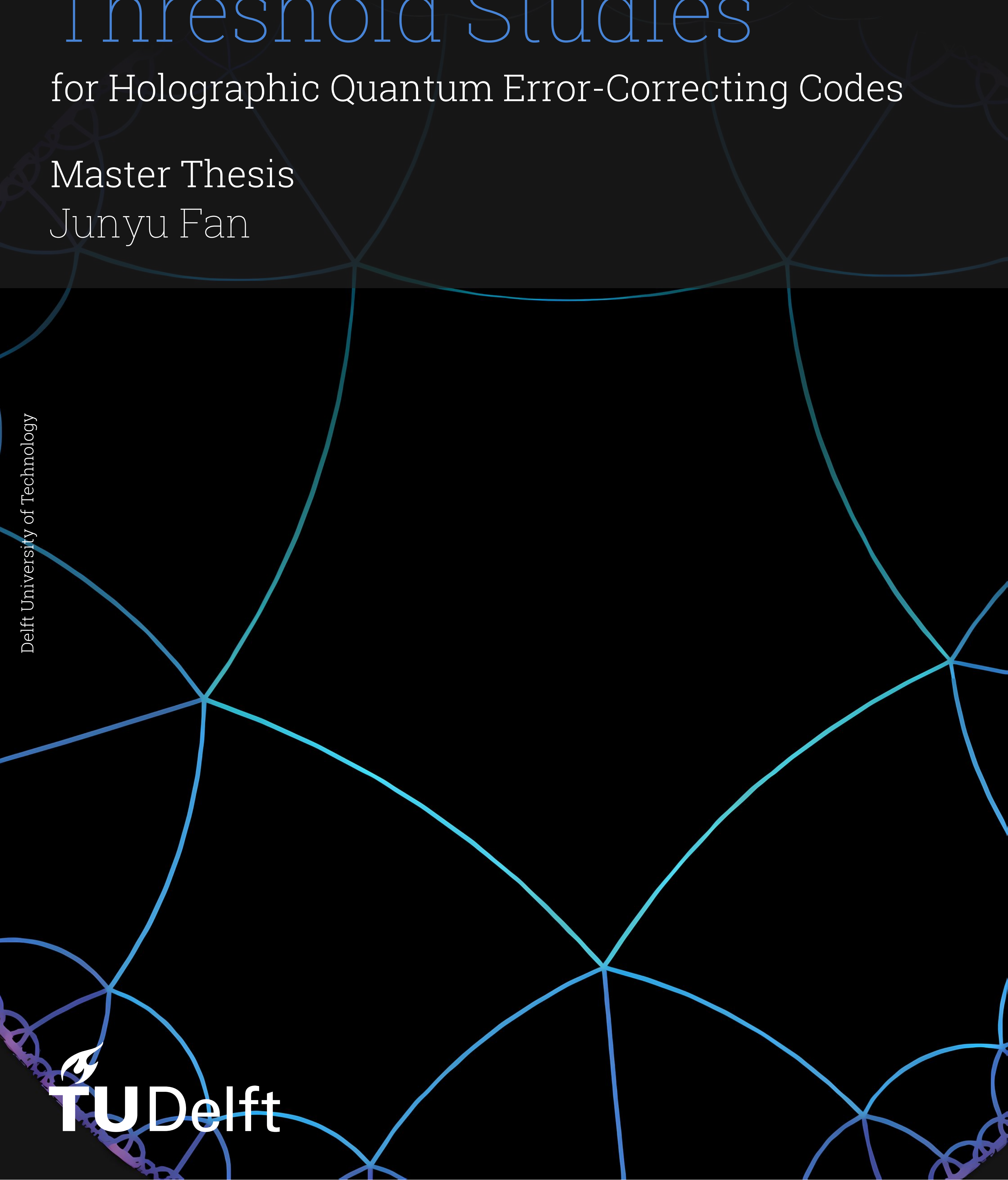
Biased-Noise Threshold Studies

for Holographic Quantum Error-Correcting Codes

Master Thesis

Junyu Fan

Delft University of Technology



TUDelft

Biased-Noise Threshold Studies

for Holographic Quantum Error-Correcting Codes

by

Junyu Fan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday May 31, 2024 at 1:00 PM.

Student number:	5726263
Project duration:	October 2023 - April 2024
Thesis committee:	Prof. Sebastian Feld,
	Prof. Barbara Terhal,
	Prof. Eliška Greplová,

Cover : Generated by Poincaré tiling on `https://malinc.se/`.
Keywords: quantum error correction, biased-noise, holographic quantum error correction, AdS/CFT, tensor networks, thresholds

Summary

The differences between T_1 and T_2 in real-world quantum computing platforms underscore the importance of studying the thresholds of quantum error-correcting codes under biased noise, also spurring active searches for error-correcting codes with thresholds exceeding the hashing bound under biased noise. Recently, new error-correcting codes such as the XZZX code [1] and the holographic seven-qubit tailored code [2] have exhibited a 50% threshold under pure Pauli noise. Notably, the XZZX code achieves a threshold exceeding the hashing bound in cases of high bias.

