Seminar on DNA Data Storage

Improved read/write cost tradeoff in DNA-based data storage using LDPC codes

Shubham Chandak, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, Tsachy Weissman, Hanlee J (2019)

Presented by: Atar Ron and Mattan Hoory

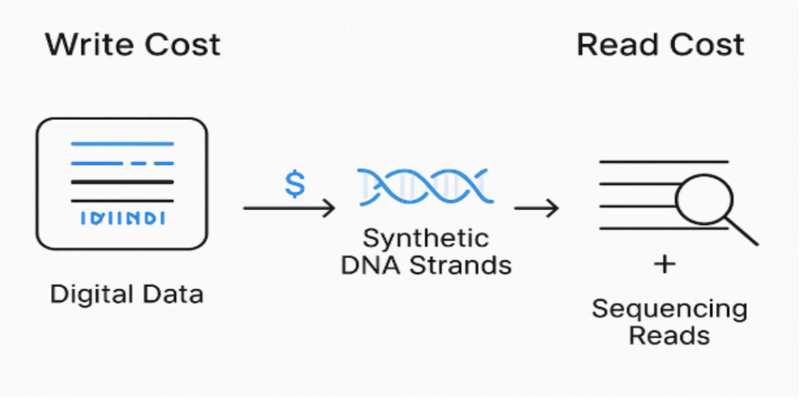
Table of contents

1 Introduction & Context

Background

In DNA-based data storage, there are two critical challenges:

- **Write cost** — how much synthetic DNA we need to store one bit of information.
- **Read cost** — how many reads are needed to reliably recover that bit.



These two costs are closely related, and thus, one of the most important problems in DNA data storage is finding an effective tradeoff between them.

Discussion Questions

- Why, in your opinion, are the cost of write and cost of read closely related? how does the write cost affect the read cost?

- What are the main factors that affect the cost of write?

- What are the main factors that affect the cost of read?

Main Goals

Since the cost of write and cost of read are strongly correlated, this paper aims to:

- Establish a theoretical lower bound on the tradeoff between write cost and read cost.
- Design and evaluate a practical coding scheme (based on LDPC codes) that achieves a better tradeoff than previous methods.
- Validate the performance of the scheme through both real experiments (DNA synthesis and sequencing) and simulations.

Cost of Read and Write

- **Cost Of Write** Average number of encoded bits synthesized per information bit:

$$c_w = \frac{\#\text{Synthesized bits}}{\#\text{Data bits}}$$

- **Cost of Read** Average number of bits read per information bit:

$$c_r = \frac{\#\text{Read bits}}{\#\text{Data bits}}$$

Coverage

- **Coverage**- Average number of bits read per synthesized bit:

$$\text{Coverage} = \frac{\#\text{Read Bits}}{\#\text{Synthesized bits}} = \frac{\frac{\#\text{Read bits}}{\#\text{Data bits}}}{\frac{\#\text{Synthesized bits}}{\#\text{Data bits}}} = \frac{c_r}{c_w}$$

- Coverage has been widely used in prior work to estimate the efficiency of read/write tradeoffs in DNA-based storage systems.
- It measures how many sequencing reads are made per synthesized bit — but is that the best way to evaluate system performance?

Is Coverage a Good Metric?

We consider the following example that demonstrates why coverage can be misleading.

Example

Suppose we compare two storage systems with the following properties:

System	c_w	c_r	$\mathrm{coverage} = c_r / c_w$
A	4	12	3
B	2	10	5

- **Note:** In this example, we assume that both the read cost and the coverage values were measured after decoding all the sampled strands.

Example - cont.

System	c_w	c_r	$\text{coverage} = c_r / c_w$
A	4	12	3
B	2	10	5

- At first glance, System A has better (lower) coverage. Therefore, when evaluating the systems based on coverage only, we will prefer System A.
- But, it is easy to see that System B reads fewer total bits per information bit (10 vs. 12), and also synthisizes significantly less DNA. So despite having higher coverage, System B is clearly more efficient overall.

Example - cont.

Conclusion: This example illustrates that relying solely on coverage can be misleading, particularly when comparing systems with varying write costs. A more meaningful comparison is to evaluate the actual read and write costs per information bit.

Important Note: We can also define coverage at the strand level: the average number of times a strand is observed in the sampled reads. We'll refer back to this concept later on.

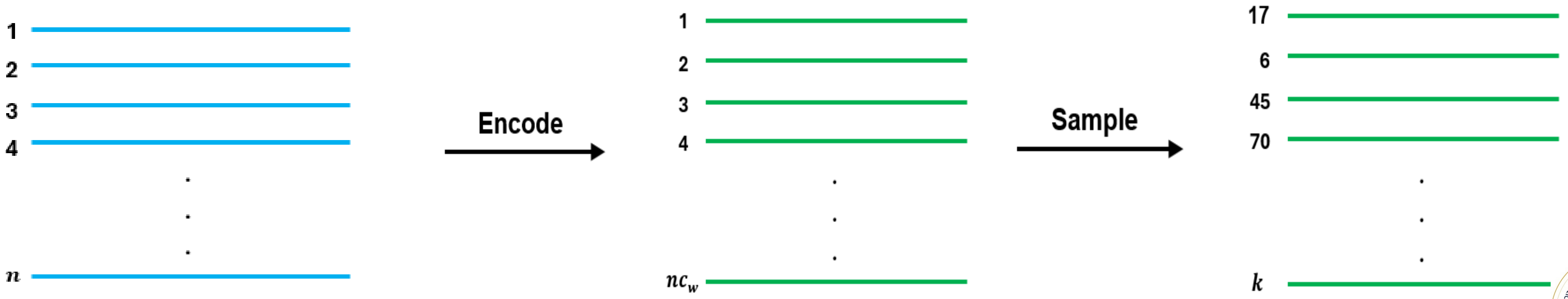
The Model

Model Notations

- n – Number of strands.
- L – Strand length.
- c_w – Cost of write.
- c_r – Cost of read.
- ϵ - Substitution error rate.

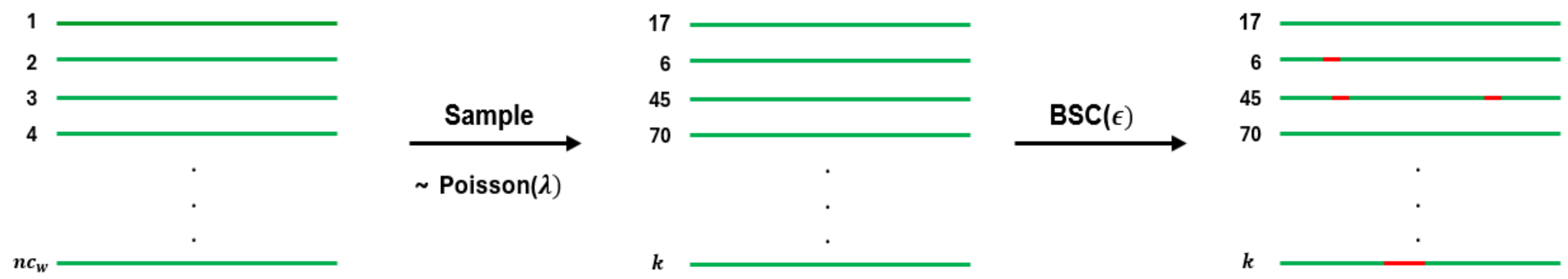
Model Definition

- The storage system encodes n information strands, each of length L, resulting in a total of nL data bits.
- Each data bit leads to the synthesis of c_w bits. Thus, encoding nL data bits requires synthesizing nc_wL bits, which are grouped into nc_w strands.
- Reading each data bit involves sampling c_r synthesized bits. Therefore, nL data bits require sampling of nc_rL synthesized bits, which can be organized into nc_r strands.



Model Definition - cont.

- It is assumed, for simplicity, that the decoder has access to the index of each strand, and that deletion and insertion errors are ignored. We later explain how the authors overcame this in practice.
- The reads are subject to:
  - **Substitution errors** — each sampled bit is flipped with probability ϵ (modeled as a BSC, discussed later).
  - **Sampling variability** — the number of times each strand is sampled is random and follows a Poisson distribution with $\lambda = \frac{c_r}{c_w}$.



2 Communication and Information 101

Communication and Information - Motivation

- We've seen that the **write cost** c_w and **read cost** c_r are tightly connected. With an optimal code, increasing c_w typically reduces c_r.
- However, a higher write cost means more redundancy, which **lowers the information rate** per bit.
- In this section, we introduce basic **channel theory** to better understand and analyze the **tradeoff between** c_w **and** c_r.

What is a Channel?

A **channel** is a mathematical model used in information theory to describe how information is transmitted from a sender to a receiver.

- Input X: a message of n bits.
- Output Y: a possibly altered version of the message

The channel introduce **noise**, leading to errors or loss.

Channel Examples

Capacity

- Capacity is the highest rate at which information can be reliably transmitted over a communication channel.
- Given n data bits transmitted through a channel, the capacity quantifies the maximum number of bits that can be reliably recovered from the output.
- For a Binary Erasure Channel (BEC) with erasure probability ε, the capacity is:

$$C = 1 - \varepsilon$$

Code Rate

- In coding theory, the **code rate** R quantifies the rate of **information** transmitted per **encoded bit**.
- Recall that the write cost c_w is defined as:

$$c_w = \frac{\#\text{Synthesized bits}}{\#\text{Data bits}}$$

which implies:

$$\frac{1}{c_w} = \frac{\#\text{Data bits}}{\#\text{Synthesized bits}} = R$$

- In other words, the **inverse write cost** captures how much information is packed into each encoded bit, which is the code rate.

Shannon’s Theorem

- Shannon’s Theorem: Reliable communication over a noisy channel is possible **if and only if** the code rate R satisfies $R \leq C$, where C is the channel’s capacity.

- Intuitively, if $R > C$, then the rate of inforamtion in each bit is higher than the rate that can be reliably recovered. Therefore, some of it must be lost or corrupted.

- However, if $R \leq C$, Shannon proved that with a suitable coding scheme and long enough messages, the error probability can be made arbitrarily small.

