). The environmental requirements can be seen at **README.MD**. The data (86808 bytes before compression) includes the 4 files shown in Supplementary Figure 16.

After synthesis, the DNA strands are dried in order to store the information. The sequenced result should be input to PEAR [44] and then to the decode program (seen at **tea_bio_06_dec.py**). The sequencing data are publicly available at https://github.com/DNASiyuanCode/SiyuanCodeSequencing and https://gitea.sjtu-edu.top/SiyuanCode/SiyuanCodeSequencing, and have been deposited in Zenodo 10.5281/zenodo.21357059 [45].

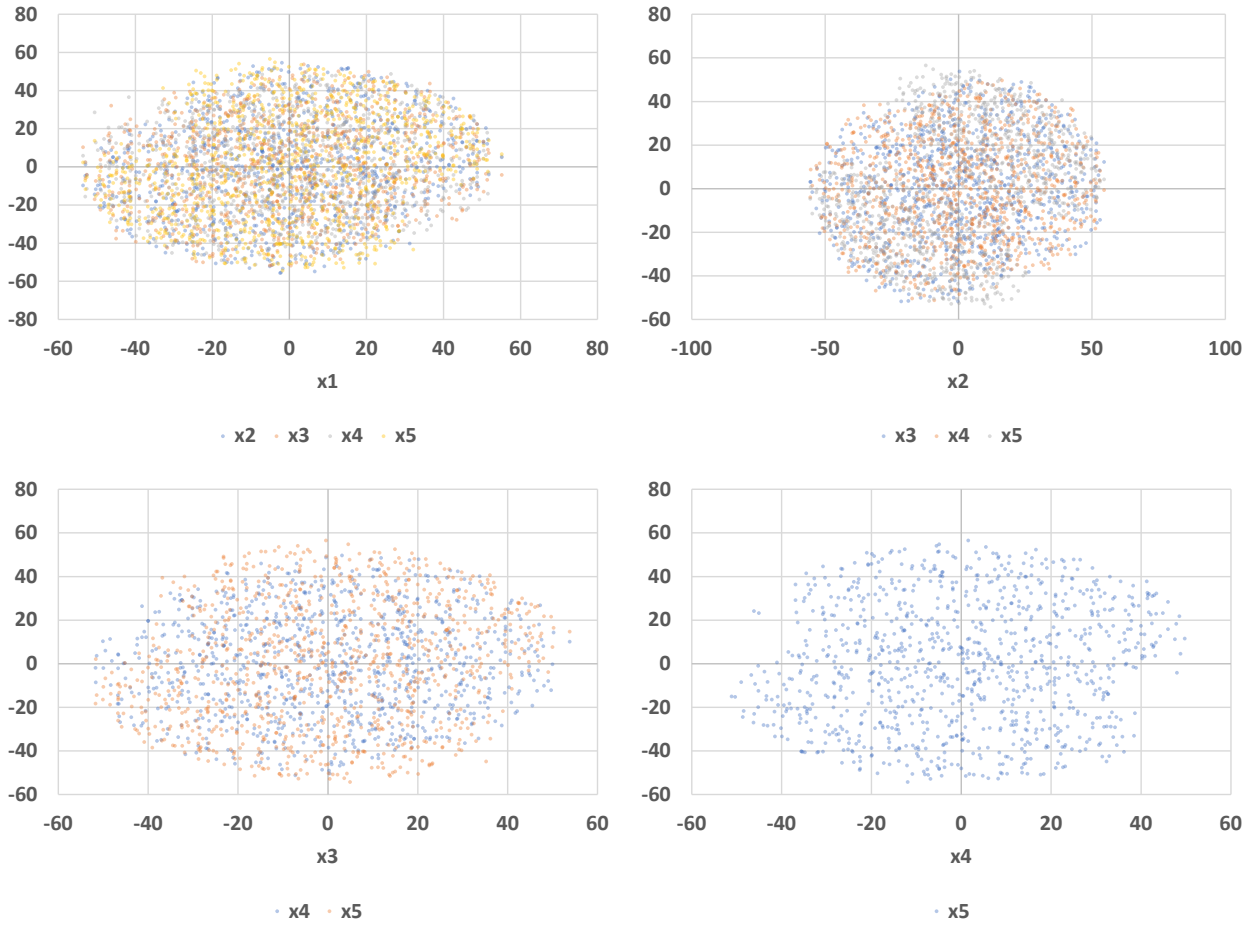
E.4 Distribution map

The DNA strands in vitro are sequenced. The sequencing result can be seen as a set of strings, each of which presents a DNA strand. Each string can be regarded as a point in a high-dimensional space. The strings are dimensionally reduced by the means of ISOMAP [46] and MDS [47]. The distribution of the DNA strands after dimension reduction $(x1, x2, x3, x4, x5)$ is shown by Supplementary Figure 17 and Supplementary Figure 18 in the form of scatter diagrams. The ISOMAP and the MDS dimension reductions are performed on a random sample with size 1000 to give a sense of the distribution of the sequenced in-vitro DNA

strands from Siyuan Code.



Supplementary Figure 17: The distribution map of the ISOMAP dimensional reduction of the high-dimension DNA strands

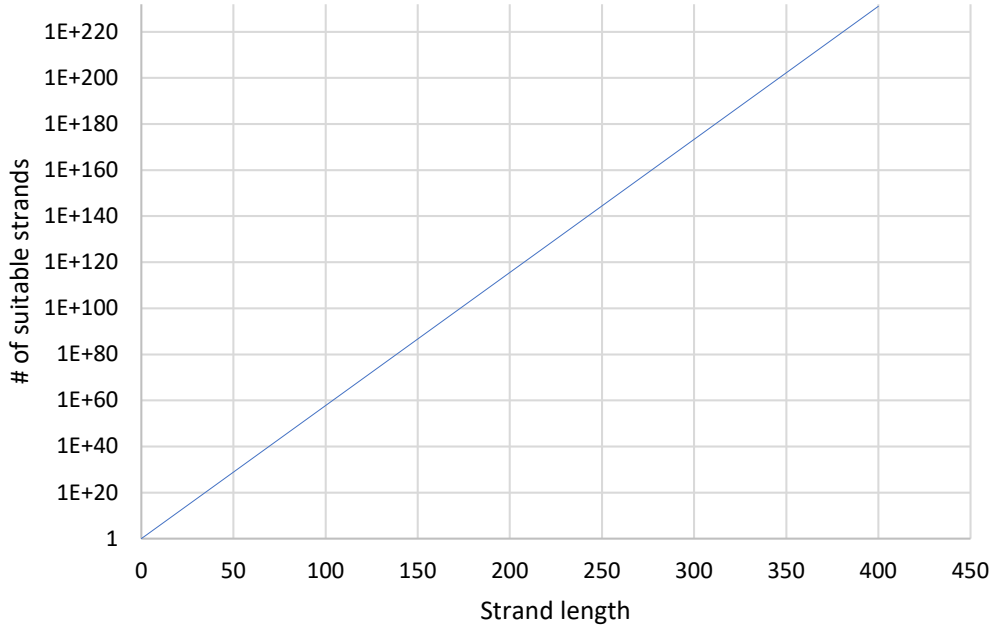


Supplementary Figure 18: The distribution map of the MDS dimensional reduction of the high-dimension DNA strands

E.5 Data Scale

There is no upper limit to the data scale of our method when the strand length is not limited. As shown in Supplementary Figure 19, the number of available paths is exponentially large compared to the length of the strands. Our coding method does not limit the length of the strands, so our method is compatible with future settings where the length and the strand number are much larger.

Even if the strand length is limited, the number of available strands is still astronomically large, rendering any concern about depletion unfounded. To take our parameters of the in-vitro experiment as an example, we use a very strict constraint to avoid the typical errors



Supplementary Figure 19: Suitable strand numbers with different strand lengths: The empirical function of the number of suitable strands is about 3.783^n where n is the strand length under the setting and the parameters of our in-vitro experiments (very strict constraints)

mentioned by MESA [38]. If the strand length is 134 (excluding the primers), there are as many as 2.65×10^{77} different suitable strands. If the strand length is 200 (excluding the primers), there are as many as 3.65×10^{115} different suitable strands. This means that even with very short strands (134 nt), we can store very much data theoretically. Assume the hash length is h bits, if there are $|S|$ suitable strands and we want to store data with m suitable strands and the adaptive index length becomes $\log_2 m$ bits, each strand can store about $\log_2 |S| - \log_2 m - h$ bits. The system can store $\frac{m(\log_2 |S| - \log_2 m - h)}{8}$ bytes. With $|S| = 2.65 \times 10^{77}$, $m = 10^{26}$, $h = 8$, the system can store 2.0×10^{27} bytes of data, which is 1.8×10^9 EByte. With $|S| = 2.65 \times 10^{77}$, $m = 10^{74}$, $h = 8$, the system can achieve its limit of 4.2×10^{73} bytes under the setting of 134 nt short strands.

We do not think that it is practical to conduct tests in order to test the limit of our coding method because the limit of our coding method is too large. Larger tests would not be beneficial because the reliability can be shown with smaller datasets and it is not easy to tell how large is "large enough" for the "too large" limits.

E.6 Scoring criteria for density and copy number

Density considering copy number is important because some previous work can play trick by using very large copy number during the storage step to bridge the gap in single-strand error rate between the two methods. The concept of density and copy number is important in physical density comparison, but the copy numbers during storage and during sequencing were deliberately conflated before our work. After synthesis (the synthesis step can include PCR amplification or not), there are multiple copies of each strand in the container. The first copy number is the copy number in the container, and this copy number influence the space and weight cost of storage. During sequencing, the multiple copies can be PCR amplified again to be sequenced. The second copy number can be regard as a part of the sequencing step, which does not influence the density of the storage step.

The storage density of DNA impacts the long-term physical storage costs, a significant concern as space is projected to become increasingly scarce in the distant future. In contrast, the sequencing density (i.e., the density of PCR-amplified DNA loaded into sequencers) primarily affects sequencing costs, which are now lower than the synthesis cost and are expected to drop substantially in the future. Furthermore, there is the future potential to reuse the nucleotide bases after sequencing in the distant future.

We think that the scoring density of DNA storage should correspond to the first copy number in the storage step. The PCR amplification during sequencing can be considered as a revision in sequencing to increase the sequencing accuracy but not a revision in storage which occupy much mass and volume during storing. We have only 6 copies in our in-vitro experiments in the storing step, and they are then PCR amplified again in the sequencing step.

E.7 Criteria for robustness against excessive errors

The focus of our work is not to give the criteria for robustness or to give an extreme robust coding method. The robustness can be regarded as an additional benefit of our one-to-one coding. Our method overcome the excessive errors with error avoidance and user-defined high-redundancy error correction code. The robustness of previous works in Table 1 was

given by Yin-yang code [11].