This work reports on a holographic quantum error-correcting code, the HaPPY code [3], which also exhibits a 50% threshold under pure Pauli noise and surpasses the hashing bound threshold under high biased noise. Additionally, this work also explores the threshold of the holographic Steane code under biased noise for comparison.

In addition to studying thresholds under biased noise, this work also investigates the thresholds of various codes, including the Hyper-Invariant Tensor-Network code (HTN code) [4], holographic Reed-Muller code [5], and some heterogeneous holographic codes, under quantum erasure channels and depolarizing channels.

This work has developed an automated quantum tensor network operator push [5] program, which supports the automated generation of stabilizers and complete logical operators for tensor network quantum error-correcting codes. This greatly enhances the research efficiency of holographic codes, and the program is now ready to be made available to the open-source community.

Contents

1 Introduction

1.1. Research Background and Motivation

Quantum computing, with its inherent high parallelism, has the potential to achieve computational power far exceeding classical computing for certain types of problems [6–8]. Since the concept of quantum computing was introduced by Richard Feynman [9–11], the theoretical framework for quantum computing has been continuously expanded and refined, and many different directions have emerged in the exploration of physical platforms for quantum computing [12–14].

In the current physical platforms for quantum computing, the main focus for improvement lies in scalability and the quality of quantum bits (qubits). The quality of qubits mainly includes the fidelity of various operations, relaxation time T_1, and coherence time T_2. Maintaining the protection of quantum states over a long period on actual physical devices remains a challenging problem, as many factors can cause quantum information to become unreliable [15, 16]. The current quality of qubits is not sufficient to support the needs of reliable quantum computing [17, 18] . Therefore, improving the quality of qubits is an urgent issue, and one viable solution is the use of quantum error correction.

In quantum error correction, a commonly used error model is the Pauli error model [19], where Pauli operators X, Y, and Z are used to model errors occurring on qubits. When the probabilities of the three types of Pauli errors are equal, it is called a depolarizing error channel. When the probabilities of the three types of Pauli errors are unequal, it is referred to as a biased noise channel. In real-world quantum computing physical platforms [20], noise is not uniformly distributed among Pauli X, Y, and Z errors.

The threshold [21] is an important indicator for measuring the error correction performance of a quantum error-correcting code. Therefore, studying the threshold of quantum error-correcting codes under biased noise and searching for quantum error-correcting codes with high threshold under biased noise can be very meaningful.

Holographic codes, inspired by the AdS/CFT duality [22], are a type of quantum error-correcting code capable of modeling certain properties of the AdS/CFT duality. Research on the error correction properties of holographic codes under biased noise is still relatively unexplored, making it highly meaningful to study the error correction performance of holographic codes under biased noise.

Holographic codes also represent a broad category of error-correcting codes, many fundamental properties of which have not yet been explored. Meanwhile, heterogeneous concatenated codes [23] have been shown to support more transversal gate operations and exhibit thresholds [24, 25], making the properties of heterogeneous holographic codes also worthy of investigation.

1.2. General Research Goals

Based on the above background, the general research goals of this work are:

1. Currently, holographic codes such as the HaPPY code and the holographic Steane code have been studied for their thresholds in depolarizing channels. This work will investigate their thresholds under biased noise.
2. For some newer holographic codes that have not been studied for their threshold performance, such as the HTN code [4] and the holographic Reed Muller code [5], this work will preliminarily investigate their thresholds under the quantum erasure channel and the depolarizing channel.
3. This work will also propose some heterogeneous holographic codes and study their thresholds under the quantum erasure channel.

1.3. Structure of the Thesis

Based on these research goals, this work aims to provide results on the thresholds of holographic codes under different noise channels, primarily biased noise channels.

To achieve these goals, this thesis will start by introducing the background concepts necessary for understanding holographic codes. It will then proceed to the methods section, detailing the tools and research methods developed and used in this work to study the thresholds of holographic codes. Finally, the results section will present the findings of this research.

The background section of this thesis starts with the basics of classical error correction, then moves on to introduce quantum error correction, followed by an exploration of the connections between tensor networks and quantum error-correcting codes. It then introduces the theoretical background of the AdS/CFT conjecture [22, 26] and its connections with quantum information, leading to the introduction of the concept of holographic quantum error-correcting codes and discussing some typical holographic codes. Finally, it covers the quantum LEGO [5] and operator-pushing rules, explaining how to obtain the operators for holographic codes.

The methods section of the thesis first introduces the functionality and architecture of an automated operator generation program for holographic codes, developed based on the quantum lego and operator-pushing rules. It then describes a quantum erasure decoder used to assess the recoverability from quantum erasure errors. Next, it introduces three types of quantum error correction decoders for Pauli errors: the first is a minimum weight decoder based on integer optimization, known as the integer optimization decoder; the second is a variable weight integer optimization decoder, which is an adaptation of the integer optimization decoder optimized for biased noise channels; and the third is a tensor network-based maximum likelihood decoder.

The results section of the thesis first reports on the threshold performance of several holographic codes under biased noise channels using the variable integer optimization decoder and tensor network decoder, and compares these results to the hashing bound. It then presents the threshold performance of various types of HTN codes under quantum erasure channels, depolarizing channels, and pure Pauli noise channels, as well as the distance calculations for different sizes of HTN codes. The section also reports on the threshold performance of the holographic Reed Muller code under quantum erasure errors. Finally, it discusses the threshold behaviors of various heterogeneous holographic codes under quantum erasure errors.