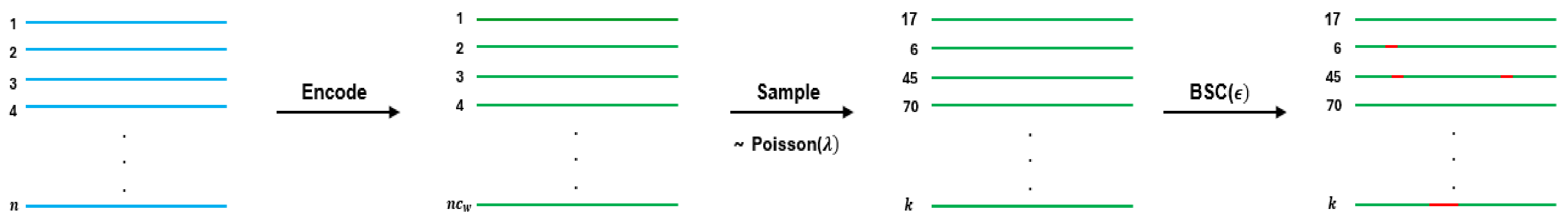
Numerical Example

Assume we have a BEC with $\varepsilon = 0.6$ and an encoder with $c_w = 2$.

- $R = \frac{1}{c_w} = \frac{1}{2}$.
- $C = 1 - \varepsilon = 1 - 0.6 = 0.4$
- It holds that $R = 0.5 > 0.4$
- Therefore, in this model, the data might not be decoded correctly, or might miss some information.

Recap

- The storage model consists of nc_w synthesized strands and nc_r sampled strands. Each strand is sampled a number of times drawn from a $\mathrm{Poisson}(\lambda)$ distribution. The index of each strand is assumed to remain intact.
- The data is transmited through a Binary Symmetric Channel (BSC) with substitution rate ε.
- In a Binary Erasure Channel (BEC), the capacity of the channel must be greater or equal to the rate of the code.



3 Theoretical bounds

The Case $\varepsilon = 0$

- When $\varepsilon = 0$, there are **no substitution errors**.

- Each DNA strand is sampled independently, and the number of times a strand appears follows a **Poisson distribution** with mean $\lambda = \frac{c_r}{c_w}$.

- The probability that a given strand is sampled 0 times is $\mathbb{P}[X = 0] = e^{-\lambda}$.

- If a strand is not observed in the read process, we effectively have an **erasure**.

- Thus, the model can be modeled as a **Binary Erasure Channel (BEC)** with erasure probability: $\varepsilon' = e^{-c_r/c_w}$.

The Case $\varepsilon = 0$ - cont.

- By Shannon’s Theorem, the capacity of the BEC must be greater or equal to the rate of the code,Therefore:

$$R = \frac{1}{c_w} \leq 1 - e^{-\frac{cr}{cw}} = C$$

Simplifying the equation, we get the following lower bound for the cost of read:

$$c_r \geq c_w ln(\frac{c_w}{c_w - 1})$$

- We can see that as c_w increases, c_r decresed, and vice-versa. This fits our intuition.

The Case $\varepsilon \neq 0$

- Now, each sampled bit is transmitted through a BSC with error probability $\varepsilon > 0$.

- Recall that it is assumed that the index of each strand remains intact, and that each strand is sampled k times, where k is $Poisson(\frac{c_r}{c_w})$ distributed.

- Assume that for each bit in each strand, we are given a tuple (k_0, k_1) that indicates how how many times the bit was sampled as 0 and 1, consecutively. Note that $k_0 + k_1 = k$.

- The probability that the samples of the bit are (k_0, k_1) given that the bit has value 0 is:

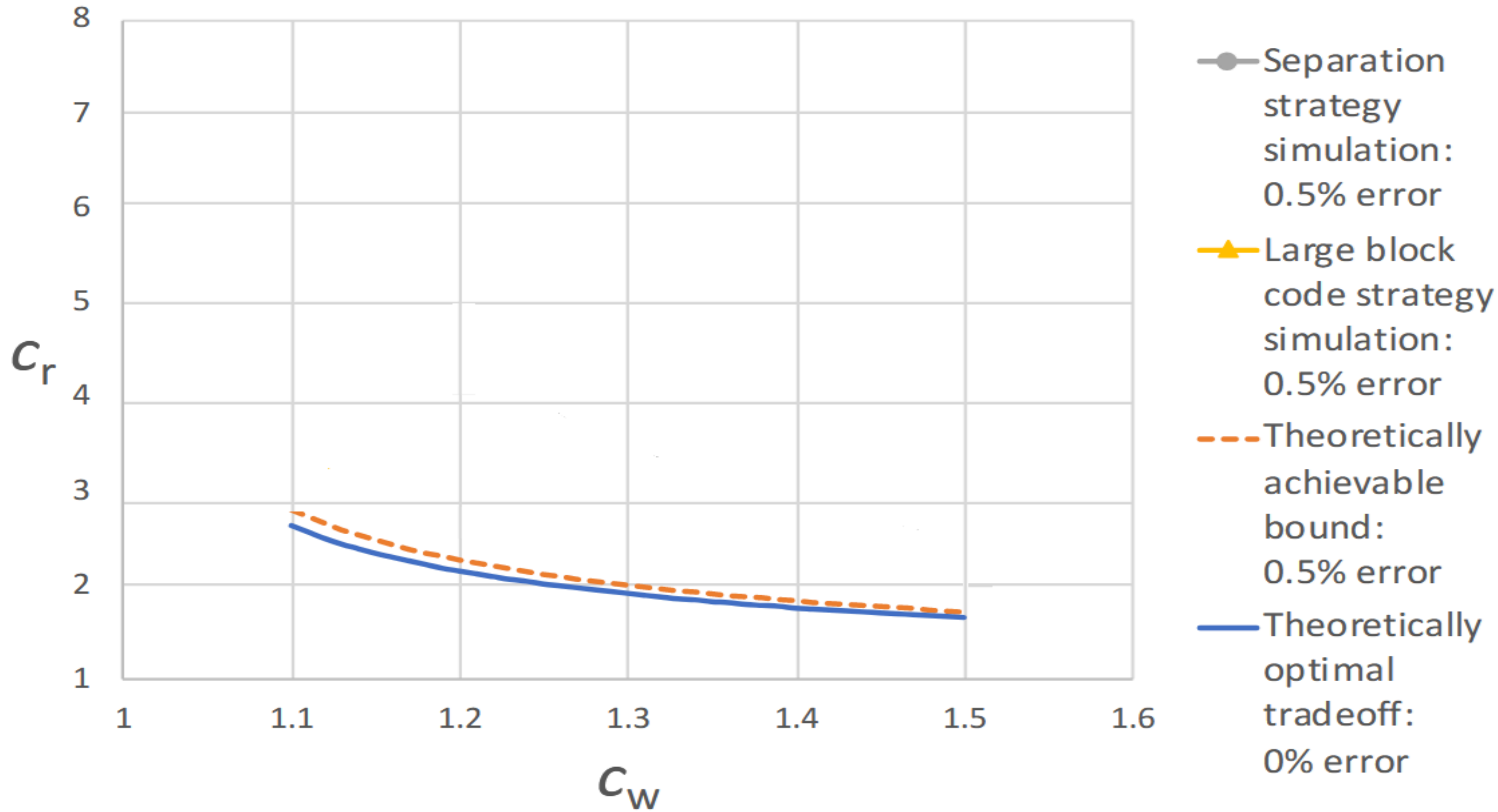
$$P((k_0, k_1)|0) = \mathbb{P}[X = k]\binom{k_0 + k_1}{k_1}(1 - \varepsilon)^{k_0}\varepsilon^{k_1}$$

The Case $\varepsilon \neq 0$ - Example

- In this example, we assume that $\epsilon = 0.1$ and $\lambda = 3$.

Source	Bit 1	Bit 2	Bit 3	Bit 4
Original	1	0	1	0
Sample 1	1	0	1	0
Sample 2	1	1	1	0
Sample 3	0	0	1	0
(k_0, k_1)	(1,2)	(2,1)	(0,3)	(3,0)
Probability	0.054	0.054	0.163	0.163

c_r vs c_w: Theoretical bounds



4 Comparison of coding strategies

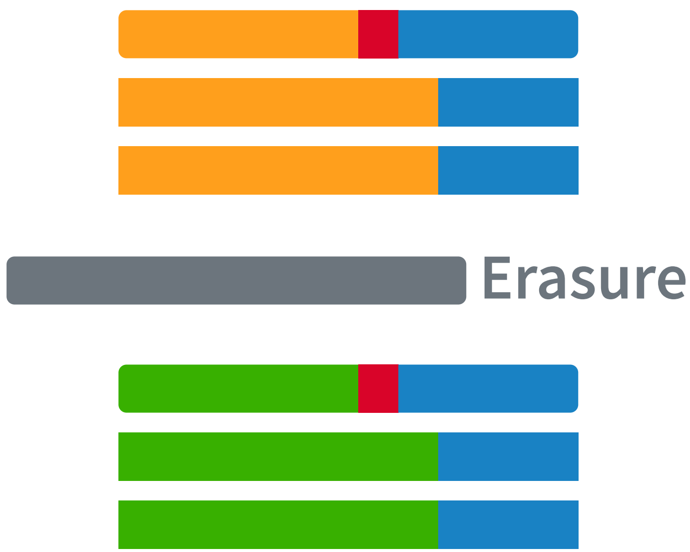
Coding Strategies - Introduction

When designing an encoding and decoding scheme, we must account for **two primary sources of error**:

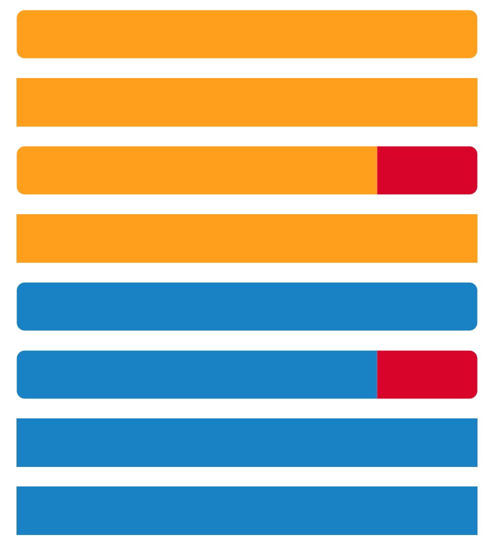
- **Outer Code Errors** — Since the data is divided into multiple strands, some strands may never be sampled or sequenced. The outer code must handle such **erasure errors** and enable recovery of the missing data.
- **Inner Code Errors** — During synthesis and sequencing, strands may undergo **substitution**, **insertion**, or **deletion** errors. The inner code must correct these errors to recover the original encoded information.