As to the criteria for robustness, we regard it as the ability of 100% data recovery with high-redundancy error correction, no matter how high the strand error (strand erasure) is. Also, the method should have the ability to use up the redundancy, instead of wasting some of the redundancy. At the same time, the method should work with strict constraints dealing with high error rates. Our basis is that modern digital files, such as contemporary photographic images, are invariably subjected to either lossy or lossless compression algorithms, where a bit changing can lead to file destruction. Therefore, we argue that partial data recovery during decoding is useless in data reconstruction.

According to the criteria, Fountain code is not robust. With strict filters, Fountain code could fail to find a suitable strand. Also, the randomness of Fountain code makes it impossible to use up the redundancy. The random matrix could be not full-ranked. Though Fountain code can perform 100% data recovery, it can not fulfill the other two criteria.

E.8 Analysis of the Underlying Causes of Disadvantages in Filtering-Based Methods

In this section, we will provide a quantitative approach to explain why filtering-based encoding is inefficient and unstable.

Filtering-based methods randomly generate strands and retain low-error ones using a filter. This process repeats iteratively until a sufficient number of strands is obtained. This regenerative step incurs exponential time cost. We will demonstrate this through theoretical algebraic analysis and experimental testing.

E.8.1 Analysis of inefficiency

To be formal, without loss of generality, filtering-based methods randomly generate a strand with the random seed inside the strand, and record the hash value into the strand as error detection. Let the random variable of the whole strand s be drawn from the probability distribution D_{gen}. $s \sim D_{gen}$. Let $\mathcal{B} = \{A, T, C, G\}$. Let the strand length be $L = m$. Let the suitable strand set be $\mathcal{S}$.

The probability distribution D_{gen} depends on the seed, the strand generator and the hash function, but the property of the filtering-based methods depends on the $\Pr_{s\sim D_{gen}}(s \in \mathcal{S}) \approx \Pr_{s\sim \text{Uniform}(\mathcal{B}^m)}(s \in \mathcal{S})$. In other words, the strand randomly generated by the filtering-based methods should closely approximate a truly random strand. So, it is reasonable to assume $\Pr_{s\sim D_{gen}}(s \in \mathcal{S}) = \frac{|\mathcal{S}|}{4^m}$. To be convenient, we assume that $s \sim \text{Uniform}(\mathcal{B}^m)$ in the following analysis.

If homopolymer length L_H is forbidden in the strand. We want to estimate the bound of the probability that a randomly generated strand is suitable. Let the probability be $P_{\mathcal{S}}$, and then the expected generating and regenerating time is $\frac{1}{P_{\mathcal{S}}}$.

Divid the L bases into $\frac{L}{L_H}$ groups and each group consists of L_H bases. The probability of filtering accepting a strand is not greater than the probability that each group is not a homopolymer. As a result, we can have a rough bound $P_{\mathcal{S}} \leq (1 - 4^{1-L_H})^{\frac{L}{L_H}}$. $\frac{1}{P_{\mathcal{S}}} \geq (\frac{1}{1-4^{1-L_H}})^{\frac{L}{L_H}}$. According to the analysis, we can already say that the regenerating time complexity is exponential with respect to the strand length L.

E.8.2 Analysis of instability

Another concern about the filtering-based method is that the random seed may be used up. About $\frac{1}{P_{\mathcal{S}}}$ distinct strands should be generated, so the number of distinct random seeds should be at least exponential with respect to the strand length L. In other words, the random seed length in each strand should be at least linear to the strand length. Also, the needed random seed length could be very long with more strict constraints (smaller L_H).

However, most filtering-based methods do not have a user-specifiable length of random seed. As a result, the encoding program of these method may crash or be trapped in an infinite loop after using up all the random seeds if the constraint is strict or the length is long.

Some methods do not explicitly mention a random seed, but the encodings they use typically contain an implicit random seed, and the aforementioned issue of random seeds cannot be resolved.

For example, fountain code [48, 12] needs to store the random indices of the blocks in each packet to successfully decode the file. The random indices can be regarded as a random

seed. Although a shorter random seed can generate the random indices, the randomness still needs to be stored as a random seed. And the random seed of each strand needs to be long enough to ensure that the number of distinct random strands is large enough.

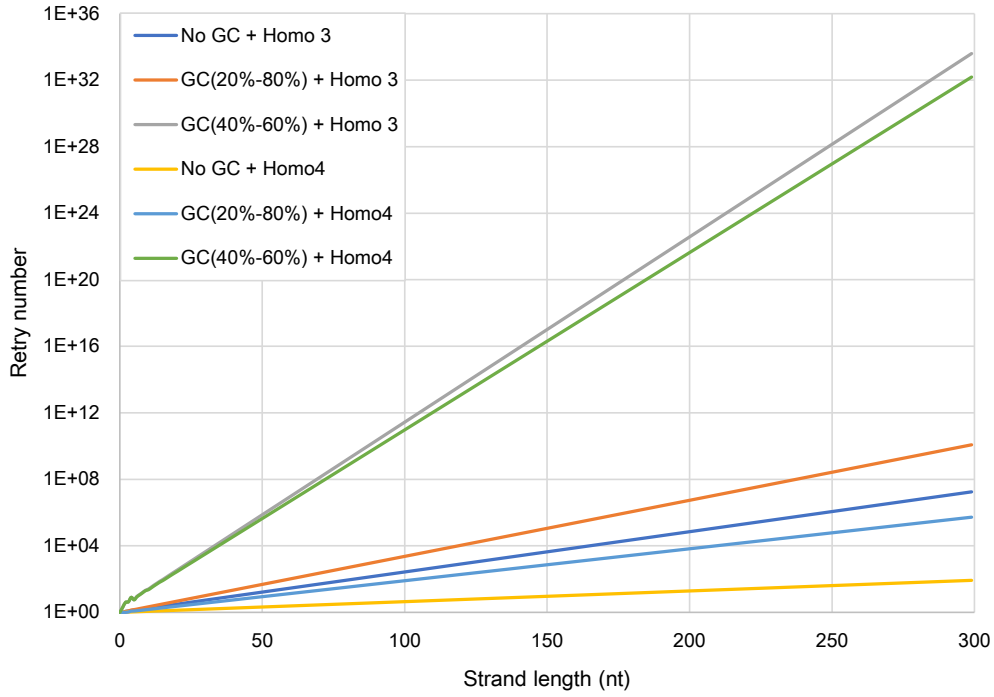
For example, yin-yang codec [11] needs to store 2 indices in each strand. The extra strand index can be regarded as the random seed. When the constraints are strict and the file is not too large, the situation that the possible pairs are used up in the regenerating can happen.

E.8.3 Tests of inefficiency

The real-world case is more complicated and tighter than the rough bound. We should consider not only the homopolymer, but also other error-prone patterns. Our simulation experimental testing can help giving a more precise estimation of the regenerating time taking all the constraints into consideration.

The setting of our tests are based on the following constraints: homopolymer, GC content and undesired motifs. The detailed undesired motifs are from MESA [38]. The homopolymer constraint and the GC content constraint vary in this test. As shown in Supplementary Figure 20, the number of retries of filtering-based methods is roughly an exponential order of magnitude that of strand length.

According to Supplementary Figure 20, with strand length controlled, the retry number is highly associated with the GC content constraint and the homopolymer constraint. The "No GC" in Supplementary Figure 20 means that there is no constraint in local GC content. The "GC($a\% - b\%$)" means that the local GC content ratio is between $a\%$ and $b\%$ with local window length 5. "Homo a" means that the homopolymer of length a is forbidden. At strand length 200, the too loose constraint "No GC Homo 4" needs about 19 retries for each strand in average, when the strict constraint " GC(40%-60%) Homo 3" needs about 3.8×10^{22} retries for each strand in average, which is too time consuming in the encoding process. The appropriate constraint "GC(20%-80%) Homo 3" needs about 5.5×10^{6} retries for each strand in average. The checking of the constraints in each retry has a time complexity of not less than $O(L)$, which means that each retry may cost at least 6.7×10^{-6} seconds on a computer equipped with a 3GHz CPU if the constant factor is at least ~ 100. That is to say, each strand may cost nearly a minute or more to generate, because of the unacceptably high retry



Supplementary Figure 20: Number of retries of filtering-based methods

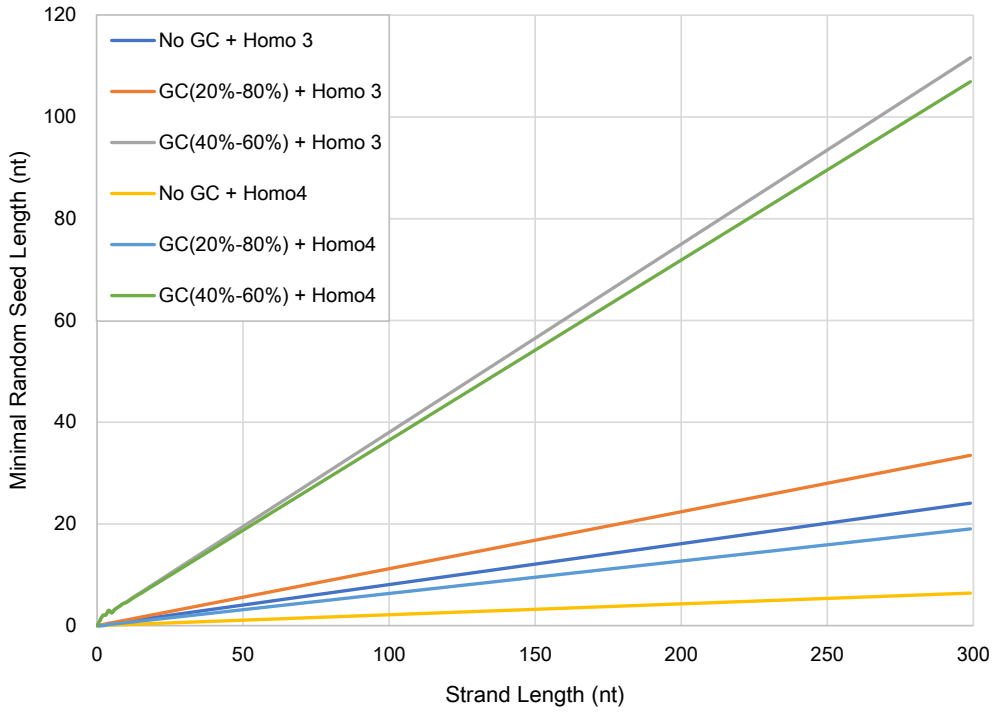
number.