2
Background

2.1. Classical Error Correction

In classical digital computers, information is fundamentally processed, transmitted, and stored in binary form. Although there exist some data transmission and storage technologies (e.g. Pulse Amplitude Modulation), which are not strictly binary, ultimately, the data from these technologies are decoded into binary form for processing.

These binary digits used to store 0s and 1s are called bits, and they are the fundamental units of information in classical digital computers. However, computers may experience errors in information due to various reasons, such as external electromagnetic interference, timing errors due to inaccurate clocks, and so on. We can abstract the errors occurring on bits as bit-flip operations. When the errors are not too severe, we hope to detect and correct these errors through classical error correction to make our computers more reliable.

A bit flip operation results in the inversion of the bit's information.

$$\hat{U}: \ 0 \to 1, \ 1 \to 0 \tag{2.1}$$

Numerous error-correcting codes have been proposed for the detection and correction of bit flip errors.

2.1.1. Majority Vote: Classical Repetition Codes

A very intuitive method involves repeating the original binary information three times [27]. Let's assume that the information we want to encode is denoted as S.

$$S = 010011101100 \tag{2.2}$$

After encoding using the repetition code, the information is denoted as $\bar{S}$:

$$\bar{S} = \begin{bmatrix} 010011101100 \\ 010011101100 \\ 010011101100 \end{bmatrix} \tag{2.3}$$

This way, when a bit flip error occurs at a certain position, for instance, at the j-th bit of the i-th set of the repeated information $\hat{U}_{ij}\bar{S}$, we can correct the erroneous bit by comparing the information of this specific bit across the three sets and employing a majority vote method to rectify the bit that underwent the flip.

$$\hat{U}_{i=2,j=5}\bar{S} = \begin{bmatrix} 010011101100 \\ 010001101100 \\ 010011101100 \end{bmatrix} \rightarrow \hat{U}^2_{i=2,j=5}\bar{S} = \bar{S} = \begin{bmatrix} 010011101100 \\ 010011101100 \\ 010011101100 \end{bmatrix} \tag{2.4}$$

We can see that, while repetition codes are effective at detecting and correcting bit flip errors, they come with a cost: only 1/3 of the transmitted information is effective, and the remaining 2/3 is redundant. Encoding information in this manner would lead to a significant waste of storage space and bandwidth. Therefore, we must seek smarter ways to detect errors.

2.1.2. Space-Saving: Parity Check

To enhance information protection with fewer error correction bits, parity checks [28] offer a practical solution. It involves appending an extra error correction bit, P, to the original binary message, forming a combined sequence $\{P|S\}$. For instance:

$$\{P|S\} : \{0|010011101100\} \tag{2.5}$$

The underlying principle of this parity check error correction code is to maintain an even total count of 1s. Consequently, if the received message contains an odd number of 1s, a single bit flip error is indicated, prompting a request for message retransmission.

However, it's noteworthy that while parity checks can detect single bit flip errors, they lack the capability for immediate error correction. This limitation renders simple parity check codes suboptimal for practical scenarios where immediate error correction is preferable.

2.1.3. Principle of Inclusion-Exclusion: Classical Hamming Code

Strategically arranging parity checks can pinpoint the error's location, with the Hamming code [29, 30] serving as a prime example. In this method, 16 bits are arranged in a matrix as depicted below:

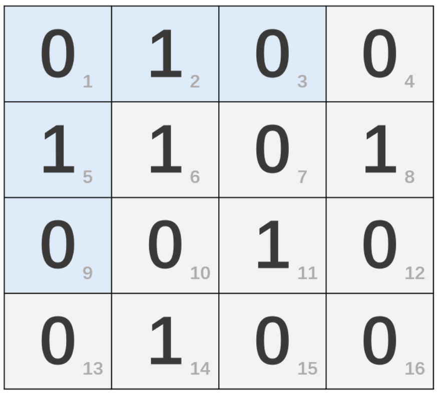


Figure 2.1: An example of a Hamming code with a total of 16 bits, where the light blue bits are parity check bits that store the parity information for different regions of bits.

Bits 1, 2, 3, 5, and 9 are designated as parity checks. Bit 1 ensures overall matrix parity, bits 2 and 3 secure parity for columns 2, 4 and 3, 4 respectively, while bits 5 and 9 maintain parity for rows 2, 4 and 3, 4 respectively. This arrangement not only identifies the location of bit flip errors but also enables immediate correction. The Hamming code is capable of detecting up to two bit flip errors and correcting a single error.

Furthermore, this concept can be represented in matrix form, known as the parity check matrix.