Two types of coding strategies

Inner / Outer Code Separation



Single Large-Block LDPC

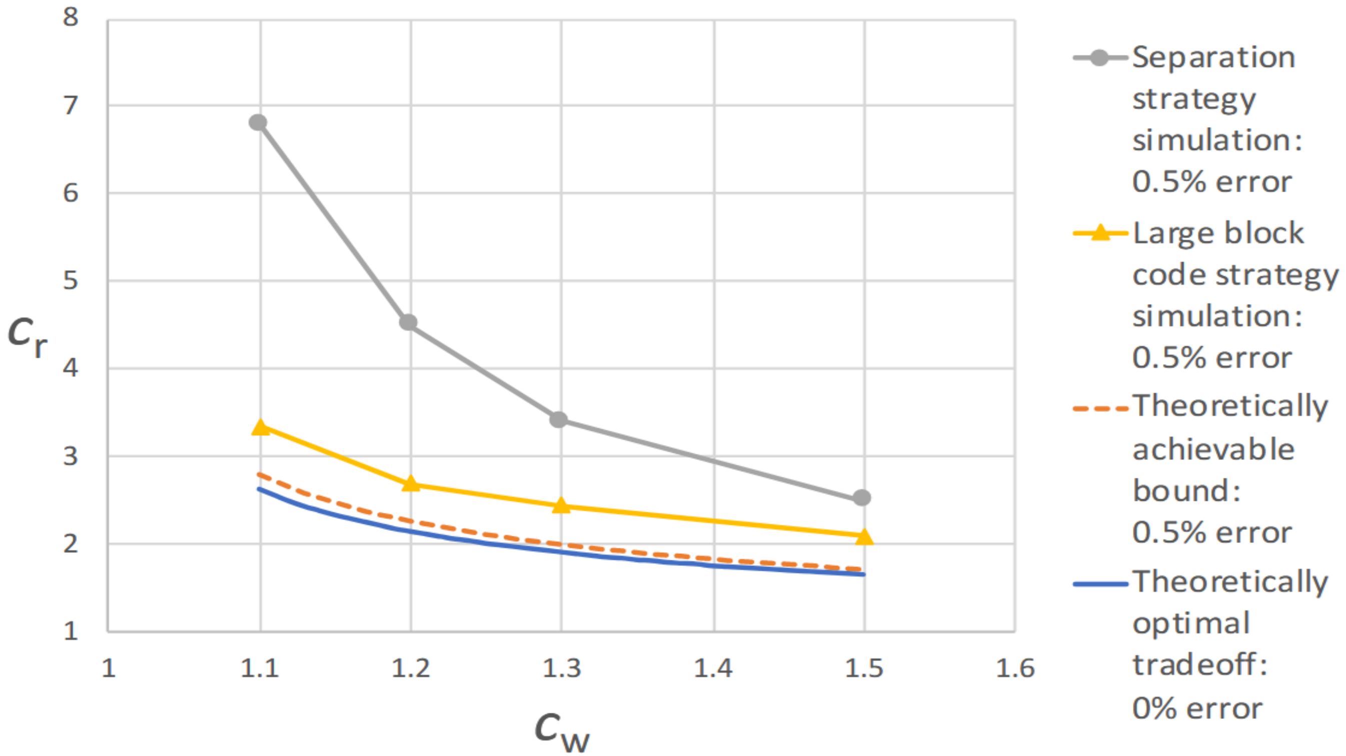


Coding Strategies - Summary

- The following table summarizes the role of each type of code in handling **erasure** and **substitution** errors:

Code Type	Erasure Errors	Substitution Errors
Inner Code	✗	✓
Outer Code	✓	✗
Large Block Code	✓	✓

c_r vs c_w: Simulation Bounds (Separated vs. Large Block)



- The parameters used in the simulation are: $n = 1000, L = 256$.

5 Low Density Parity Check (LDPC)

LDPC Codes

- LDPC (Low-Density Parity-Check) codes are a powerful class of linear error-correcting codes.
- First introduced by Gallager in the 1960s, but gained widespread adoption in the 1990s with improved decoding algorithms.
- They are now used in modern systems such as Wi-Fi, 5G, satellite communication, flash storage, and digital broadcasting.
- LDPC codes achieve performance close to the Shannon limit, making them highly efficient for reliable communication.
- They support fast and scalable decoding using iterative message-passing algorithms.

LDPC Motivation

Assume we have the following data to be transmitted:

$$100110$$

and after transmitting the message, it was received as:

$$1?0110$$

What can help us recover the erasured bit?

Pairity Check

We can enhance our data by adding a parity check bit:

1 0 0 1 1 0 1

Suppose we later receive the following (with one bit erased):

1 ? 0 1 1 0 1

To recover the missing bit, we compute the parity of the known bits and compare it to the parity check bit. This allows us to deduce the value of the erased bit.

What happens if more than 1 erasure?

Assume with have the same data as before with the pairity check bit:

1 0 0 1 1 0 1

And now, after transmitting this data, the recieved data is:

1 ? 0 1 ? 0 1

Can our pairity check bit recover both erasured bits?

Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

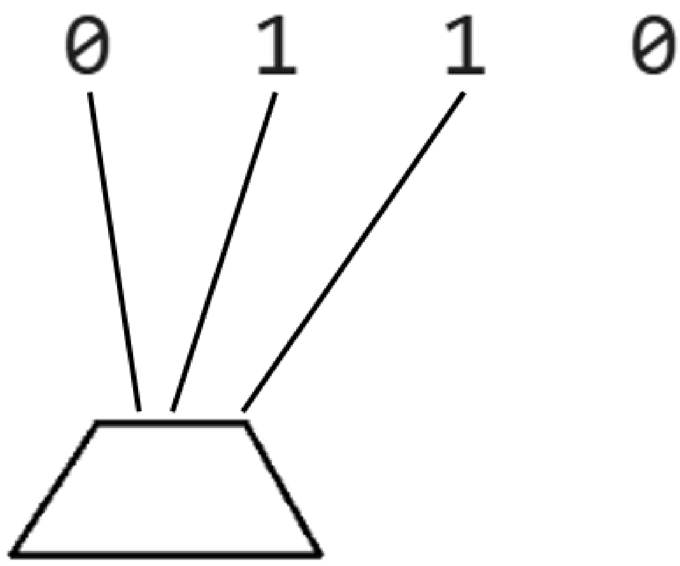
```
0   1   1   0
```

Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

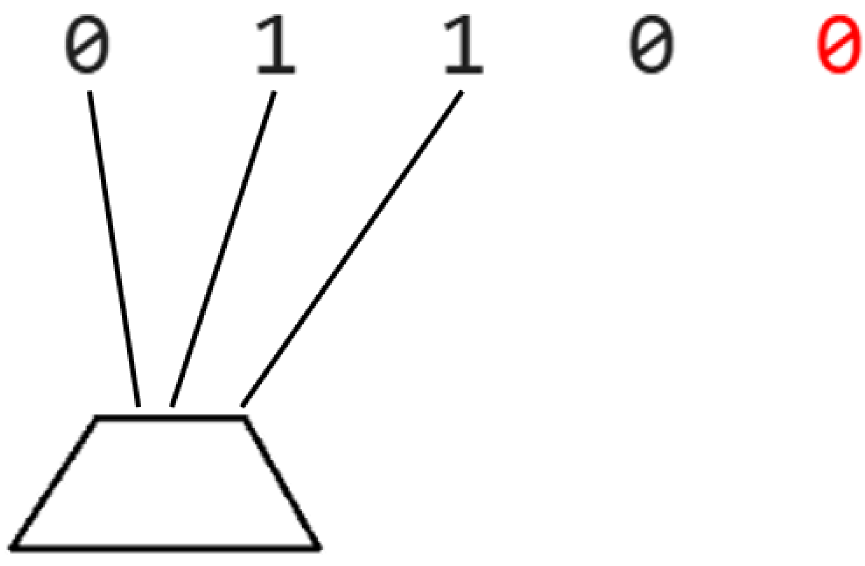


Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

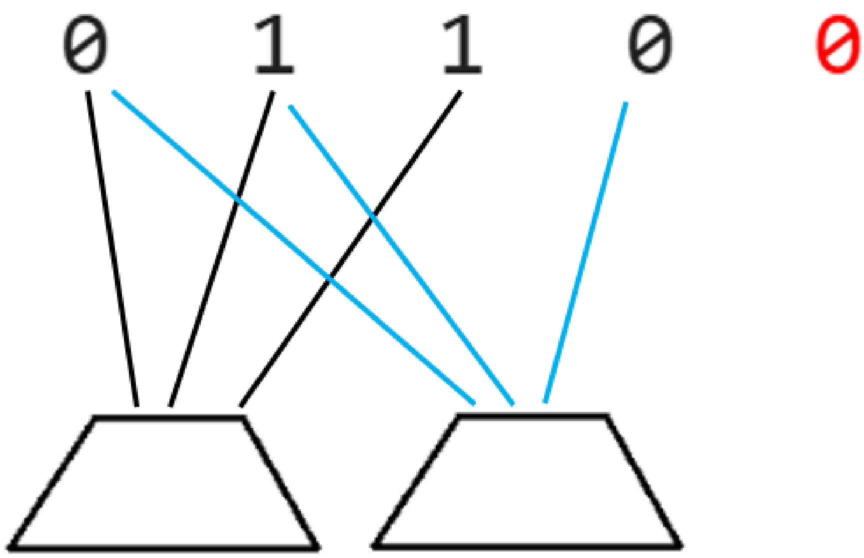


Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

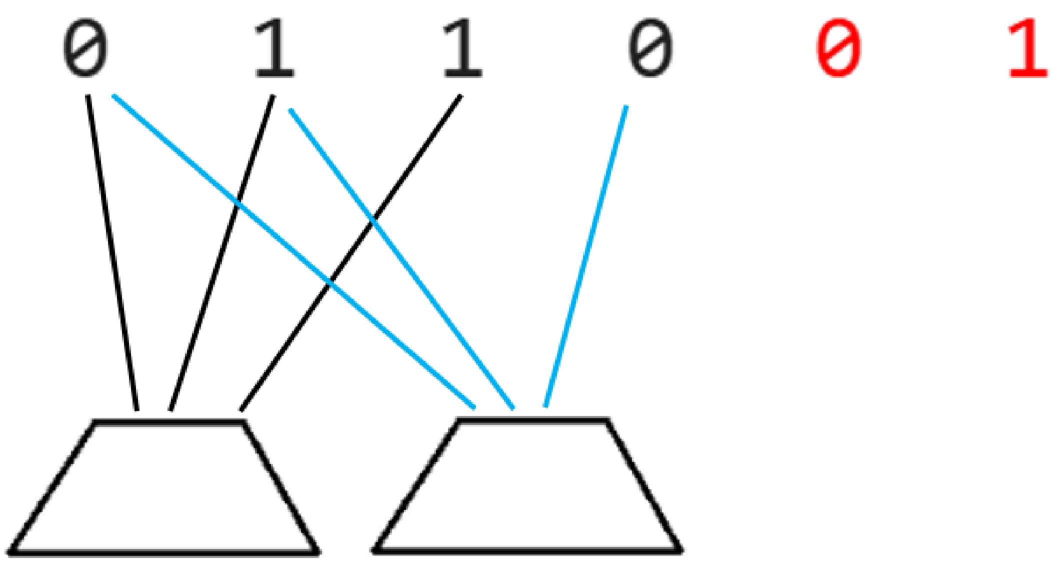


Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

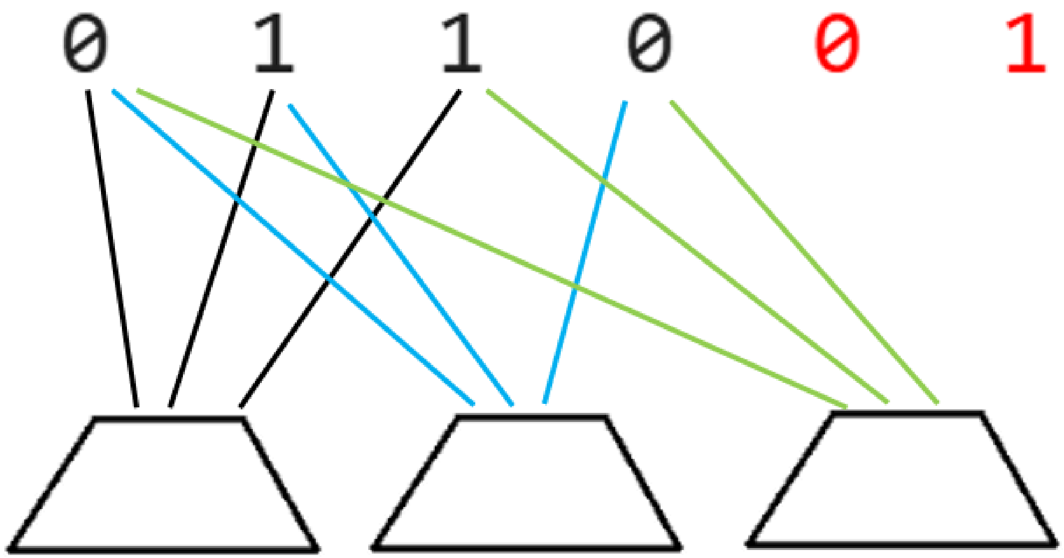


Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

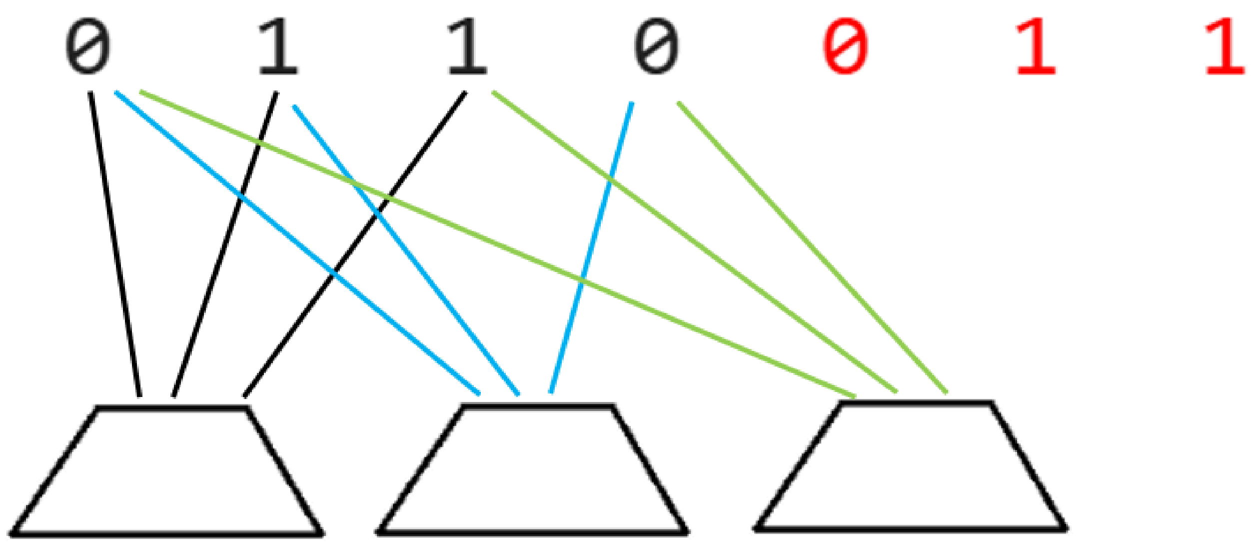


Possible Solution

Overlapping Pairity Checks

We can group bits from the strand into subsets and apply parity checks to each group.

- Consider the following 4-bit example:

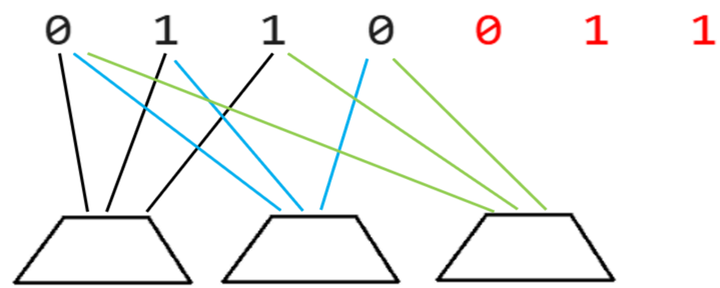


- With this 4-bit block pairity checks, we can correct any 2 erasures in the block.

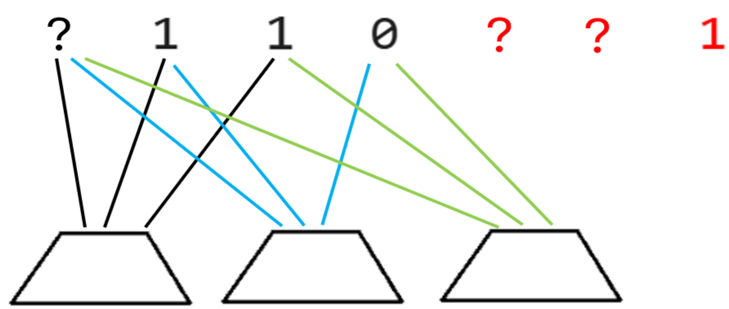
Possible Solution

Parity Check Bits Protection

- Consider the previous example:



- Assume that the erasures are as follows:

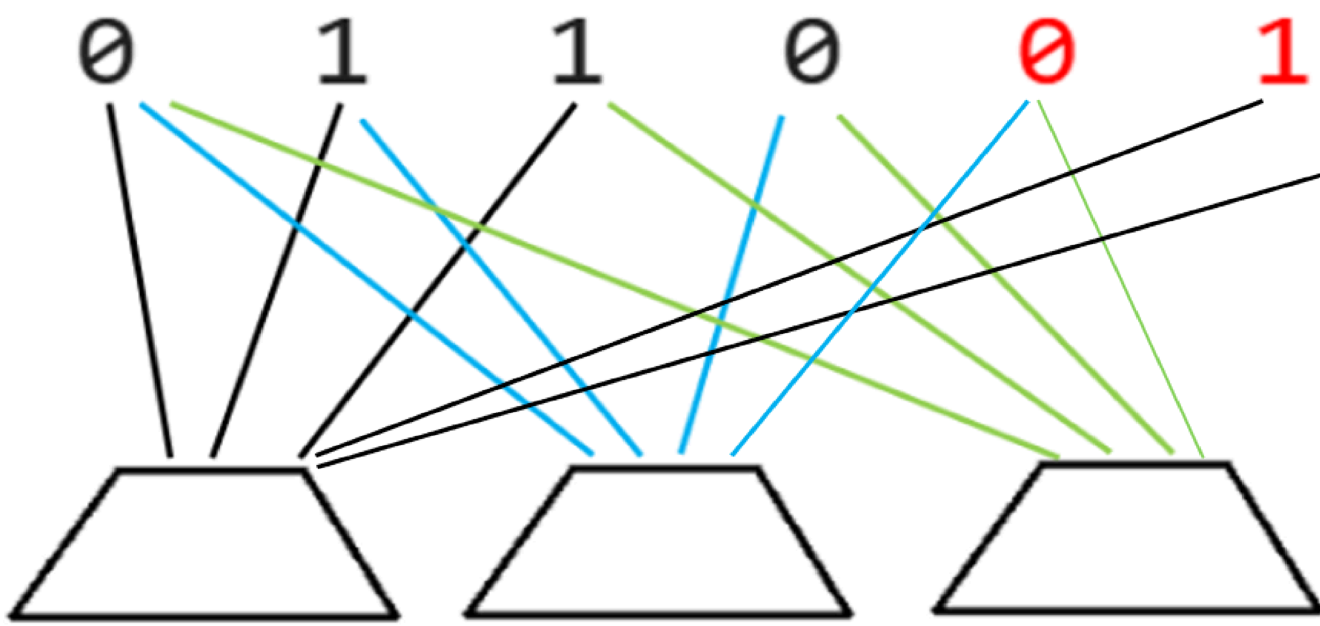


- Can the erasured data bit be recovered?