Such inefficiency is one of the reasons why the existing filtering-based methods use very loose constraints (Homo 5 forbidden etc.), while too loose constraints may lead to a high error rate and pose an obstacle to copy number reduction. Also, the high retry number may be a impediment to increasing the length of strand in vitro in the future. Certain straightforward modifications, for instance, segmentation in filtering-based methods, would undermine the whole-strand characteristics and incur greater fundamental overhead, for instance, more addresses and more random seeds.

E.8.4 Tests of instability

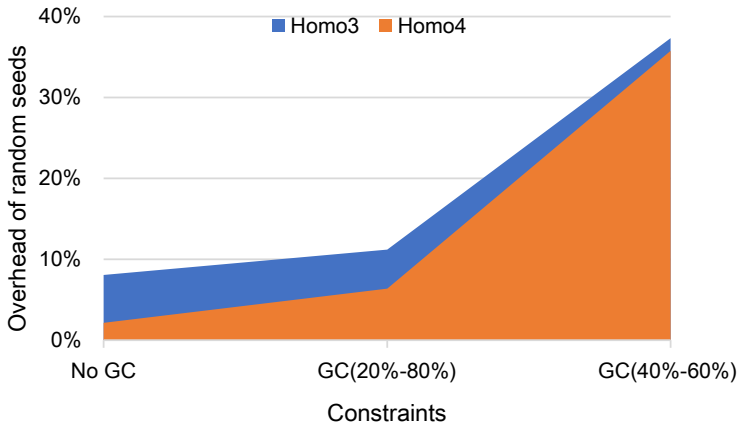
The number of distinct random seeds should be at least the number of retries. Otherwise the random seeds could be used up. The random seed length should be at least $\log_2 \frac{1}{P_S}$ bits or $\log_4 \frac{1}{P_S}$ bases, where $\frac{1}{P_S}$ is the expected retry number. In case of randomness, a safer choice for the minimal random seed is to double the random seed length, as shown in Supplementary Figure 21.

Our simulation experimental testing can help estimate the minimal random seed length



Supplementary Figure 21: Minimal random seed length

necessary for each strand according to previously calculated expected retry number. The settings are the same of those in Supplementary Figure 20. By showing that the minimal random seed length is long, we can determine that filtering-based methods not only suffer from low encoding efficiency but also have a significant drawback: they are prone to exhausting the random seed and failing to encode correctly with strict constraints due to the short random seed length used in practice. A strand with length 200, the "GC(20%-80%) Homo 3" constraint needs a random seed length of more than 22.4 bases for each strand to ensure that distinct random seeds are not used up, while fountain code uses a random seed length of 16 bases. The filtering-based methods would fail in encoding and show its instability with strict constraints, as shown in Figure 3e.



Supplementary Figure 22: Overhead of the random seeds

An inelegant solution to this instability of filtering-based methods is to increase the length of the random seed. However, the overhead of the increased random seed can be up to 11.2% under "GC(20%-80%) Homo 3" constraint and up to 35.8% under "GC(40%-60%) Homo 4" constraint, as shown in Supplementary Figure 22. At the same time, yin-yang codec uses an implicit random seed by encoding the seeds into the index, which means that yin-yang codec can never increase its random seed length. We give an example of how to make the encoding process of yin-yang codec fail in Figure 3e.

E.9 Incremental redundancy generation property with RS Code

This part is to simply explain the fountain's incremental redundancy generation property implemented with RS Code.

Traditional Reed-Solomon (RS) codes have a fixed erasure correction capability, whereas RS codes applied to DNA storage can achieve an expandable error tolerance. Specifically, by applying RS codes across DNA strands—treating each strand as a symbol in the code—we can initially generate a certain number of redundant strands with higher redundancy. Note that not all redundant strands are generated at first. They can be generated on demand. Since each strand contains its own checksum, it is straightforward to verify whether it has been stored correctly. If some strands are corrupted or lost, additional redundancy can be dynamically appended. Note that the incremental redundancy generation is practical because a large number of redundant strands have not been generated yet and the RS Code

redundancy is very high. Correct decoding is guaranteed as long as the total number of correctly retrieved strands—including both data and redundant strands—is no less than the original number of data strands. From this perspective, the ability to incrementally add redundancy makes RS codes more efficient and waste-free compared to Fountain codes. Fountain code needs ϵ times more strands than the data strands. A defining parameter of Fountain codes is ϵ, which represents the fraction of extra strands required beyond the minimum number for decoding.

The reason traditional RS codes were not designed this way is that, in conventional usage scenarios, it is often difficult to simply determine whether a particular symbol is correct. Moreover, such a modification would require additional metadata to record the position of each symbol, incurring significant overhead. In contrast, in the new context of DNA storage, every strand already includes a checksum and an address index, so this adaptive design introduces no additional overhead.

E.10 Different densities of Siyuan Code

The density of Siyuan Code varies depending on the specific definition used. To avoid ambiguity, we will introduce and calculate the various density definitions in this section.

The value of 41 EB/g reported in our manuscript is the final result that already includes the logical redundancy and the copy number necessary for achieving robust, perfect retrieval, allowing for a direct, "apples-to-apples" comparison with the effective densities reported in works like DNA Fountain. It is not a theoretical upper limit without redundancy.

Our redundancy includes logical redundancy and copy number redundancy. The density with constraints but without redundancy is $\frac{1.920\times6.022\times10^{23}}{308.95\times8\times1024^6} \approx 405.8$ EB/g. The $1.920 = \log_2 3.783$ bits/nt is the information density with constraints but without redundancy. The 3.783 is given in Supplementary Figure 19. The 6.022×10^{23} is the Avogadro Constant. The 8×1024^6 bit/EB is the unit conversion. The 308.95g/mol is the Molar Mas. Though 405.8 EB/g is more eye-catching, the 405.8 EB/g density does not take the copy number, the indices and the redundancy into consideration, and thus we did not use the 405.8EB/g to describe our in-vitro density.

Our density with constraints, indices, error correction and logical redundancy is based on

the in-vitro wet experiments. We stored 86808 bytes in vitro. The files were compressed into 31127 bytes. There are 1253 different strands of length 170 (including the primer, the error correction and the indices) in all. The Molar Mass of each strand should be $79+308.95\times170 = 52600.5$ g/mol. The weight of the strands should be $\frac{1253\times52600.5}{6.022\times10^{23}} \approx 1.0945 \times 10^{-16}$ g. Thus, the density with constraints, indices, error correction and logical redundancy should be $\frac{31127}{1.0945\times10^{-16}} \approx 2.8439 \times 10^{20}$ byte/g, which is 246.673 EB/g. If we consider the compression as a part of our system, the density should then be $\frac{86808}{1.0945\times10^{-16}} \approx 7.9313\times10^{20}$ Byte/g, which is 687.93 EB/g.

Our method's in-vitro experiment has a copy number redundancy of exactly 6 in the Storage Stage in Figure 1. Our density considering the copy number should be $\frac{246.673}{6} \approx 41.112$ EB/g. To avoid unfairness, we use the 41 EB/g instead of the eye-catching 688 EB/g.

E.11 Encoding and decoding time of very long strands

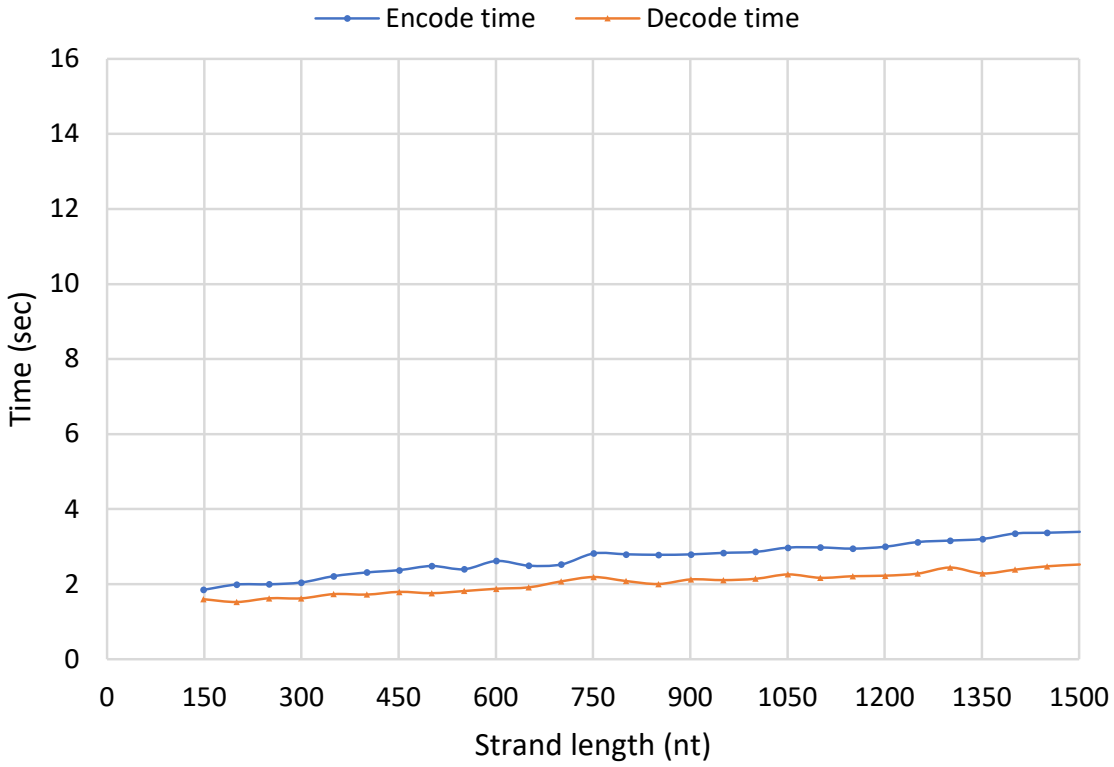
While we have previously demonstrated that our method exhibits time complexity of linear calculations, we recognize that readers may still have reservations regarding its computational efficiency. To address potential concerns about scalability, this section provides empirical evidence that our method can efficiently handle very long DNA strands.

For the theoretical analysis of time complexity, see Supplementary §B.

In order to show that our method is not highly influenced by the strand length and the constraints, we have already given Figure 3bc. The strand length of the figure is from 150nt to 300nt. However, it does not mean that our method can not handle longer strands. The strand is not longer than 300nt to ensure that Yin-yang Codec and Fountain Code does not crash.