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{2.6}$$

Each row of this matrix represents a parity check, where an element of 1 indicates inclusion of that bit in the parity check. In other words, the parity check incorporates this bit when calculating the parity (odd or even) of the total number of 1 elements. Consequently, a generalized parity check matrix can be denoted as H, where $\boldsymbol{h}_i$ represents a specific parity check.

$$H = \begin{pmatrix} \boldsymbol{h}_1 \\ \boldsymbol{h}_2 \\ ... \\ \boldsymbol{h}_{m-1} \\ \boldsymbol{h}_m \end{pmatrix} \tag{2.7}$$

By applying the matrix H to the binary codeword $\boldsymbol{c}$ and performing modulo-2 matrix multiplication, we can obtain the results of all parity checks, which we refer to as the syndrome γ. This syndrome γ is then used to locate and correct the erroneous bits.

$$\gamma = H\boldsymbol{c} \tag{2.8}$$

A significant advantage of Hamming code lies in its ability to locate and correct errors directly through simple logic gates [31], resulting in minimal computational overhead for decoding. Additionally, there are error-correcting codes like LDPC (Low-Density Parity-Check) that are widely employed in modern classical computing [32–36].

2.1.4. Classical Linear Stabilizers Codes

For classical stabilizer codes [37], they are denoted by $[n, k]$, where n and k indicate that the code encodes k bits into n bits of error-correcting code (always having $n > k$). In the code space C, the Hamming distance between any two different codewords y and z is at least d. The Hamming distance between codewords y and z is the number of positions where their elements (0 or 1) differ. A $[n, k]$ code with a minimum distance d can be referred to as an $[n, k, d]$ code. Such a code can correct any bit flips that are less than half of the code's minimum distance. Specifically, a binary $[n, k]$ linear code (with elements only 0 or 1) actually forms a k-dimensional subspace of $\mathbb{Z}_2^n$. In this case, the minimum distance of the code is the same as the minimum Hamming weight (or number of 1s) of any non-zero codeword.

Generator Matrix

The generator matrix G of a linear code C of $[n, k]$ is an $n \times k$ matrix defined over $\mathbb{Z}_2$. This matrix defines the range of the code C:

$$C = \text{range}(G) = \{Gx : x \in \mathbb{Z}_2^k\}. \tag{2.9}$$

To obtain the codeword y from the original information x, simply multiply x by the generator matrix G:

$$y = Gx \in \mathbb{Z}_2^n \tag{2.10}$$

This process actually maps the original information x into a higher-dimensional space, thereby potentially incorporating the capability for error detection and correction.

Parity-Check Matrix

The parity-check matrix H for a classical linear stabilizer code $[n,k]$ is an $(n-k)\times n$ matrix that operates over $\mathbb{Z}_2$:

$$C = \ker(K) = \{y \in \mathbb{Z}_2^n : Hy = 0\}. \tag{2.11}$$

For a given codeword $y' \in \mathbb{Z}_2^n$, its syndrome is $Hy' \in \mathbb{Z}_2^{n-k}$. The syndrome can be used to infer the locations of errors and thus to correct the codeword.

Dual Codes

For a linear code C of $[n,k]$, its dual code, denoted as $C^\perp$, includes all strings whose modulo 2 inner product with any codeword in C is zero:

$$C^\perp = \{y \in \mathbb{Z}_2^n : y \cdot x = 0 \text{ for all } x \in C\}. \tag{2.12}$$

$C^\perp$ is a $[n, n-k]$ linear code. If we have the generator matrix G and the parity-check matrix H for C, then the generator matrix for $C^\perp$ is the transpose of H (H^T), and the parity-check matrix is the transpose of G (G^T). If a code C satisfies $C \subseteq C^\perp$, it is called weakly self-dual; if $C = C^\perp$, it is called self-dual.

2.2. Quantum Error Correction

In quantum computing, a fundamental computing unit is called a qubit [19]. It represents a normalized quantum state within the Hilbert space spanned by two mutually orthogonal states.

Typically, we denote these two mutually orthogonal states as $|0\rangle$ and $|1\rangle$. Consequently, the pure state of a qubit can be expressed in the following form:

$$|\psi\rangle = e^{i\delta}(cos(\theta/2)|0\rangle + sin(\theta/2)e^{i\phi}|1\rangle) \tag{2.13}$$

In this context, θ is polar angle, ϕ is azimuthal angle, and $e^{i\delta}$ is a global phase. For a single qubit, people are often more concerned with the polar angle and the azimuthal angle. Therefore, a sphere, called the Bloch sphere, is used to represent the state of the qubit. This representation visually encapsulates the state's position in its two-dimensional Hilbert space.

2.2.1. Error Modeling

From this, we can deduce that describing a quantum state on the Bloch sphere requires two degrees of freedom. Consequently, the errors that can occur in a qubit are not limited to the simple bit flips found in classical cases. The relative phase between the quantum states $|0\rangle$ and $|1\rangle$ is also crucial, making phase flips a significant type of quantum error. Both of these errors, bit flips and phase flips, can be regarded as manifestations of Pauli operators [38].

$$Bit\ Flip : X|0\rangle = |1\rangle,\ X|1\rangle = |0\rangle \tag{2.14}$$

$$Phase\ Flip : Z|0\rangle = +1|0\rangle,\ Z|1\rangle = -1|1\rangle \tag{2.15}$$

Certainly, besides these quantum errors that can be represented by Pauli operators, there are other types of quantum errors as well. For example, some actual quantum devices are not strictly two-level systems, and quantum states may be excited to states beyond the defined $|0\rangle$ and $|1\rangle$ states. This type of error is known as leakage [19].