Possible Solution

Parity Check Bits Protection

We can embed **parity-check protection bits** within the existing parity-check structure:

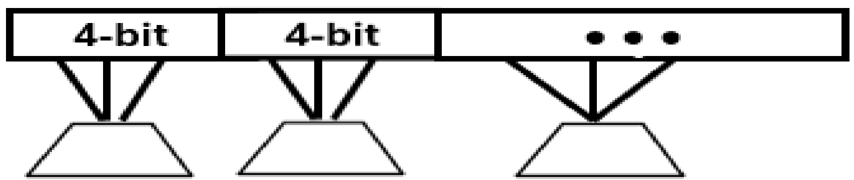


This allows us to protect and recover even the **parity-check bits themselves**.

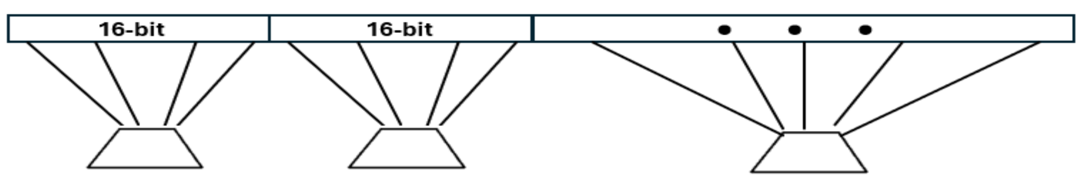
Possible Solution

Pairity Chack Implementation

- What would happen if we apply the 4-bit strategy repeatedly over consecutive 4-bit blocks in a longer message?

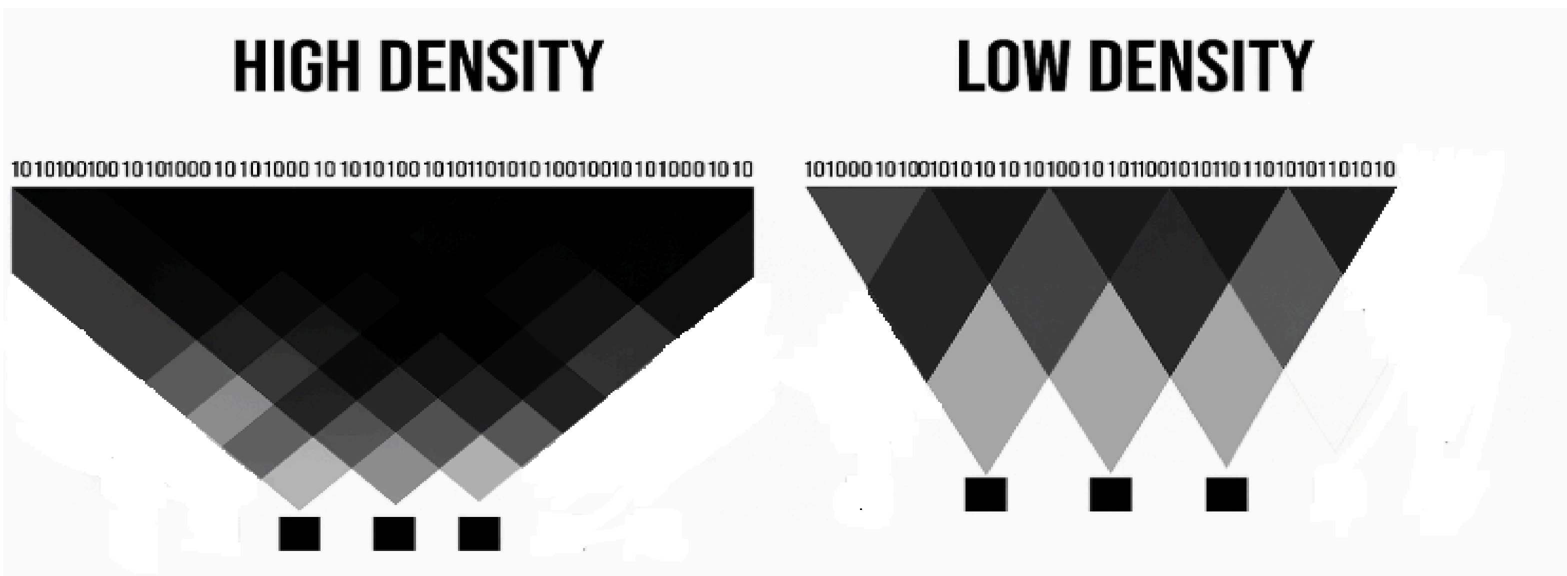


- And what if we increase the overlap size together with the amount of pairity checks per overlap — how does that impact performance and error correction?



Density and the Role of Overlap

- Therefore, our goal is to design a parity-based error correction code that:
  - Has **low write cost**,
  - Enables **fast and efficient decoding**,
  - **recovers erasures effectively**.
- In the 1960s, Robert Gallager introduced the term of **density**, as shown in the image below.

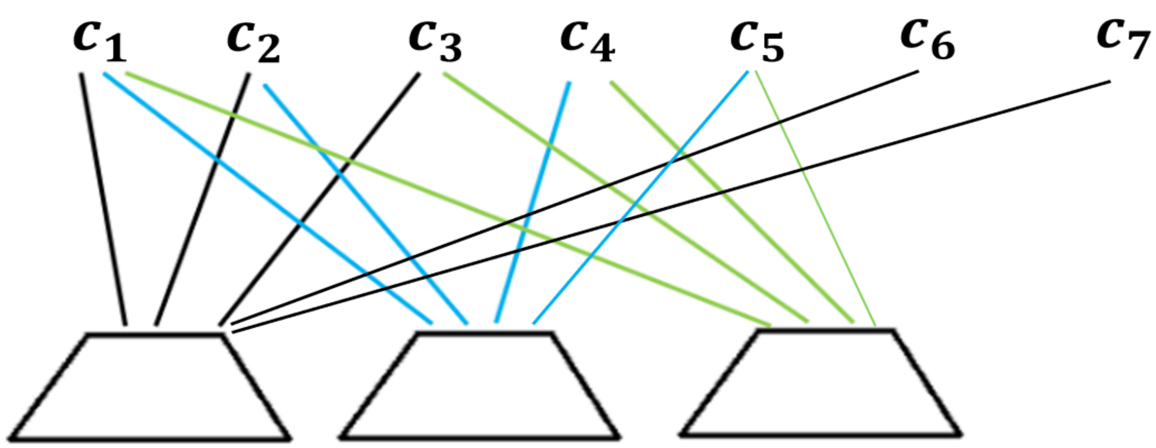


LDPC Theory

Tanner Graphs

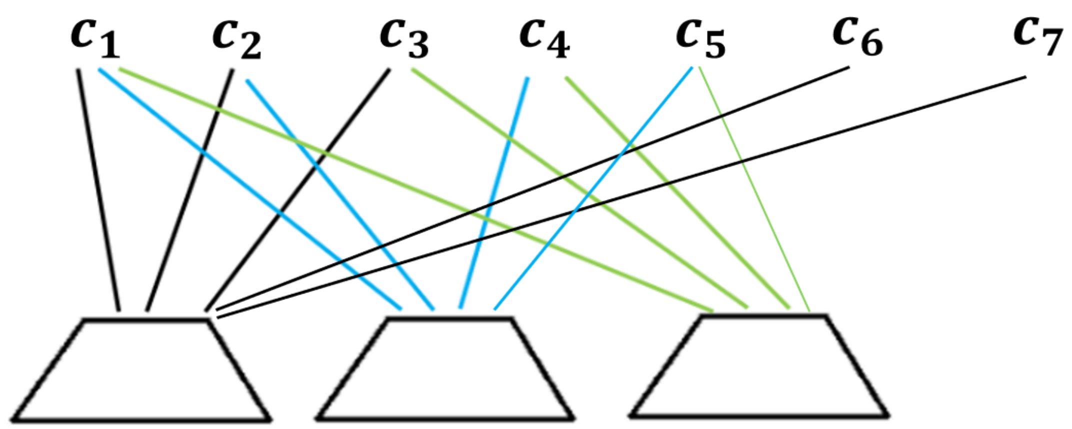
- A **Tanner graph** is a **bipartite graph** with:
  - **Variable nodes**: one for each bit.
  - **Check nodes**: one for each parity-check.
- An edge connects variable node v_j and check node c_i if and only if v_j participates in c_i.

Let's take a look at a familiar Tanner graph:



LDPC Theory

Parity Check Equations



This Tanner graph defines 3 parity check equations:

$$c_1 \oplus c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0$$
$$c_1 \oplus c_2 \oplus c_3 \oplus c_5 = 0$$
$$c_1 \oplus c_3 \oplus c_4 \oplus c_5 = 0$$

LDPC Theory

Matrix Formulation

$$
\begin{aligned}
c_1 \oplus c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0 \\
c_1 \oplus c_2 \oplus c_3 \oplus c_5 = 0 \\
c_1 \oplus c_3 \oplus c_4 \oplus c_5 = 0
\end{aligned}
$$

Codeword constraints are often written in matrix form. The above constraints become:

$$
\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}}_{H} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
$$

LDPC Theory

Pairity Check Matrix

- As seen in the previous slide, LDPC code can be presented as a parity-check matrix H.
- The matrix is composed of n columns and k rows.
- Each column represents a bit (pairity bit or regular bit).
- Each row represents a parity-check equation over some bits.
- Each row has only a few non-zero entries, which mean that the matrix is sparse and the pairity checks have low density.