To prevent any misinterpretation, Supplementary Figure 23 is provided. In this figure, Siyuan Code is run with different strand length and the same constraints. The strand length is much longer than 300nt. The constraints includes MESA constraints [38] explained in Supplementary §B.1.2 and the maximized allowed homopolymer length is 2. We would like to say that the constraints are strict (slower constraints). To avoid accusations of selectively choosing data, the ∼2MB file given by Fountain Code [12] is encoded and decoded. In

this figure, it is clear that even under very strict constraints, the running speed is still very high. Although the running time appears to increase with DNA strand length, the growth is gradual and near-constant. It is reasonable to argue that the non-constant portions of these timings are attributable to specific Python interpreter implementation details and computer architecture, rather than the algorithmic time complexity itself. In summary, we argue that concerns about the time complexity exploding with length are unfounded.



Supplementary Figure 23: Siyuan Code running time with different strand length.

Siyuan Code's superior coding/decoding speed is not merely a software performance metric; it has profound practical implications for DNA storage system design. Unlike traditional filter-based encoding methods, whose time complexity can grow exponentially as constraints become stricter or strand lengths increase, often forcing the adoption of relaxed, suboptimal parameters, our algorithm ensures efficient operation even under stringent combinatorial constraints. As illustrated in Figure 3c, while conventional approaches may become computationally intractable, Siyuan Code maintains a stable and feasible encoding/decoding process, enabling the use of stricter error-avoidance rules without compromising practical

usability. Thus, the computational advantage translates directly into greater design freedom and enhanced physical reliability of the encoded DNA strands.

E.12 Preprocessing time and space

Supplementary §B has already given the time complexity and the space complexity analysis of the preprocessing, encoding and decoding process. However, we did not give the exact in-silico space consumption and time consumption of the preprocessing stage. In this section, we will review the theorems and the conclusions from Supplementary §B and measure the exact runtime and memory usage of the preprocessing stage.

E.12.1 Review of the conclusions and theorems from Supplementary §B

In this context, n denotes the length of the strand length for encoding and decoding. L denotes the total length of the constraints. L_{file} denotes the length of the file. For rigour, we assume that operations on integers and their storage are $O(1)$ and treat them as atomic.

As proved in Supplementary §B, the preprocessing time complexity is $O(nL)$, the preprocessing space complexity is $O(nL)$. The bottleneck is not the $O(L)$ time trie automaton construction, but the DP table of size $O(nL)$.

The encoding time complexity is $O(n)$. The encoding space complexity is $O(n)$. Note that the encoding time and space complexity is the complexity of encoding one strand but not one file. Note that the space complexity excludes the preprocessing space complexity which is always stored in the memory.

Theorem E.1. *When considering the encoding of an entire file rather than a single DNA strand, the time complexity becomes $O(L_{file})$.*

Proof. The reasons are as follows: each strand can store $O(n)$ bits or bytes of the information, so there should be $O(\frac{L_{file}}{n})$ strands. Each strand encoding has a time complexity of $O(n)$, so the file encoding has a time complexity of $O(L_{file})$. □

Theorem E.2. *The file encoding space complexity is $O(n)$.*

Proof. When analyzing space complexity, the theoretical model could employ a pipelined approach, which reads, computes, and writes data in a streaming fashion, resulting in a streaming algorithm whose space cost matches that of processing a single strand. Therefore, the file encoding space complexity is $O(n)$. □

However, our current python implementation reads the entire file into memory, performs all computations, and then writes the results, leading to a space complexity of $O(L_{file})$. This design choice was made for three reasons:(1) The memory footprint of our method remains modest in practice, allowing it to run reliably without complex modifications. (2) We previously did not emphasize low space complexity as a distinct advantage of our approach. (3) In our in-silico experiments, we prioritized runtime performance over memory usage and therefore did not pursue memory-specific optimizations.

The complexity analysis for the decoding process largely mirrors that of the encoding process. Accordingly, we present its complexity here. The entire file decoding time complexity is $O(L_{file})$. The entire file decoding space complexity is $O(n)$. Our python implementation file decoding space complexity is $O(L_{file})$.

The complexities are listed in Supplementary Table 1. The calculation of "Preprocessing+Encode" and "Preprocessing+Decode" complexities are trivial with each stage complexity is given.

E.12.2 In-silico preprocessing time and space

We have already given the in-silico encoding time and decoding time previously. We also have already given the mathematical proofs in Supplementary §B. Here, we give the preprocessing time and space under different settings. When considering a streaming algorithm from a theoretical perspective, the total memory usage during encoding and decoding is essentially equivalent to that of the preprocessing stage, with negligible additional overhead. The non-streaming implementation reads the entire file at once, so part of the memory consumed during encoding/decoding scales with the file size. To isolate the effect of file size on memory measurements, we report only the memory usage of the preprocessing phase.

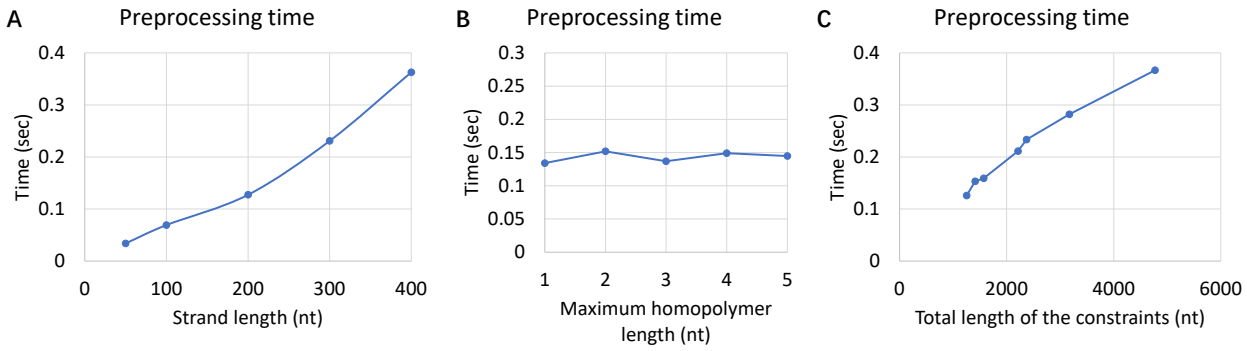
Generally, the strand length, the maximum homopolymer length and the total length of

Stage	Sub-stage	Time complexity	Space complexity
Preprocessing	Trie automaton construction	$O(L)$	$O(L)$
	DP value calculation	$O(nL)$	$O(nL)$
Encode	Single strand	$O(n)$	$O(n)$
	Entire file (streaming)	$O(L_{file})$	$O(n)$
	Entire file	$O(L_{file})$	$O(L_{file})$
Decode	Single strand	$O(n)$	$O(n)$
	Entire file (streaming)	$O(L_{file})$	$O(n)$
	Entire file	$O(L_{file})$	$O(L_{file})$
Preprocessing+Encode	Single strand	$O(nL)$	$O(nL)$
	Single strand (amortized)	$O(n)$	$O(nL)$
	Entire file (streaming)	$O(L_{file}+nL)$	$O(nL)$
	Entire file	$O(L_{file}+nL)$	$O(L_{file}+nL)$
Preprocessing+Decode	Single strand	$O(nL)$	$O(nL)$
	Single strand (amortized)	$O(n)$	$O(nL)$
	Entire file (streaming)	$O(L_{file}+nL)$	$O(nL)$
	Entire file	$O(L_{file}+nL)$	$O(L_{file}+nL)$

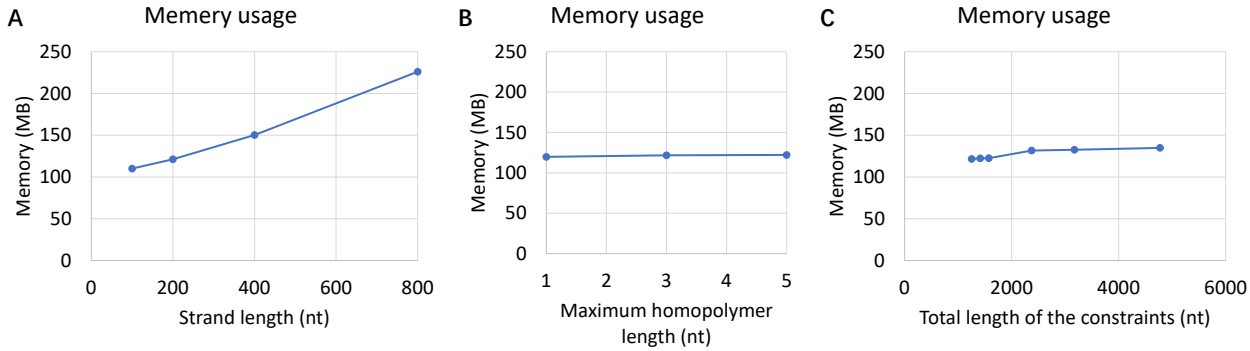
Supplementary Table 1: The table of time complexity and space complexity.

the constraints have the probability to influence the preprocessing time. We show whether these settings influence the preprocessing time or not in Supplementary Figure 24. In Supplementary Figure 24a, the total length of the constraints is 1252 and the maximum homopolymer length is 2. As shown in Supplementary Figure 24a, the preprocessing time scales linearly with the strand length, which aligns with the time complexity in Supplementary Table 1. In Supplementary Figure 24b, the total length of the constraints is 1252 and the strand length is 200. As shown in Supplementary Figure 24b, the preprocessing time is dependent with the maximum homopolymer length, which aligns with the time complexity in Supplementary Table 1. In Supplementary Figure 24c, the strand length is 200 and the maximum homopolymer length is 2. As shown in Supplementary Figure 24c, the preprocessing time scales linearly with the total length of the constraints, which aligns with the time complexity in Supplementary Table 1.

It is worth noting that our preprocessing method is implemented in Python without specific optimizations for efficiency, yet it remains fast in practice. Our implementation allows the preprocessing strand length to be set slightly longer than the actual storage strand length, providing flexibility in adjustment.



Supplementary Figure 24: Siyuan Code preprocessing time with settings. **(a)** Preprocessing time under varying strand lengths (nt). **(b)** Preprocessing time under varying maximum homopolymer lengths (nt). **(c)** Preprocessing time under varying constraints lengths (nt).



Supplementary Figure 25: Siyuan Code preprocessing memory usage with settings (including the 99.63 MB baseline memory consumption). **(a)** Memory usage under varying strand lengths (nt). **(b)** Memory usage under varying maximum homopolymer lengths (nt). **(c)** Memory usage under varying constraints lengths (nt).

Like the situation in preprocessing time, the strand length, the maximum homopolymer length and the total length of the constraints have the probability to influence the memory usage. We show the memory usage under different settings. It should be noted that the memory usage we report is based on actual measurements, which includes overhead from the Python interpreter and the code itself (such as the numpy library). Our empty-file testing shows that this baseline memory consumption amounts to approximately 99.63 MB.