Single Pauli Error Channel

A typical error model is the single Pauli error model, defined as follows:

$$\mathcal{E}(\rho) = (1-p)\rho + p(r_x X\rho X + r_y Y\rho Y + r_z Z\rho Z) \tag{2.16}$$

Where ρ is the density matrix, p is the probability of a Pauli error occurring, and r_x, r_y, r_z are the proportions of the various Pauli errors occurring, with $r_x + r_y + r_z = 1$.

When $r_x = r_y = r_z = \frac{1}{3}$, this channel is known as the depolarizing channel; in other cases, it is referred to as a biased noise channel.

Using X (phase flip), Z (bit flip), and Y (both phase and bit flip) Pauli operators to model quantum errors discretely might seem insufficient because the movement of states on the Bloch sphere (considering only pure states) can be continuous, and X, Z, and Y operators cannot describe all possible movements on the Bloch sphere. However, using X, Z, and Y errors to model single qubit errors is a common practice and is practically feasible. Here is an example to illustrate:

A rotation on the Bloch sphere can be represented as:

$$\hat{R}_n(\theta) = \cos(\theta/2)\hat{I} - i\sin(\theta/2)n \cdot \sigma \tag{2.17}$$

Where:

$$n \cdot \sigma = n_x\hat{X} + n_y\hat{Y} + n_z\hat{Z} \tag{2.18}$$

θ is the angle of rotation around the axis (n_x, n_y, n_z). For example, when the rotation for the first qubit of a quantum error-correcting code is around the axis $(n_x, n_y, n_z) = (1, 0, 0)$, the rotation is:

$$\hat{R}_n(\theta) = \cos(\theta/2)\hat{I}_1 - i\sin(\theta/2)\hat{X}_1 \tag{2.19}$$

Assuming the entire error-correcting code's state is $|\Psi\rangle$, then after this rotation, the entire system's quantum state is:

$$|\Psi'\rangle = (\cos(\theta/2)\hat{I}_1 - i\sin(\theta/2)\hat{X}_1)|\Psi\rangle \tag{2.20}$$

Assuming there is an ancillary qubit whose stabilizer includes a Z check at the position of qubit 1, at this point, the ancillary qubit and qubit 1 are actually in an entangled state. Measuring the ancillary qubit will actually affect the state of qubit 1. However, the measurement of the ancillary qubit will cause the entire error-correcting code system $|\Psi'\rangle$ to collapse to the error-free state $|\Psi\rangle$ with a probability of $\cos^2(\theta/2)$, and to a state with an error $\hat{X}|\Psi\rangle$ with a probability of $\sin^2(\theta/2)$, thereby allowing for error correction [38].

Therefore, for errors that keep the quantum state on the Bloch sphere, it is reasonable to model these errors using X, Y, and Z operators.

Quantum Erasure Error Channel

During the process of quantum communication and computing, quantum information may be lost for various reasons (for example, the loss of photons in quantum optical communication, or qubits transitioning to non-computational energy levels known as leakage). These losses occur with certain probabilities, and at the time of the final readout of quantum information, we explicitly know which qubits' information has been lost. As illustrated in the diagram, this channel is referred to as the quantum erasure channel [39–42]:

$$\rho \rightarrow (1 - \epsilon)\rho + \epsilon\,|e\rangle\,\langle e| \tag{2.21}$$

Where ϵ is the probability of quantum erasure errors, and ρ is the density matrix. An example of a quantum erasure channel for the transmission of 4 qubits is shown in the figure.

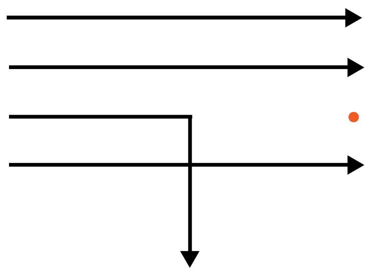


Figure 2.2: Example of a quantum erasure channel: Quantum information of 4 qubits is sent from the left to the right side. During transmission, one qubit is lost, so only 3 qubits are received on the right side, and it is known which qubit was lost.

2.2.2. No Simple Majority Vote: Quantum No-cloning Theorem

Lemma 2.2.1. *Known as the **Quantum No-cloning Theorem** [43], it is impossible to clone the quantum state of a qubit using unitary transformation.*

Proof. Given the qubit to be cloned is $|\psi\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.22}$$

And the initial state of second qubit and the environment is at some reference states, for instance $|\phi\rangle$ and $|A_i\rangle$. Suppose a unitary operation $\hat{U}$ is able to clone the quantum state of the first qubit to the second qubit:

$$U(|\psi\rangle|\phi\rangle|A_i\rangle) = |\psi\rangle|\psi\rangle|A_{f\psi}\rangle = (\alpha|0\rangle + \beta|1\rangle)(\alpha|0\rangle + \beta|1\rangle)|A_{f\psi}\rangle \tag{2.23}$$

The final state of the environment will in general depend on the state to be cloned. Let's pick an example that the first qubir is at the state $|0\rangle$, then the cloning action is:

$$U(|0\rangle|\phi\rangle|A_i\rangle) = |0\rangle|0\rangle|A_{f0}\rangle \tag{2.24}$$