LDPC Decoding: Belief propagation algorithm

- Gallager introduced the idea of iterative, messagepassing decoding of LDPC codes.
- The idea is to iteratively share the results of local node decoding by passing them along edges of the Tanner graph.

Message decoding

Y

↓

Decode

↓

X

Probabilistic Decoding (Posterior from Prior)

Prior ($P(x = 0) = 0.7, P(x = 1) = 0.3$)

↓

Decode

↓

Posterior ($P(x = 0 \mid y) = 0.9, P(x = 1 \mid y) = 0.1$)

Optimality of the Sum–Product Algorithm

$$\hat{x}_i = \arg\max_{x_i \in \{0,1\}} P(X_i = x_i \mid \mathbf{y})$$

- The code bits x_i are independent
- The Tanner graph has no cycles
- **Not** met in practice

A-priori and LLRs probability

- **Def**: For a binary variable x_i The **log of posterior ratio** is given by:

$$L_{\text{posterior}}(x_i) = \log \frac{p(x_i = 0 \mid y)}{p(x_i = 1 \mid y)} \quad \begin{cases} > 0 & \text{bit more likely 0} \\ < 0 & \text{bit more likely 1} \\ = 0 & \text{no preference (erasure)} \end{cases}$$

- Bayes':

$$L_{\text{posterior}} = L_{\text{prior}} + \tilde{L}$$

- **log prior ratio**:

$$L_{\text{prior}}(x_i) = \log \frac{p(x_i = 0)}{p(x_i = 1)}$$

- **log-likelihood ratio** (LLR):

$$\tilde{L}(x_i) = \log \frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)}$$

- Iterative message-passing updates $L_{\text{posterior}}$ using parity-check constraints.

Variable and *Check* Node Connection Sets

For an LDPC code with parity-check matrix $H \in \mathbb{R}^{m \times n}$

Definition – Variable → Check Messages

- We denote by $L_{i\to j}^{[v]}$ the LLR information **computed by variable node** v_i and sent (along its edge in the Tanner graph) **to check node** c_j.
- We denote by $L_{i\leftarrow j}^{[c]}$ the LLR information **computed by check node** c_j and sent (along its edge in the Tanner graph) **to variable node** v_i.

Sum–Product Decoding Algorithm

1. **Init:** For every $i \in \{1, \ldots, n\}$ set $L^{[v]}_{i \to j} = \tilde{L}_i = L(y_i \mid X_i)$ for all $j \in \mathcal{N}(i)$.
2. **CN update:** For each check node c_j $(j \in \{1, \ldots, m\})$ compute

$$L^{[c]}_{i \leftarrow j} = 2 \tanh^{-1}\Big(\prod_{i' \in \mathcal{M}(j) \setminus \{i\}} \tanh\big(L^{[v]}_{i' \to j}/2\big)\Big), \quad \forall i \in \mathcal{M}(j).$$

3. **VN update:** For each variable node v_i $(i \in \{1, \ldots, n\})$ compute

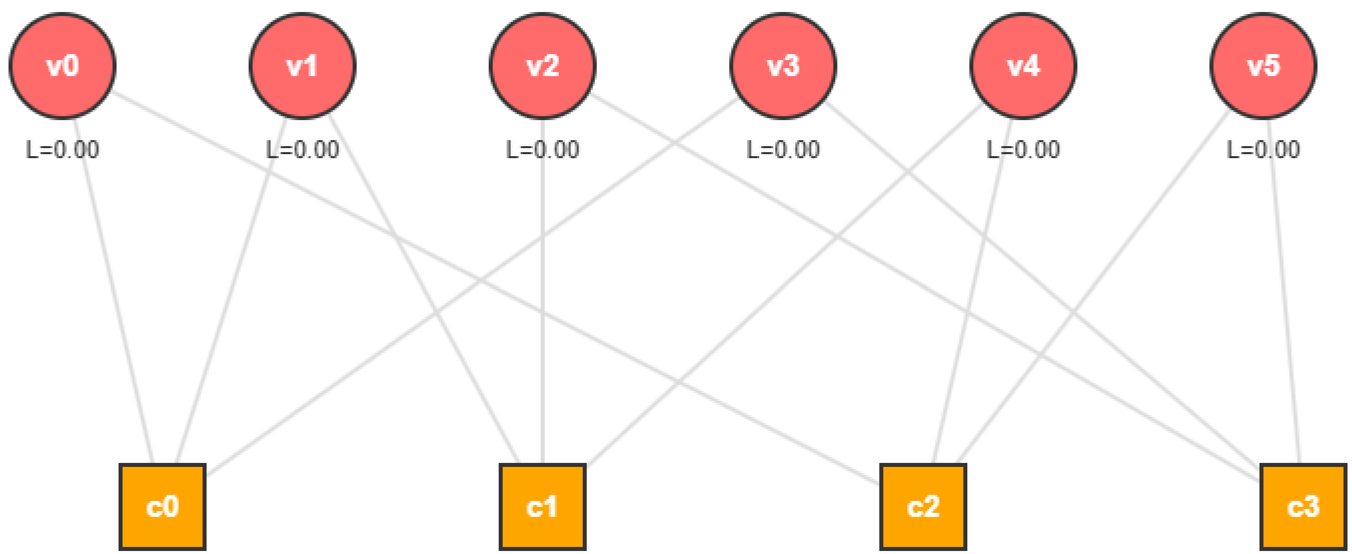
$$L^{[v]}_{i \to j} = \tilde{L}_i + \sum_{j' \in \mathcal{N}(i) \setminus \{j\}} L^{[c]}_{i \leftarrow j'}, \quad \forall j \in \mathcal{N}(i).$$

4. **Total LLR & decision:** $L^{[\text{total}]}_i = \tilde{L}_i + \sum_{j \in \mathcal{N}(i)} L^{[c]}_{i \leftarrow j}, \qquad \hat{x}_i = \begin{cases} 1, & L^{[\text{total}]}_i < 0, \\ 0, & \text{otherwise.} \end{cases}$

   Stop if $H\hat{\mathbf{x}}^T = 0$ **or** the iteration limit is reached; otherwise return to **step 2**.

Example - Channel LLRs for a BSC

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \implies$$



Example – Channel LLRs for a BSC

- For a BSC with crossover probability $p = 0.2$, assume:

$$\mathbf{x} = [0\ 0\ 1\ 0\ 1\ 1] \qquad \mathbf{y} = [1\ 0\ 1\ 0\ 1\ 1]$$

- The LLR for bit i is

$$\tilde{L}_i(x) = \log \frac{p(y \mid x=0)}{p(y \mid x=1)} = \begin{cases} \log \dfrac{p}{1-p}, & y_i = 1, \\ \log \dfrac{1-p}{p}, & y_i = 0. \end{cases}$$

- With $\log \dfrac{0.2}{0.8} = -1.3863$ and $\log \dfrac{0.8}{0.2} = +1.3863$ we get

$$\tilde{\mathbf{L}} = [\,-1.3863,\ 1.3863,\ -1.3863,\ 1.3863,\ -1.3863,\ -1.3863\,]$$

- These values are the initial messages $\tilde{L}_i$ fed into the decoder.

LDPC Decoding example

🔗 LDPC Sum-Product Decoder – Interactive Tanner Graph

Original Codeword (c)

0 0 1 0 1 1

Received Word (y)

1 0 1 0 1 1

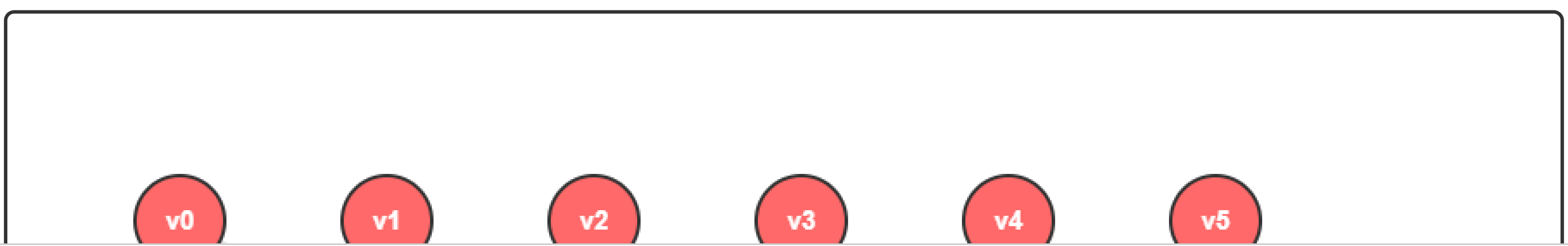
BSC Crossover Probability (p)

0.25

Maximum Iterations

6

Initialize Decoder Step Auto Decode Reset



6 Encoding/Decoding Schema

Encoding a single binary file

- File size: $192_{\text{KB}} = 192 \times 1000_{\text{B}} = 1536000_{\text{bits}}$
- Data is encrypted and compressed before encoding (to reduce homopolymers)
- Encoding steps from based on github code
- KB is decimal (1000 bytes = 1 KB), not binary (1024 bytes = 1 KiB)