In Supplementary Figure 25a, the total length of the constraints is 1252 and the maximum homopolymer length is 2. As shown in Supplementary Figure 25a, the memory usage scales linearly with the strand length, which aligns with the space complexity in Supplemen-

tary Table 1. Note that the 99.63 MB baseline memory consumption should be excluded. In Supplementary Figure 25b, the total length of the constraints is 1252 and the strand length is 200. As shown in Supplementary Figure 25b, the memory usage is dependent with the maximum homopolymer length, which aligns with the space complexity in Supplementary Table 1. Also as shown in the figure, the memory usage is more stable than the preprocessing time. In Supplementary Figure 25c, the strand length is 200 and the maximum homopolymer length is 2. As shown in Supplementary Figure 25c, the memory usage scales nearly linearly with the total length of the constraints, which aligns with the space complexity in Supplementary Table 1.

We acknowledge that the linear trend in Supplementary Figure 25c is less pronounced. This occurs because in the trie automaton, L serves as an upper bound but not the exact supremum on the number of vertices. Consequently, the observed scaling does not align perfectly with a strict linear upper bound. In contrast, the number of vertices in Supplementary Figure 25a remains constant, which results in a more clearly defined linear relationship.

E.13 Deterministic mapping security and information leakage: a clarification

This section addresses the reviewer's concerns regarding the potential implications of the deterministic mapping employed in Siyuan Code, including payload reuse, equality leakage, and its interaction with the introduced redundancy schemes.

Payload reuse refers to the fact that identical data blocks are mapped to identical DNA sequences. Consequently, one might suspect that after encoding, identical data could become indistinguishable, leading to retrieval failures. In practice, this does not pose a problem because our method binds a unique index (address) to each data block prior to encoding. Therefore, even in the presence of intentional manipulation, each encoded strand remains distinct and no ambiguity arises during decoding. This characteristic does not introduce any adverse effects on the storage system.

Equality leakage is an information-theoretic property: an observer could infer whether two original data blocks are identical by detecting repeated DNA sequences. However, it

is important to emphasize that Siyuan Code is designed as a hardware-oriented storage codec, not an encryption algorithm. Thus, if an attacker gains physical access to the DNA pool, they can directly use our open-source decoder to recover the complete original data, rendering any inference about block equality unnecessary. This means physical access to the DNA pool implies full access to all stored information. Consequently, equality leakage does not introduce a new vulnerability beyond that of complete data exposure.

Interaction with redundancy concerns how the bijective mapping in Siyuan Code interacts with the built-in redundancy layers. To ensure clean separation, we adopt a layered encoding approach: first, inter-strand error-correcting code are added to the data; the resulting blocks are then prepended with address indices; subsequently, per-strand checksums are appended; finally, the entire augmented block containing the original data, inter-strand error-correcting code, address, and intra-strand checksum is encoded via the deterministic bijective mapping. This design ensures that inter-strand correction, intra-strand error detection, and checksum verification are all decoupled from the outer bijective mapping. Meanwhile, because the bijective mapping operates on the whole augmented strand, it is not constrained by the redundancy structures themselves.

Regarding equality leakage, we would like to further clarify that even similar or identical data blocks – if located at different addresses – can be mapped to completely distinct DNA strands after deterministic encoding. The following techniques ensure such differentiation:

1. Inserting a checksum in the high-order bits. Even if two data blocks are identical, different addresses will generally yield different checksums. Modifying a single high-order bit of the DNA strand, due to the complexity of the constraint set, does not simply correspond to a change by a power of two in the binary representation, but rather alters all subsequent lower-order bits. Similarly, altering a high-order digit produces cascading changes in the lower-order bases. Therefore, placing a checksum in the high-order bits effectively prevents similar subsequences from appearing in the encoded output.

2. Applying compression before encoding. Compression reduces highly repetitive patterns in the input data, thereby decreasing the occurrence of highly similar data blocks submitted to the encoder.

3. Building on method 1, using the high-order checksum as a random seed to generate

a pseudo-random sequence, which is then XORed with the entire strand to achieve randomization.

In our current implementation, methods 1 and 2 are employed. These already suffice to ensure that highly similar data blocks are mapped to entirely dissimilar DNA strands, making method 3 unnecessary. Moreover, method 1 was originally introduced to provide a checksum that verifies strand integrity. This allows inter-strand error correction to function as an erasure code—dealing only with erased or known-invalid strands—rather than attempting to correct potentially erroneous data. As a result, this design incidentally addresses potential malicious tampering scenarios that could otherwise exploit deterministic encoding.

E.14 Scoring criteria clarification for density in Table 1

This section provides a detailed explanation of the scoring criteria and comparison methodology used in Table 1 of the main text, which summarizes the physical density and net density of various DNA storage encoding schemes. Its primary objective is to answer a fundamental question for the field: Among existing methods, which one achieves the highest amount of storable information per unit of DNA in a practical, robust setting?

E.14.1 Rationale for the Chosen Metrics: Physical Density and Net Density

We focus on two ultimate, application-oriented metrics:

- Physical Density (PB/g or EB/g): The total amount of recoverable information stored per gram of synthetic DNA. This metric directly correlates with the ultimate storage cost and efficiency of a physical storage system.

- Net Density (bits/nt): The amount of information encoded per nucleotide within the DNA sequence itself, after accounting for all encoding overheads (indices, error-correction codes, etc.) but before applying copy number redundancy. This metric reflects the intrinsic information efficiency of the encoding scheme.

We selected these final output metrics over intermediate parameters (e.g., specific error-correction codes or constraint sets) because they allow for a direct comparison of the overall

system performance relevant to an end-user: how much data can be reliably stored and retrieved.

E.14.2 Methodology: Comparison at Reported Optima

A key design choice for Table 1 is that the data for each method are sourced from its original publication or authoritative demonstration, representing its performance under the parameters and conditions optimized by its authors. This approach is adopted for two principal reasons:

- Field Standard Practice: It aligns with common benchmarking practice within the DNA storage community, enabling a direct comparison of the best-achieved results across different research efforts. Foundational benchmarks in the field, for instance, often compare methods based on their reported peak performances.

- Preserving Method-Specific Optimizations: Different methods employ distinct strategies and parameter synergies to maximize their final density. Forcing all methods into a single, arbitrary parameter set could unfairly degrade the performance of methods whose strengths are tied to specific configurations, thus distorting the comparison. Using author-optimized parameters respects the design intent of each work and provides a fair assessment of its potential.

E.14.3 Clarifying the Core Trade-off: Error Tolerance, Copy Number, and Physical Density

A critical insight for interpreting Table 1 is the inherent, system-level trade-off between an encoding scheme's intrinsic error tolerance, the biological copy number redundancy used, and the final physical density. This trade-off manifests in several distinct optimization strategies in practice:

- Strategy A (High Intrinsic Tolerance, Low Copy Number): Schemes with very strong built-in error resilience (e.g., via powerful ECC) may require only a low copy number for perfect retrieval. This can yield high physical density, as the total mass of DNA synthesized is low.

- Strategy B (Lower Intrinsic Tolerance, High Copy Number): Conversely, methods with modest intrinsic tolerance may require a high copy number to compensate, using molecular redundancy to ensure reliability. Their net density is high though the final physical density may not be competitive caused by copy number.

- Strategy C (Moderate Tolerance, Proactive Error Avoidance, Low Copy Number): Some schemes employ sophisticated code design to proactively circumvent error-prone patterns (e.g., via specific sequence constraints) before information is written. They may not have the strongest corrective power, but by "preventing errors preemptively", they significantly reduce the raw error rate, enabling reliable recovery with a low copy number and potentially achieving high physical density.

Therefore, comparing only the copy number or only the net density in isolation is insufficient. The Physical Density reported in Table 1 is the final product of this complex optimization. It encapsulates the net result of each method's specific choices along this trade-off curve at its optimal operating point.

E.14.4 Argument for Fairness: Why Enforcing Uniform Parameters is Problematic

The aforementioned strategic differences form the core rationale for our methodology. Attempting a "controlled variable" comparison by mandating a uniform parameter (e.g., copy number, specific error-correction strength, or per-strand error rate) is conceptually flawed because:

- Mandating a low copy number would invalidate prior work relying on Strategy B, which is unfair to that research.

- Mandating a specific (controlled) intrinsic tolerance level would invalidate Strategy A, also unfair.

- Mandating an identical per-strand error rate may not fairly evaluate the advantages gained by Strategy C through proactive error avoidance.

Consequently, the fairest and most informative approach is to evaluate the final physical density each method can achieve under its own designed optimal parameters. This is the method employed in Table 1. It allows methods based on fundamentally different strategies to compete on their own strongest grounds, ultimately comparing them on a unified, application-critical end metric: physical density.

E.14.5 Conclusion

The methodology used in Table 1 is designed for fairness and relevance. It compares all methods based on the final, critically applicable metric of physical density, evaluated at their respective performance peaks as established in the literature. This approach answers the most pressing question for advancing the field: Which method, when tuned to its best, delivers the highest information storage capacity per gram of DNA? We believe this provides a transparent, meaningful, and equitable basis for comparison.

E.15 Clarification on the Choice of 170-nt Strands

Regarding optimality, we posit two distinct layers: 1) Coding-Theoretic Optimality: Designing an encoding method that, for a given hardware specification (e.g., strand length, set of error patterns to avoid), maximizes the utilization of the strands to achieve the optimal density under that specification. 2) System-Level Optimality: Identifying the best hardware specification (e.g., optimal strand length) for a given encoding method under current technology to minimize total system cost.

Our work primarily targets the first point. Therefore, our focus is on creating an encoding method whose mathematical construction remains valid and efficient even if underlying biochemical technology advances. The core idea is that by simply updating the input parameters (e.g., target length, constraint set), our method can adapt to new hardware specifications. We aim to demonstrate the mathematical formulation of our encoder—specifically, how it implements an enumerative coding via an efficient bijective mapping based on combinatorial counting—rather than presenting a framework for choosing the optimal hardware parameters (like strand length) for current technology.

Our motivation for focusing on the first layer is that biochemical platforms are subject to rapid iteration; a technology that is standard today may become obsolete. In contrast, a length-agnostic encoding method that achieves the theoretical optimum for anygiven strand length possesses inherent longevity. As proven in Section B, our method achieves the theoretical upper bound of information density for any given set of forbidden patterns (constraints).

As shown in Figure 3, for different strand lengths, longer strands benefit from the amortization of fixed overheads (e.g., indices) but suffer from accumulated error rates. However, our method can suppress the accumulated error rate with minimal redundancy, meaning the dominant factor remains the density gain from overhead amortization. Furthermore, in-silico experiments confirm that our method can surpass the density of baselines even when operating at a length disadvantage, underscoring the fundamental superiority of the coding scheme.