Similarly, if the first qubit to be cloned is at the state $|1\rangle$

$$U(|1\rangle|\phi\rangle|A_i\rangle) = |1\rangle|1\rangle|A_{f1}\rangle \tag{2.25}$$

Therefore, the action of cloning a generic state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is:

$$U((\alpha|0\rangle + \beta|1\rangle)|\phi\rangle|A_i\rangle) = \alpha U(|0\rangle|\phi\rangle|A_i\rangle) + \beta U(|1\rangle|\phi\rangle|A_i\rangle) \tag{2.26}$$

Now insert Eq.2.24 and Eq.2.25 into this equation, we get:

$$\alpha|0\rangle|0\rangle|A_{f0}\rangle + \beta|1\rangle|1\rangle|A_{f1}\rangle \tag{2.27}$$

Which is different from the desired cloned state of Eq.2.23.

2.2.3. Entangled Majority Vote: Quantum Repetition Code

A very intuitive idea is to emulate classical repetition codes by constructing quantum repetition codes [44]. However, the cautious would immediately recall the quantum no-cloning theorem and question the feasibility of this approach. Indeed, quantum repetition codes do not involve cloning the original single quantum state. Instead, they encode it into an entangled state, with the state vector being $|000\rangle$ and $|111\rangle$.

The encoding and decoding circuit for quantum repetition codes is depicted in the following picture.

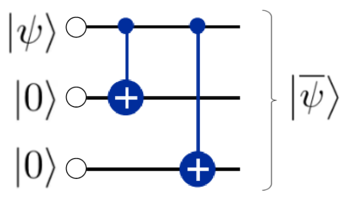


Figure 2.3: An encoding circuit for a 3-qubits quantum repetition code, The top qubit starts with a quantum state $|\psi\rangle$, which is the state we want to encode using the repetition code. The middle and bottom qubits each start with the state $|0\rangle$. Two CNOT gates are used to set the second qubit and the third qubit in entanglement with the first qubit.

After encoding, the state is as follows:

$$|\psi\rangle = \cos\theta|0\rangle + e^{i\phi}\sin\theta|1\rangle \rightarrow |\bar{\psi}\rangle = \cos\theta|\bar{0}\rangle + e^{i\phi}\sin\theta|\bar{1}\rangle \tag{2.28}$$

Direct measurement of a qubit leads to state collapse, destroying quantum coherence. To circumvent this, we employ parity checks to extract error information. Since we measure the parity information of qubits, we avoid collapsing the original qubit back to its eigenstate. As illustrated below, we introduce two ancilla qubits to store the parity information. Subsequently, we measure the information from these two parity-check ancillas. This forms the simplest decoding circuit for quantum repetition coding.

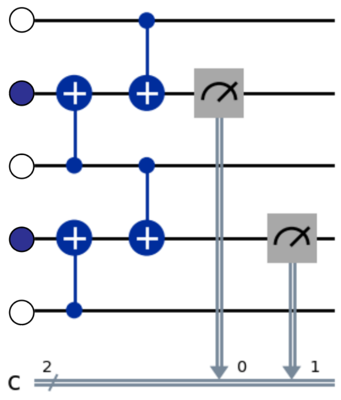


Figure 2.4: Quantum repetition code decoding circuit, where the blue qubits are ancillary qubits, used to store parity check information from their neighboring qubits. This parity check information is transferred to the ancillary qubits via a set of CNOT gates, followed by the measurement of the two ancillary qubits, thereby obtaining the syndrome.

Beyond the three-qubit quantum repetition code example, encoding n qubits into a quantum repetition code follows a similar process.

$$|+\rangle = \frac{1}{\sqrt{2^n - 1}} \sum_{\sum v_i = 0} |v_1, \ldots, v_n\rangle \tag{2.29}$$

$$|-\rangle = \frac{1}{\sqrt{2^n - 1}} \sum_{\sum v_i = 1} |v_1, \ldots, v_n\rangle \tag{2.30}$$

Quantum repetition codes are limited to detecting a single error type; bit-flip codes identify X errors in $(n-1)/2$ qubits without recognizing Z phase-flips, while phase-flip codes discern Z errors in $(n-1)/2$ qubits but are blind to X bit-flips.

2.2.4. To Enhance Error-Correcting Performance: Code Concatenation

Code concatenation [45–47] is a technique that can enhance a code's error-correcting capabilities, where one or more different quantum error-correcting codes are "chained" together, thereby achieving higher fidelity at lower single-qubit error rates. This technique stems from a straightforward idea: if an error-correcting code can achieve a logical error rate $p_e^{\text{log}}(p)$ that is lower than the physical qubit error rate p, then it means that one can further encode the original error-correcting code using the same error-correcting method to obtain an even lower logical error rate under the same physical qubit error rate.

This basic version of code concatenation is referred to as "tree-like" code concatenation in this work. It re-encodes the original physical qubits into an increased number of physical qubits, thereby enhancing redundancy and improving error correction capabilities.