Large block LDPC encoding

- LDPC data-block size (`LDPC_dim=256K`): 256000_{bits}
- Number of data blocks: $\frac{1536000}{256000} = 6$
- Added parity bits (`LDPC_alpha=0.5`): For each data block, we add 128000_{bits} of parity bits.
- Encoded bits per block:

$$256000_{\text{bits}} + 128000_{\text{bits}} = 384000_{\text{bits}}$$

Segment and map to DNA

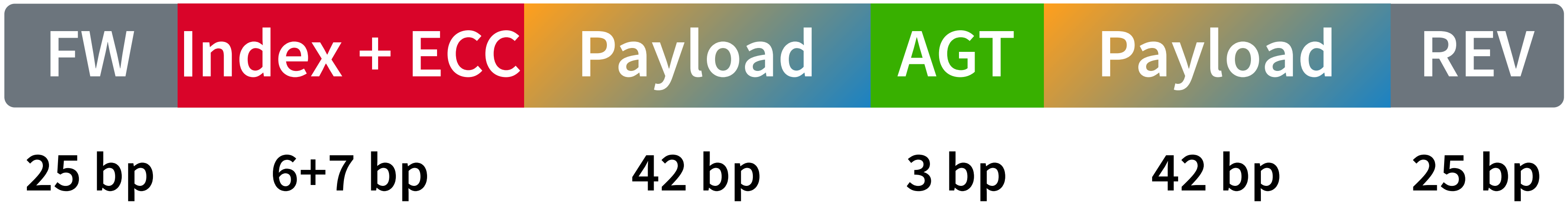
- Binary mapping: $00 \rightarrow A, 01 \rightarrow C, 10 \rightarrow G, 11 \rightarrow T$
- Bits per oligo (`payload size=84bp`): $84 \times 2_{\text{bits}} = 168_{\text{bits}}$
- Number of oligos per block:

$$\frac{\text{payload bits}}{\text{bits per oligo}} = \frac{384000_{\text{bits}}}{168_{\text{bits}}} = 2285.71 \approx 2286_{\text{oligos}}$$

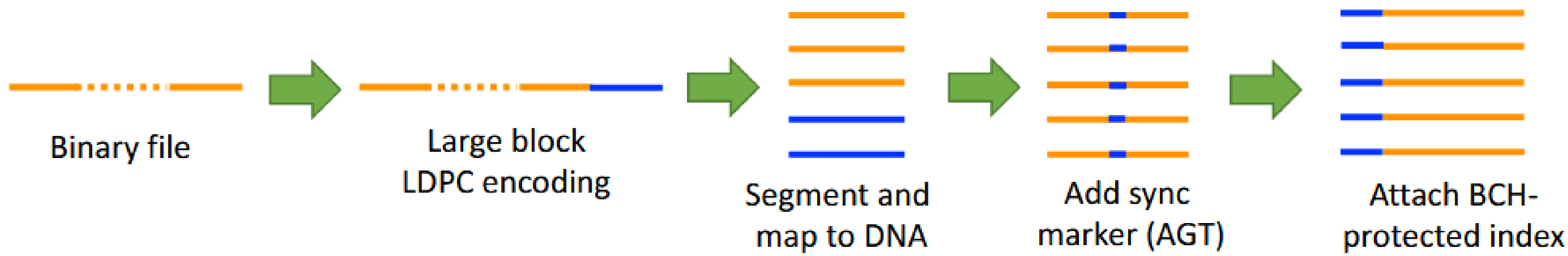
- Total number of oligos:
  $\text{blocks} \times \text{oligos per block} = 6 \times 2286 = 13716$ (✓)

Encoded oligo structure

Segment	bp
Index (pseudorandom permutation)	$14_{\text{bit}}/2 = 7_{\text{bp}}$
BCH redundancy	$12_{\text{bit}}/2 = 6_{\text{bp}}$
Sync marker (AGT)	3_{bp}
Payload	84_{bp}
Primers	$2 \times 25_{\text{bp}}$
Total	150_{bp}



Encoding schema overview



Challenges data storage in DNA

Error Type	Solution
1. Unordered reads	• Addressing index + BCH codes

2. Insertion/deletion errors	• Convert to substitution/erasure
	  • Sync markers
	  • MSA (via indexed clusters)

3. Substitution errors	• LDPC codes

Synchronization Marker

- **Goal:** Convert insertion/deletion errors into erasures for LDPC decoder
- If a read has unexpected length (due to indels), we try to **recover the marker** using Multiple Sequence Alignment (MSA).
- If the marker is located:
  - We **split** the read at the marker.
  - Retain only the expected-length portion.
  - Mark the rest as **erasures**.
- In simulations: 10% improvement in the reading cost while having little impact (2-3%) on the writing cost.

Multiple-Sequence Alignment (MSA)

- Cluster **per index bucket** (≈ 3–10 reads, 150 bp) → tiny, fast alignments.
- **Lines up indels** → gaps become **erasures (?)** instead of bad bits.
- Outputs **base counts**
- If consensus length off, use the in-strand **"AGT" marker** to trim half and mark the rest erasures.
- **Result:** LDPC sees a large code-word with *substitutions* + *explicit erasures* it can correct simultaneously.

```
1   -GCG-ACAT--
2   TACG-ACAT--
3   -GCGGACTTGG
4   -GCGGAATTGG
5   -GG----TCCG
```

Decode Schema

The Decode pipeline

1. **Primer trim & orientation** → raw reads
2. **Index + BCH decode**
    - correct up to 2–3 bit errors
    - single-indel heuristic recovers extra reads
3. **Per-index MSA (+ sync marker)**
    - aligns indels, outputs counts → LLRs
    - truncated halves → erasures
4. **Large-block LDPC decode**
    - fixes remaining substitutions **and** erasures (missing strands) in one shot

Decoding – how the counts become LLRs

After per-index MSA we have counts k_0, k_1 (gaps discarded).

$$\boxed{\mathrm{LLR}(k_0, k_1) \ = \ln \frac{P((k_0, k_1)|0)}{P((k_0, k_1)|1)} = \ (k_0 - k_1)\ \ln\left(\frac{1-\varepsilon}{\varepsilon}\right)}$$

- k_0, k_1 = how many times column voted “0” or “1”
- ε = post-MSA substitution rate (paper uses 4 %)
- Positive → bit likely **0**, negative → bit likely **1**, near 0 → **erasure**

These soft LLRs seed the LDPC **belief-propagation** decoder instead of hard 0/1 decisions.

7 Experimental results

Writing Cost c_w: what we pay to synthesize

$$c_w = \frac{\text{bases synthesized}}{\text{file size (bits)}}$$

- **Bases synthesized** = $\#\ \text{oligos} \times (\text{oligo length} - \text{primers})$

Unit: bases / information bit — lower is cheaper to write.

Reading Cost c_r: what we pay to read

1. **Physical coverage** – lab metric

$$\text{Raw coverage} = \frac{\text{total aligned reads}}{\text{distinct oligonucleotides}}$$

2. **Reading cost** c_r – information metric

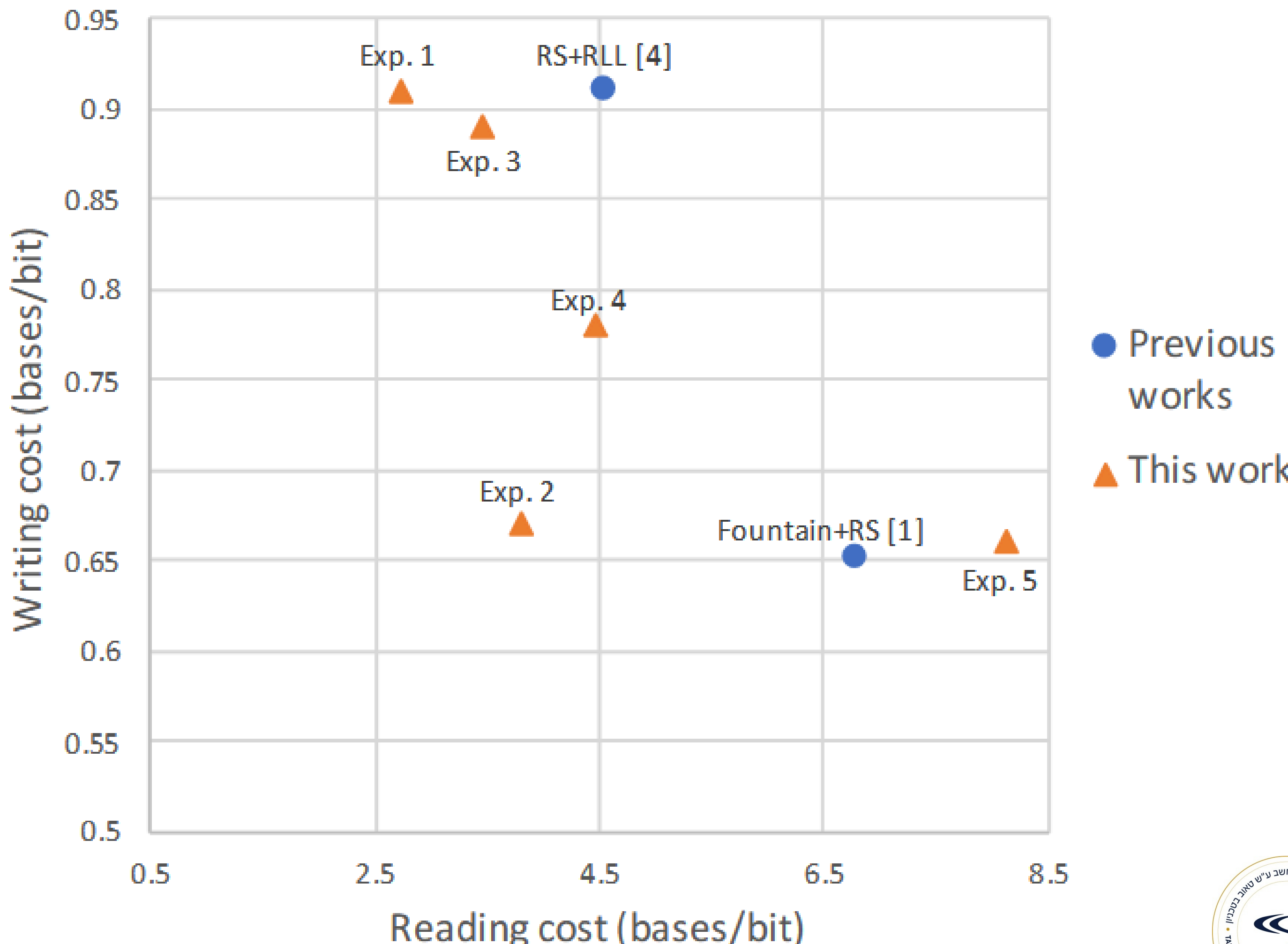
$$c_r = \frac{\text{bases sequenced for a guaranteed decode}}{\text{file size (bits)}}$$

Unit: bases / bit

How do we find the numerator: *Empirical threshold:* randomly subsample the read pool until decoding works **20 / 20 times** → that read count is “min reads for decode”.