To be forthright, the selection of 170 nt for our in-vitro experiments was primarily dictated by the pricing of a third-party synthesis provider (GenScript). The quotation for significantly longer strands was substantially higher. A principled interpretation of this choice is that 170 nt represents a point of high cost-efficiency under current market conditions. It demonstrates compatibility with budget-conscious, practical DNA storage setups and partially alleviates the negative impact of high synthesis costs on real-world deployment, providing a realistic and economically viable validation point for our coding-theoretic advance.

E.16 Clarification on the Originality Boundaries of the Three First-Layer Components

This appendix aims to precisely elucidate the origins and boundaries between the "first-layer contribution" described in the main text and the related classical ideas. Our core thesis is: The originality of this layer lies not in reinventing textbook-level fundamental concepts such as the Trie tree [40], the Aho-Corasick (AC) algorithm [41], Dynamic Programming (DP) [49], or enumerative coding [50], but in making necessary and provable improvements to these ideas to address the specific, complex constraints of DNA storage, and in their first systematic integration into a unified algorithmic coding framework, thereby bridging

classical theory and practical application.

E.16.1 Overview of Core Theoretical Contributions

The first-layer contribution is manifested as three closely interlinked component innovations:

1. **Novel Trie Automaton**: Inspired by the idea of the Aho-Corasick algorithm but **redesigned** for the encoding task. Its core innovation, the introduction of the *press edge*, strengthens the time complexity of state transitions from **amortized O(1)** to **strict O(1)**, accompanied by new proofs for its construction and properties. The original AC algorithm only guarantees amortized complexity for pattern matching; strictly speaking, it is an algorithm for a specific function, not an automaton with guaranteed complexity control under all circumstances. Thus, "automaton-ization" and the subsequent DP on the automaton are also benefits enabled by the *press edge*.

2. **Dynamic Programming (DP) Counting Component**: For the first time, an efficient DP counting algorithm is implemented **on this specific automaton structure**, solving the "suitable DNA strand" counting problem under complex constraints.

3. **Enumerative Coding Component**: Based on the DP counting results, a complete bijective encoding and decoding process is constructed, applying the classical enumerative idea to a complex state space.

E.16.2 Clarifying the Boundaries with Classical Works for the Three Components

To avoid confusion, the boundaries between this work and related classical ideas are detailed as follows.

1) Bijective Encoding : From Simple Combination to Complex State Space Mapping

Heritage of Ideas: Our method shares conceptual similarities with Thomas Cover's 1973 "Enumerative Source Encoding" [51] in its digit-by-digit determination approach. Cover's method for enumerative encoding under combinatorial constraints also uses a counting problem combined with a trial-and-error process per digit. Decoding similarly involves counting

how many binary strings are lexicographically smaller. However, the simple problem he addressed differs from ours. Cover's constraints were not the complex ones of DNA storage, his encoding does not involve a Trie automaton, nor does it require preprocessing a DP table on an automaton's state graph; instead, it directly calculates combinatorial numbers. Thus, the similarity is at the conceptual level, while the specific algorithms differ.

Core Improvement: Our encoding core inherits the bijective idea from Cover's enumerative coding. However, the classical method of directly calculating combinatorial numbers cannot handle the complex state space in DNA storage. Our key adaptation lies in basing the enumeration process on a novel Trie automaton that precisely models all constraints, and relying on a precomputed DP count table on this automaton for each lexicographic calculation step. Consequently, each step of encoding/decoding transforms from a simple combinatorial number lookup into a deterministic state transition operation on a complex state graph based on precomputed results. This constitutes a fundamental methodological innovation to address the problem's complexity.

2) Dynamic Programming Counting : Efficient Realization on a Specific Automaton

Heritage of Ideas: Our method does not use Thomas Cover's direct combinatorial calculation for the counting problem because the counting problem in our scenario is more complex. Our counting problem is solved via preprocessed dynamic programming[49]. In textbooks, many different mathematical counting problems can be solved using DP. "Using DP to solve counting problems" is not our invention but a common textbook example. Although many textbook DP examples for counting are not on automata, and many DPs are not for coding problems, and our method performs DP on a Trie automaton to solve a coding problem, we still consider textbook DP as the ideological and theoretical foundation for the DP part of our encoding.

Core Improvement: The innovation of this work lies not in proposing the DP idea, but in designing and implementing an efficient DP algorithm suitable for this specific problem. This includes: (a) identifying that the states of the Trie automaton are the natural, optimal DP state definitions; (b) utilizing automaton features like the *press edge* to design a state transition equation with strict O(1) complexity, ensuring efficient DP table construction;

(c) the resulting DP table is a compact data structure that compresses the exponential search space under complex constraints into a polynomially computable form supporting O(1) queries. This queryable DP table itself is the key bridge connecting the underlying automaton to the top-level encoder, representing the engineering and systematization of the classical DP idea in this work.

3) Novel Trie Automaton :

Heritage of Ideas: Our method shares conceptual similarities with the AC algorithm [41] in constructing the Trie automaton. The rules for storing pattern strings in the Trie tree of our automaton are consistent with those in the AC algorithm. Our Trie automaton adopts many definitions from AC, such as the longest suffix present in the Trie. In Appendix Section B, we detail the similarities and differences between our Trie automaton and the AC algorithm, acknowledging that our Trie automaton is an improvement based on AC.

Core Improvement & Necessity: To support the random access pattern required by encoding, our novel Trie automaton introduces innovations like the *press edge*, a new construction algorithm based on it, and a new non-amortized O(1) complexity. The original AC algorithm is typically used for pattern matching. If a slightly modified original AC algorithm were forced to be used in Siyuan Code, minor issues could arise, such as the failure of amortized complexity analysis, which will be analyzed in detail later.

Conclusion: Our more significant contribution lies in adapting these textbook ideas to obtain novel small data structures, novel small algorithms, and novel small codes, and combining them into a novel large codec algorithm. From the perspective of algorithms, information theory, and coding theory, this large codec algorithm can be seen as an engineering realization of these ideas. In the context of interdisciplinary research, theoretical and algorithmic innovation can refer to the formation of a new, large algorithm, while engineering practice can refer to conducting actual biochemical experiments using existing DNA storage algorithms. There exists a significant gap between the most fundamental theoretical computer science perspectives on ideas and theories (e.g., the AC algorithm) and the interdisciplinary perspective on engineering practice (e.g., using an existing algorithm to directly

synthesize DNA dry powder *in vitro* and then sequence it). Siyuan Code, as a whole, bridges this gap.

E.16.3 Core Innovation of the Trie Automaton: Introduction and Necessity of the ***Press Edge***

As mentioned in the main text, a key innovation of our novel Trie automaton is the introduction of the *press edge*. This subsection details its design characteristics and argues for its necessity in the DNA storage coding task, thereby clarifying its fundamental distinction from the classical Aho-Corasick algorithm.

1) Design Characteristics and Advantages of the *Press Edge* Our automaton possesses the following features, which the original AC algorithm does not:

1. **New Data Structure**: The *press edge* is a precomputed, deterministic transition array for each state. It directly maps to the next state given an input character, eliminating the need to traverse *fail* pointers.

2. **New Construction Method**: With the *press edge*, we provide a new construction method in the paper. Using the values of the *press edge* in the recursion, it directly avoids accessing any *fail edge* more than once per node. The original 1975 AC algorithm from textbooks cannot achieve this.

3. **New Pattern Matching Process**: Pattern matching also avoids accessing *fail edges*. This differs from the AC algorithm.

4. **Non-amortized Time Complexity**: Our method provides a non-amortized time complexity bound for both construction and pattern matching. In pattern matching, while the AC algorithm has an amortized complexity of O(1) per transition, it is not strictly O(1) for every node. Our method using *press edge* guarantees strict O(1). This non-amortized property is crucial because the subsequent encoding algorithm requires it.

5. **Simplified Time Complexity Proof**: Utilizing our *press edge* array, the proofs for construction and pattern matching complexity no longer require amortized analysis and are more straightforward. In our automaton construction process, there is only one `for` loop with no nested loops of non-constant size inside, making the time and space complexity evident. Our pattern matching proof is also simpler, with the O(1) per-transition time complexity being more obvious. In contrast, the AC algorithm's time complexity proof relies on amortized analysis and is not immediately apparent.

6. **Acceptable Extra Space Overhead**: In our scenario of DNA storage, the alphabet is small. Therefore, the constant factor increase in space is modest - only about four times that of the *fail edge*, or roughly equivalent to the size of the Trie itself. This is a worthwhile trade-off for the aforementioned strictly non-amortized time complexity, simpler construction method, and simpler complexity proof.

7. **New Capability for Advanced Algorithms**: The automaton equipped with the *press edge* transcends the scope of pattern matching. It can efficiently support complex algorithmic operations like dynamic programming on its state space, a capability not directly provided by the original AC algorithm.

2) Why the Encoding Algorithm Requires "Strict O(1)": The Failure of Amortized Analysis for the Classical AC Algorithm In the classical AC algorithm, the O(1) complexity for state transition is **amortized**. This holds for pattern matching (a linear scanning process) because each character is typically processed once. However, in our encoding algorithm, the access pattern to the automaton is fundamentally different:

1. The encoding process requires frequent, random access to **all successor states** of any given state for counting and decision-making.

2. This access pattern invalidates the premise of "amortization" (certain average properties of the operation sequence). Maliciously constructed inputs or specific access sequences could cause the algorithm to frequently trigger costly *fail* pointer traversals in the worst case, causing the amortized complexity guarantee to fail and leading to performance degradation.

Therefore, an automaton guaranteeing worst-case O(1) state transitions is the theoretical foundation for ensuring the entire encoding algorithm remains efficient even in the worst case. The introduction of the *press edge* is precisely to strengthen the "efficient state transition" property inherent in the AC algorithm idea from an **amortized guarantee based on a specific operation sequence** to an **unconditional, deterministic worst-case guarantee**, thereby fulfilling the fundamental requirements of the encoding task.

Summary: The *press edge* is not a minor engineering optimization but a **critical data structure innovation** to safely and reliably apply an idea originating from pattern matching to the new domain of encoding. It ensures the robustness and predictable performance of the automaton as the core component of the encoding algorithm.

E.16.4 Boundaries with Related Works

1. DNA run-length constraint works, such as "Codes with Run-Length and GC-Content Constraints for DNA-based Data Storage"[19], essentially use a large lookup table. This is fundamentally different from classical information-theoretic enumerative encoding, as it is not based on mathematical counting problems but closer to block-based enumeration. These works also differ from traditional run-length limited coding discussed in coding theory, employing different solutions and merely borrowing the terminology.