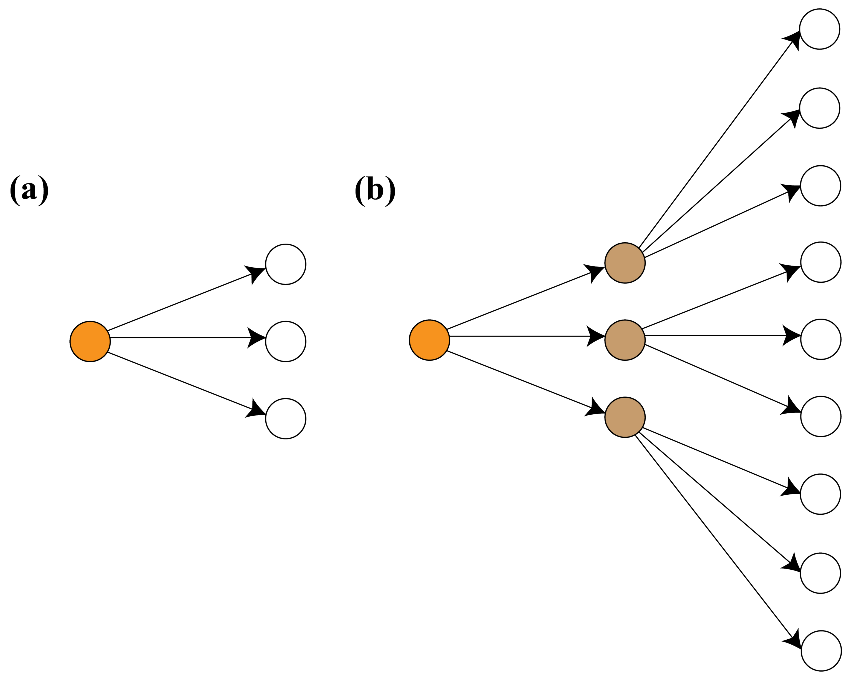


Figure 2.5: An example of code concatenation based on the $[[3,1,d]]$ code: (a) A $[[3,1,d]]$ code encodes one orange logical qubit into three white physical qubits. (b) The original three white physical qubits are then treated as new "logical" qubits (they are not truly the logical qubits of the entire code, hence they are depicted in brown) and are once again encoded using the $[[3,1,d]]$ code into a total of nine new physical qubits. This type of code concatenation is referred to as "tree-like" code concatenation in this work, with its network structure presenting a simple tree-like architecture.

Code concatenation also has many additional benefits, such as expanding the code's ability to correct different types of errors. For example, as will be discussed below, the Shor 9-qubit code [48, 49] benefits from code concatenation, which helps enhance its error-correcting capabilities for various types of errors. Additionally, the Eastin–Knill theorem [50] states that no quantum error-correcting code can allow for the transversal implementation of a universal gate set, where a transversal logical gate is one that can be executed independently on each physical qubit. However, concatenated codes can exploit the transversal properties of two different codes to transversally implement a universal gate set with an additional round of error correction [23]. This approach avoids the need for extra ancillary state preparation, such as operations like magic state distillation, to achieve universality. Additionally, concatenated codes are also very useful for magic state distillation [51, 52].

2.2.5. From Code Concatenation to the Threshold Theorem

The idea behind code concatenation to enhance error correction capabilities is that if an error-correcting code at a certain physical error rate p can provide a fidelity $F_{ec}(p)$ higher than $1-p$, then the same code, which can also deliver fidelity above $1-p$ at this error rate p, can be used again to re-encode the original code, thus achieving higher fidelity. A simple example is using the same code for one round of code concatenation, resulting in the concatenated code having a fidelity $F_{conc} = F(1 - F_{ec}(p))$. When p is sufficiently small, $F(p) > 1 - p$, and $F(p)$ is monotonically decreasing within the range of interest, the concatenated code will have improved fidelity.

As shown in the figure, for a small p, if $F_{ec}(p) > 1 - p$, then we can re-encode the physical qubits of this code to achieve better fidelity. Conversely, if the physical error rate p is large, then the concatenated code will have worse fidelity. By estimating the fidelity of concatenation for all physical error rates p as shown in figure 2.6.c, we can obtain the fidelity curve of the concatenated code as depicted in figure 2.6.d.