Write/read costs (Fig. 8 / Table 1)

Exp. No.	LDPC Redundancy	File Size	No. of Oligonucleotides	Normalized Coverage	Writing Cost	Reading Cost
1	50%	160 KB				
2	10%	224 KB				
3	50%	192 KB				
4	30%	192 KB				
5	10%	192 KB				



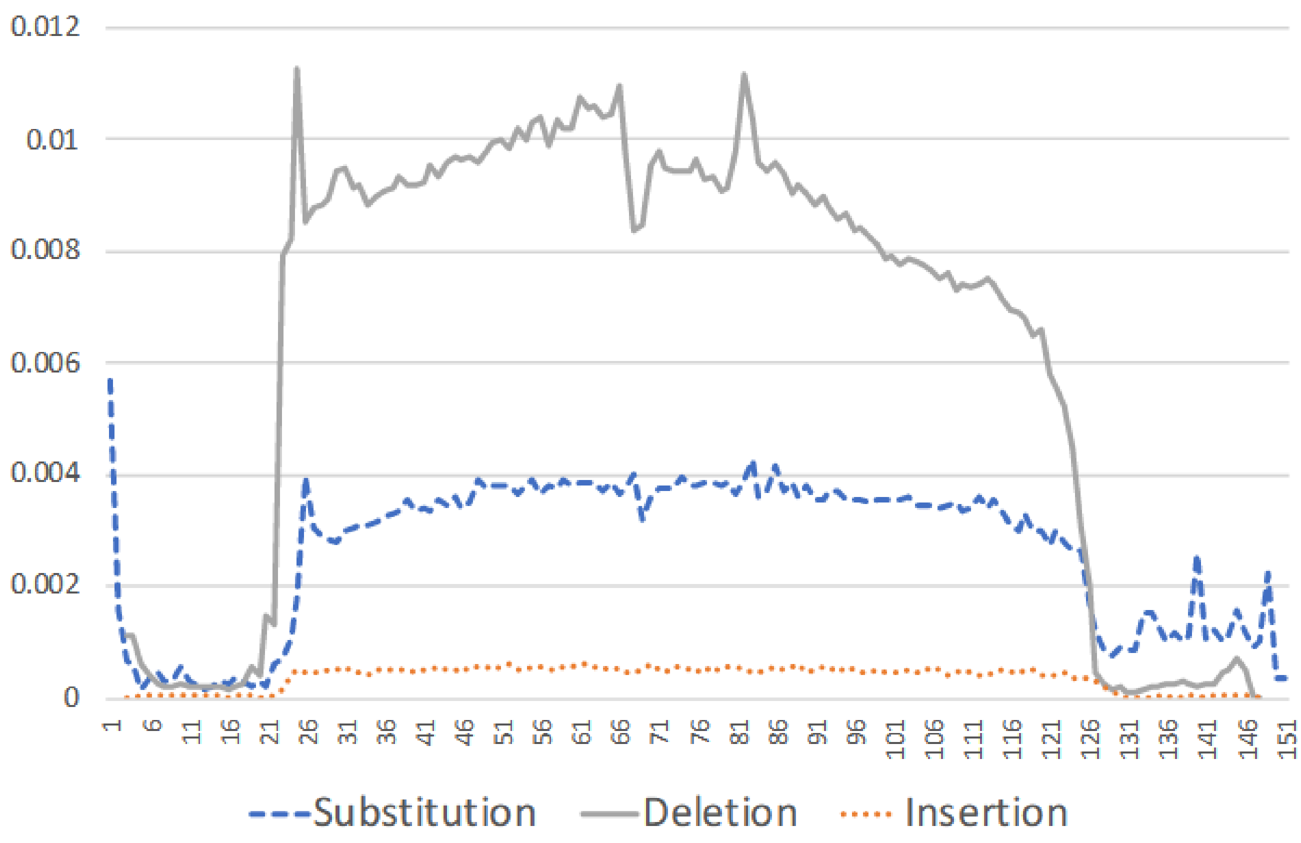
The ugly: experiment 9

Exp. No.	LDPC Redundancy	BCH Parameter	Index Length (bp)	Sync Marker	Sync Pos.	File Type	File Size (bytes)	No. of Oligos	Writi Cost (base
9	0.1	1	10	None	–	Text	286,432	14,094	0.62

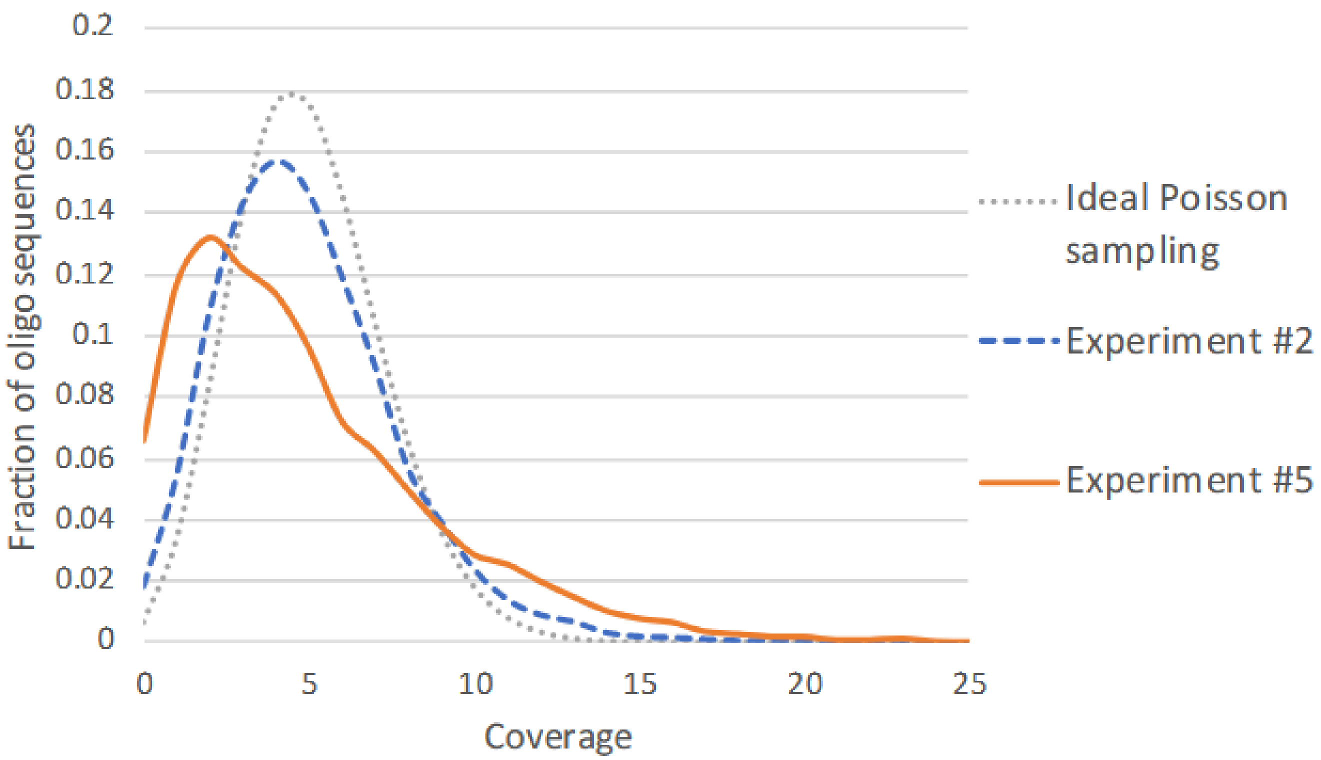
For **Experiment 9,** which utilized the weakest error correction settings (0.1 LDPC Redundancy), a definitive “minimum coverage for decoding” is **not reported**.

- The decoding process **did not succeed for 20 out of 20 trials**, even when very high reading costs were applied
- While decoding *did* succeed for some fraction of trials at reading costs above 10 bases/bit, it did not meet the consistent success criteria (20/20 trials) required for reporting a minimum value

Error profile (Fig. 9)



Coverage profile (Fig. 10)



Stress testing

- Simulated **6% total error** (2% sub / 2% del / 2% ins)
- Added **15% random reads** (unaligned noise)
- **224 KB** file, **50% LDPC**, BCH (3-error correction)
- Increased LDPC decoder threshold to **10%**
- **Write cost**: 1.07 bases/bit
- **Decoding succeeded** at **10.5 bases/bit read cost**

8 Concluding Remarks

Conclusions

- Achieves **better read/write cost tradeoff** than prior work
- Combines **LDPC codes** with **heuristics** for indel correction
- Insights may help improve **bioinformatics tools** and **error models**

Future Work

- Use **channel-optimized LDPC** or **marker codes**
- Extend to **nanopore sequencing** (high indel rates)
- Improve **index error correction efficiency**

Limitations

- **Random access** No range queries or selective decoding
- **Counting on heuristics** for indel correction, which may not generalize well
- **Error model** Assumes independent errors, which may not hold in practice
- **Relying on simulations** for performance evaluation, which may not fully capture real-world complexities

In the words of Robert Gallager

LDPC was an interesting midpoint, cute and interesting to theoriticanats but 35 years before the technological feasibility. That's the way research is and the way it should be. Hard research problems take years to solve and should not be overly dependant on details of current technological capabilities … Applications resolve when both are ready.

Dr. Robert Gallager, 2019

Thank you for your attention!

Questions?

Crafting the Presentation: Tools

- `Quarto`: markdown-based authoring system that supports multiple output formats.
- `revealjs`: a framework for creating interactive presentations using HTML and JavaScript.
  - `simplemenu`: a plugin to create a menu bar that allows us to navigate through the presentation.