2. The core distinction from classical enumerative coding (Thomas Cover) lies in the complexity of the handled constraints and the methodological system, as elaborated above.

E.16.5 Summary: Integration as the Key Innovation

In summary, the originality of the first-layer work is rooted in two levels: **first**, in the targeted improvements to classical ideas (e.g., the *press edge* for trie automaton, DP on trie automaton, bijective coding for complex constraints); but **more importantly**, in the deep integration of these improved components into a "automaton $\leftrightarrow$ DP counting $\leftrightarrow$ enumerative coding" closed-loop algorithmic framework.

The intrinsic coupling of this framework is the key innovation: the automaton provides the precisely modeled state space for DP; DP provides the core, efficiently queryable data structure for encoding; the access pattern of the encoding algorithm, in turn, defines the requirement for the automaton's worst-case performance. The three are interdependent and inseparable.

Therefore, the essence of this work is transforming the potential of classical coding theory into a viable solution for the specific challenge of DNA storage through systematic adaptation and integration, bridging the key gap between general algorithmic ideas and specialized domain application.

Our more significant contribution lies in adapting these textbook ideas to obtain adapted data structures, algorithms, and codes, and combining them into a comprehensive codec algorithm. From the perspectives of algorithm design, information theory, and coding theory, this comprehensive codec can be viewed as an engineering realization of these ideas. In the interdisciplinary context, theoretical and algorithmic innovation can refer to the formation of a new, integrated algorithm, while engineering practice might refer to conducting actual biochemical experiments using existing DNA storage algorithms. There exists a significant gap between the most fundamental theoretical computer science ideas and theories (e.g., the AC algorithm) and the interdisciplinary perspective on engineering practice (e.g., using an existing algorithm for direct *in vitro* DNA synthesis and sequencing). Siyuan Code, as a whole, bridges this gap.

E.17 Amortization failure of classic AC algorithm with amortization

Here we want to explain that the *press edge* is not a minor engineering optimization but a **critical data structure innovation** to safely and reliably apply an idea originating from pattern matching to the new domain of encoding. It ensures the robustness and predictable performance of the automaton as the core component of the encoding algorithm.

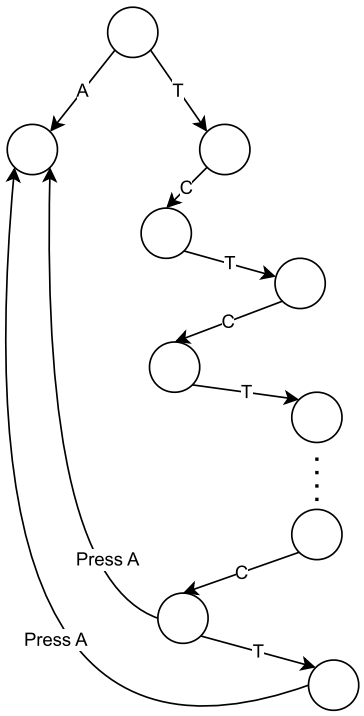
Here is an example of amortization failure of classic AC algorithm in Supplementary Figure 26. A slightly modified original AC algorithm were forced to be used in Siyuan Code.

Let there be 2 unsuitable strands A and $TCTC\cdots TCT$ (length $L-1$). The P_A of all the vertices are A.

DP amortization failure: When calculating DP, for each vertex v in the right part, for each length l, we do the DP calculation.

$$\mathrm{DP}(v,l) = \begin{cases} \sum_c DP\left(P_c(v), l-1\right) & v \notin \mathcal{I} \\ 0 & v \in \mathcal{I} \end{cases} \tag{51}$$

If $P_A(v)$ is not provide in $O(1)$, then at least $\frac{\mathrm{depth}(v)}{2}$ fail edges will be visited. For each l, the total cost of visiting fail edge from all the vertices is at least $1+1+2+2+3+3+\cdots+(\frac{L}{2}-1) = O(L^2)$. This means that the amortized cost is $O(L)$ but not $O(1)$ any longer.



Supplementary Figure 26: Amortization failure example

Encoding/Decoding amortization failure: When decoding $TCTCTCT\cdots TCT$ (sufficiently long), the current vertex v will skip back and forth between the 2 deepest vertices. $P_A(v)$ should be calculated, in order to get the DP value of $P_A(v)$. if P_A is not provided in $O(1)$, it will cost $O(L)$ in each step, which is not acceptable.

The same would happen when encoding the decoding result of $TCTCTCT\cdots TCT$.

Conclusion: It is important to have the non-amortized strict $O(1)$ press edge transition. If the classic AC algorithm with amortization was forced to be employed after slightly modifications, the amortization might fail with the new mission.

E.18 Supplementary Comparison Table for Method Positioning

Supplementary Table 2 provides a multi-dimensional comparison of Siyuan Code with representative methods in DNA storage coding, covering design objectives, density, error models, copy numbers, biochemical constraints, and experimental validation. This table is intended to help readers better understand the positioning of our work within the broader landscape of the field.

Supplementary Table 2: Supplementary Comparison Table for Method Positioning

	Fountain Code [12]	Yin-yang Codec [11]	DNA-Aeon [16]	DNAFormer [7]	DNA StairLoop [28]	Siyuan Code
Design Objective	Practicality+Density	Density	Error correction	Error correction	Error correction	Density
Physical Density (EB/gram)	0.214	2.25	Not Given	Not Given	Not Given	41
Physical Density Growth Factor	[4]× 8.56	[12]× 10.51	Not Given	Not Given	Not Given	[11] × 18.22
Net Information Density (bits/nt)	1.57	Not Given	1.146*	1.6	0.42*	1.66
Perfect Retrieval Copy Numbers	~ 1300	$\sim 10^4$	Not Given	Not Given	Not Given	6
Error Models	Real-world error	Real-world error	Real-world error	Real-world error	Edit distance error	Real-world error
Homopolymer Length Constraints	≥ 4	≥ 4	≥ 3	≥ 4	=3	≥ 1
GC Content Constraints (%)	40 - 60	40 - 60	40 - 60	45 - 55	1/3 - 2/3	42.9 - 57.6*
Other Biochemical Constraints	No	Free energy	Codebook	No	No	Codebook
Experimental Validation	In vitro	In vitro and in vivo	In vitro	In vitro	In vitro	In vitro

Design Objective refers to the primary goal of each method. Some works aim to achieve higher storage density and therefore emphasize the corresponding density values in their manuscripts, while others focus on aspects other than density and do not provide explicit density figures.

Physical Density is expressed in units of EB/gram, representing the in-vitro physical density of dry DNA powder–i.e., the amount of binary data (in exabytes) that can be stored per gram of dry DNA. It should be noted that these methods did not actually fill an entire gram of DNA with exabyte-scale data; rather, the density is calculated by dividing the stored file size by the mass of DNA consumed. Importantly, DNA-Aeon, DNAFormer, and DNA StairLoop are not designed to maximize physical density, and their density figures are neither competitive nor explicitly reported.

The reason our work achieves a significantly higher physical density is that existing methods, due to substantial redundancy, have not yet reached the theoretical high-density potential of DNA storage in terms of orders of magnitude. Our work reduces the need for redundancy by avoiding error-prone strands, where both copy number and error correction contribute to redundancy. The Physical Density Growth Factor indicates the fold improvement in physical density relative to the previous best-performing method. Methods that do not report physical density are excluded from this comparison. Prior to our work, the highest physical density was achieved by the Yin-yang Codec; our method improves upon it by a factor of 18.22.

Net Information Density refers to the effective density after accounting for overheads such as error correction and addressing. It is worth noting that DNA-Aeon and DNA StairLoop do not directly report net information density values in their manuscripts; the figures listed in the table are estimated based on the parameters provided in their respective papers. It can be observed that methods primarily focused on error correction tend not to achieve particularly high net information densities.

Perfect Retrieval Copy Number refers to the number of physical copies required per logical DNA strand (including the strand itself) to achieve 100% perfect data recovery. Our work aims to minimize this number, thereby bringing physical density closer to the theoretical high-density promise of DNA storage. Many prior works have not recognized the importance of this metric and have not reported it.

Error Models describe the types of errors that each method assumes its design must handle. Some methods focus primarily on edit-distance errors and devote substantial discussion to demonstrating their tolerance to such errors. Other methods target real-world errors. It should be acknowledged that even when targeting real-world errors, simulations necessarily involve approximations and cannot perfectly replicate real-world conditions; therefore, simulation alone is insufficient to fully validate real-world feasibility. The key distinction between Real-world error and Edit-distance error models is as follows: the latter assumes that every position on every strand has an equal error probability independent of the strand's sequence pattern, whereas the former recognizes that different strands exhibit different error rates and that error probabilities may vary across positions within a strand.

Homopolymer Length Constraint refers to the maximum allowable homopolymer length. Since some methods cannot reliably enforce a very short maximum length, the reported constraint indicates the range within which the method operates stably. For example, the Yin-yang Codec performs stably when the homopolymer length limit is set to 4 or larger, but becomes increasingly unstable when shorter limits are imposed. As detailed in Supplementary §A.2.1 (Summary of Filtering-based DNA Coding) and Supplementary §E.8.4 (Tests of Instability), the filtering-based methods exhibit instability under stricter homopolymer constraints. In contrast, our method can operate stably under any arbitrary homopolymer length constraint.

GC Content Constraint refers to the GC ratio of the entire strand. To be precise, our reported range of 42.9-57.6% does not mean that all strands fall within this interval. Rather, through local GC control combined with the law of large numbers, approximately 99% of the experimentally measured strands have GC content within this range. A more detailed discussion of GC content can be found in Supplementary §B.1.3 (Rareness of Global GC Content Error).

Other Biochemical Constraints refer to additional error patterns that a method avoids beyond conventional errors. Codebook indicates the ability to custom-avoid a specified list of substrings. However, it should be noted that DNA-Aeon cannot handle certain complex codebooks. According to the DNA-Aeon manuscript [16], a deliberately designed codebook such as {AA, AT, AC, AG} or {ATCA,TATCT,ATCG,TATCC} can easily cause its encoding and decoding to fail. In contrast, our method can handle all valid codebooks.

Experimental Validation shows that, with the exception of the Yin-yang Codec (which includes both in-vitro and in-vivo experiments), most methods are validated solely through in-vitro experiments.

Supplementary References

[1] Joe Davis. Microvenus. *Art Journal*, 55(1):70–74, March 1996.

[2] George M Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628, 2012.

[3] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, jan 2013.

[4] Robert N Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J Stark. Robust chemical preservation of digital information on dna in silica with error-correcting codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, 2015.

[5] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649, 2016.

[6] Kelsey Suyehira, Simon Llewellyn, Reza M. Zadegan, William L. Hughes, and Tim Andersen. A coding scheme for nucleic acid memory (nam). In *2017 IEEE Workshop on Microelectronics and Electron Devices (WMED)*, pages 1–3, 2017.

[7] Daniella Bar-Lev, Itai Orr, Omer Sabary, Tuvi Etzion, and Eitan Yaakobi. Scalable and robust DNA-based storage via coding theory and deep learning. *Nature Machine Intelligence*, 7(4):639–649, April 2025. Publisher: Nature Publishing Group.

[8] Meinolf Blawat, Klaus Gaedke, Ingo Hütter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W. Pruitt, and George M. Church. Forward Error Correction for DNA Data Storage. *Procedia Computer Science*, 80:1011–1022, January 2016.

[9] Yixin Wang, Md. Noor-A-Rahim, Erry Gunawan, Yong Liang Guan, and Chueh Loo Poh. Construction of Bio-Constrained Code for DNA Data Storage. *IEEE Communications Letters*, 23(6):963–966, June 2019. Conference Name: IEEE Communications Letters.

[10] Tuan Thanh Nguyen, Kui Cai, Han Mao Kiah, Duc Tu Dao, and Kees A. Schouhamer Immink. On the Design of Codes for DNA Computing: Secondary Structure Avoidance Codes, February 2023. arXiv:2302.13714 [cs, math].

[11] Zhi Ping, Shihong Chen, Guangyu Zhou, Xiaoluo Huang, Sha Joe Zhu, Haoling Zhang, Henry H Lee, Zhaojun Lan, Jie Cui, Tai Chen, et al. Towards practical and robust dna-based data archiving using the yin–yang codec system. *Nature Computational Science*, 2(4):234–242, 2022.

[12] Yaniv Erlich and Dina Zielinski. Dna fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.

[13] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher N Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze, and Karin Strauss. Random access in large-scale DNA data storage. *Nature Biotechnology*, 36(3):242–248, March 2018.

[14] Lifu Song, Feng Geng, Zi-Yi Gong, Xin Chen, Jijun Tang, Chunye Gong, Libang Zhou, Rui Xia, Ming-Zhe Han, Jing-Yi Xu, Bing-Zhi Li, and Ying-Jin Yuan. Robust data storage in DNA by de Bruijn graph-based de novo strand assembly. *Nature Communications*, 13(1):5361, September 2022.

[15] Ben Cao, Xiaokang Zhang, Shuang Cui, and Qiang Zhang. Adaptive coding for DNA storage with high storage density and low coverage. *npj Systems Biology and Applications*, 8(1):23, July 2022.

[16] Marius Welzel, Peter Michael Schwarz, Hannah F. Löchel, Tolganay Kabdullayeva, Sandra Clemens, Anke Becker, Bernd Freisleben, and Dominik Heider. DNA-Aeon provides flexible arithmetic coding for constraint adherence and error correction in DNA storage. *Nature Communications*, 14(1):628, February 2023.

[17] Luis Ceze, Jeff Nivala, and Karin Strauss. Molecular digital data storage using dna. *Nature Reviews Genetics*, 20(8):456–466, 2019.

[18] David Buterez. Scaling up dna digital data storage by efficiently predicting dna hybridisation using deep learning. *Scientific Reports*, 11(1):20517, 2021.

[19] Wentu Song, Kui Cai, Mu Zhang, and Chau Yuen. Codes With Run-Length and GC-Content Constraints for DNA-Based Data Storage. *IEEE Communications Letters*, 22(10):2004–2007, October 2018. Conference Name: IEEE Communications Letters.

[20] Shubham Chandak, Joachim Neu, Kedar Tatwawadi, Jay Mardia, Billy Lau, Matthew Kubit, Reyna Hulett, Peter Griffin, Mary Wootters, Tsachy Weissman, and Hanlee Ji. Overcoming High Nanopore Basecaller Error Rates for DNA Storage via Basecaller-Decoder Integration and Convolutional Codes. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8822–8826, May 2020. ISSN: 2379-190X.

[21] Hannah F Löchel, Marius Welzel, Georges Hattab, Anne-Christin Hauschild, and Dominik Heider. Fractal construction of constrained code words for DNA storage systems. *Nucleic Acids Research*, 50(5):e30, March 2022.

[22] Ben Cao, Peijun Shi, Yanfen Zheng, and Qiang Zhang. FMG: An observable DNA storage coding method based on frequency matrix game graphs. *Computers in Biology and Medicine*, 151:106269, December 2022.

[23] Ben Cao, Bin Wang, and Qiang Zhang. GCNSA: DNA storage encoding with a graph convolutional network and self-attention. *iScience*, 26(3):106231, March 2023.

[24] Peng Fei and Zhiying Wang. LDPC Codes for Portable DNA Storage. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 76–80, July 2019. ISSN: 2157-8117.

[25] Shufang Zhang, Beibei Huang, Xiangming Song, Tao Zhang, Hanjie Wang, and Yuhong Liu. A high storage density strategy for digital information based on synthetic DNA. *3 Biotech*, 9(9):342, August 2019.

[26] William H. Press, John A. Hawkins, Stephen K. Jones, Jeffrey M. Schaub, and Ilya J. Finkelstein. HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints. *Proceedings of the National Academy of Sciences*, 117(31):18489–18496, August 2020. Publisher: Proceedings of the National Academy of Sciences.

[27] Xuyang Zhao, Junyao Li, Qingyuan Fan, Jing Dai, Yanping Long, Ronghui Liu, Jixian Zhai, Qing Pan, and Yi Li. Composite Hedges Nanopores codec system for rapid and portable DNA data readout with high INDEL-Correction. *Nature Communications*, 15(1):9395, October 2024. Publisher: Nature Publishing Group.

[28] Zihui Yan, Guanjin Qu, Xin Chen, Gang Zheng, and Huaming Wu. DNA StairLoop: enabling high-fidelity data recovery and robust error correction in DNA-based data storage. *Nature Communications*, 16(1):9191, October 2025.

[29] S. M. Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and Error-Free DNA-Based Data Storage. *Scientific Reports*, 7(1):5011, July 2017.

[30] S. M. H. Tabatabaei Yazdi, Han Mao Kiah, Ryan Gabrys, and Olgica Milenkovic. Mutually Uncorrelated Primers for DNA-Based Data Storage. *IEEE Transactions on Information Theory*, 64(9):6283–6296, September 2018.

[31] Pooja Mishra, Chiranjeev Bhaya, Arup Kumar Pal, and Abhay Kumar Singh. Compressed DNA Coding Using Minimum Variance Huffman Tree. *IEEE Communications Letters*, 24(8):1602–1606, August 2020. Conference Name: IEEE Communications Letters.

[32] Melpomeni Dimopoulou, Marc Antonini, Pascal Barbry, and Raja Appuswamy. Image storage onto synthetic DNA. *Signal Processing: Image Communication*, 97:116331, September 2021.

[33] Min Li, Jiashu Wu, Junbiao Dai, Qingshan Jiang, Qiang Qu, Xiaoluo Huang, and Yang Wang. A self-contained and self-explanatory DNA storage system. *Scientific Reports*, 11(1):18063, September 2021. Number: 1 Publisher: Nature Publishing Group.

[34] Yeongjae Choi, Taehoon Ryu, Amos C. Lee, Hansol Choi, Hansaem Lee, Jaejun Park, Suk-Heung Song, Seojoo Kim, Hyeli Kim, Wook Park, and Sunghoon Kwon. High information capacity DNA-based data storage with augmented encoding characters using degenerate bases. *Scientific Reports*, 9(1):6582, April 2019. Number: 1 Publisher: Nature Publishing Group.

[35] Leon Anavy, Inbal Vaknin, Orna Atar, Roee Amit, and Zohar Yakhini. Data storage in DNA with fewer synthesis cycles using composite DNA letters. *Nature Biotechnology*, 37(10):1229–1236, October 2019. Number: 10 Publisher: Nature Publishing Group.

[36] Meng Yu, Xiaohui Tang, Zhenhua Li, Weidong Wang, Shaopeng Wang, Min Li, Qiuliyang Yu, Sijia Xie, Xiaolei Zuo, and Chang Chen. High-throughput DNA synthesis for data storage. *Chemical Society Reviews*, 53(9):4463–4489, 2024. Publisher: Royal Society of Chemistry.

[37] Kunjie Li, Xiaoyun Lu, Jiaqi Liao, Heng Chen, Wei Lin, Yuhan Zhao, Dongbao Tang, Congyu Li, Zhenyang Tian, Zhi Zhu, Huifeng Jiang, Jun Sun, Huimin Zhang, and Chaoyong Yang. DNA-DISK: Automated end-to-end data storage via enzymatic single-nucleotide DNA synthesis and sequencing on digital microfluidics. *Proceedings of the National Academy of Sciences*, 121(34):e2410164121, August 2024. Publisher: Proceedings of the National Academy of Sciences.

[38] Michael Schwarz, Marius Welzel, Tolganay Kabdullayeva, Anke Becker, Bernd Freisleben, and Dominik Heider. Mesa: automated assessment of synthetic dna fragments and simulation of dna synthesis, storage, sequencing and pcr errors. *Bioinformatics*, 36(11):3322–3326, 2020.

[39] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how*, volume 488. Springer, 2005.

[40] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[41] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[42] Emily M LeProust, Bill J Peck, Konstantin Spirin, Heather Brummel McCuen, Bridget Moore, Eugeni Namsaraev, and Marvin H Caruthers. Synthesis of high-quality libraries of long (150mer) oligonucleotides by a novel depurination controlled process. *Nucleic acids research*, 38(8):2522–2540, 2010.

[43] Jiasen Li, Mingkai Dong, Zhengwei Qi, and Haibing Guan. Siyuan code — towards density-optimal dna storage with practical bijective coding. Code Ocean, 2026. DOI: 10.24433/CO.4952245.v1.

[44] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. Pear: a fast and accurate illumina paired-end read merger. *Bioinformatics*, 30(5):614–620, 2014.

[45] Jiasen Li, Mingkai Dong, Zhengwei Qi, and Haibing Guan. Siyuancodesequencing — sequencing data for towards density-optimal dna storage with practical bijective coding, 2026. DOI: 10.5281/zenodo.21357059.

[46] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[47] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2007.

[48] Michael Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–271. IEEE Computer Society, 2002.

[49] Charles Eric Leiserson, Ronald L Rivest, Thomas H Cormen, and Clifford Stein. *Introduction to algorithms*, volume 3. MIT press Cambridge, MA, USA, 1994.

[50] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 2nd edition, 2006.

[51] T. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, January 1973.