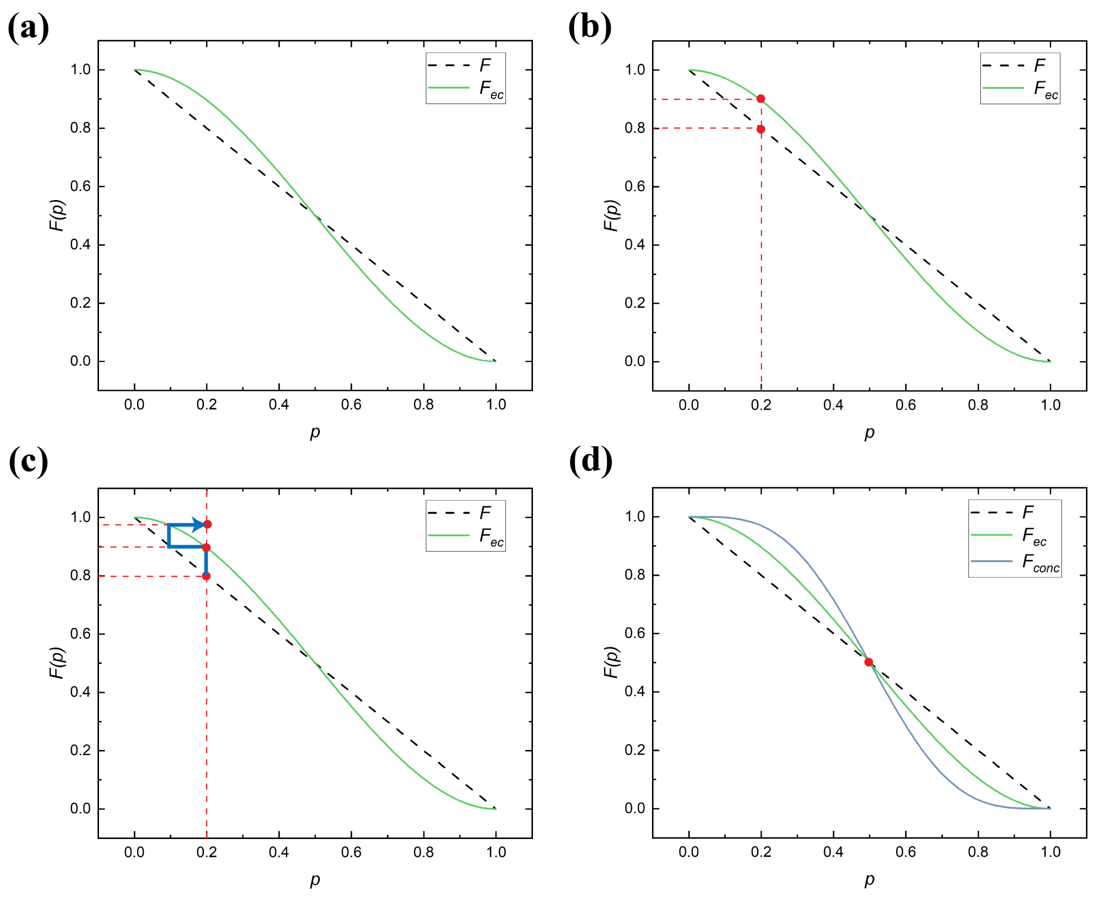


Figure 2.6: (a) The black dashed line F represents the physical fidelity without error correction code protection, while the light green line F_{ec} represents the fidelity of a logical qubit protected by one layer of error correction. (b) As seen, at a lower physical error rate p, $F_{ec} > F = 1 - p$. (c) When using the same error-correcting code for one round of code concatenation, the actual error rate of the physical qubits is p (shown in the diagram as $p = 0.2$). Then, when these physical qubits are encoded into "logical" qubits, these "logical" qubits have a higher fidelity (blue arrow pointing upwards). When using these "logical" qubits to encode our final logical qubit, the intermediate "logical" qubits have an effectively higher fidelity (blue arrow bending to the left), thus the actual "physical error rate" that the final logical qubit "feels" is in fact lower (blue arrow bends left then up again intersecting F_{ec}), resulting in a higher fidelity for the final logical qubit after one round of code concatenation (blue arrow bends right intersecting the original p dashed line). (d) By performing the operation described above for each p, we obtain the fidelity curve for the concatenated code (dark green), and these curves intersect at a point p_{th}. When p is below p_{th}, the concatenated code achieves better fidelity, otherwise, it results in worse fidelity.

From fig d, it can be seen that the physical fidelity curve F, the fidelity of a single error correction code F_{ec}, and the fidelity of the concatenated code F_{conc} all pass through a common intersection point p_{th}. At this critical point, the contribution of the concatenated code to fidelity is starkly different on either side. When p is sufficiently small, less than p_{th}, the concatenated code can provide better fidelity. However, when p exceeds p_{th}, the concatenated code results in fidelity that is even worse than not using error correction at all. This critical point is known as the threshold [21] of the quantum error correction code.

Lemma 2.2.2. ***The threshold theorem**: If a quantum computer has a physical error rate below a specific threshold, it can use quantum error correction schemes to reduce the logical error rate to extremely low levels.*

The threshold theorem is a crucial guarantee for achieving fault-tolerant quantum computing through quantum error correction methods, and code concatenation is one method to achieve this threshold (among others). Additionally, as can be seen from figure2.6.d, for "tree-like" concatenated codes using the same type of error correction code, the threshold is actually the fixed point of the original error correction code's logical error rate $1 - F_{ec}(p)$.

2.2.6. A Concatenated Code: Shor 9-Qubit Code

The quantum repetition code described earlier can correct only one type of error, either X or Z errors. The Shor 9-qubit code [48, 49] builds on the 3-qubit repetition code by performing code concatenation, thereby gaining the ability to detect and correct both X and Z errors simultaneously. The state preparation for the Shor 9-qubit code involves concatenating X-type and Z-type repetition codes. By performing such concatenation, the Shor 9-qubit code leverages the error-correcting capabilities of both types of repetition codes for different errors. The encoding circuit is shown in the figure 2.7.

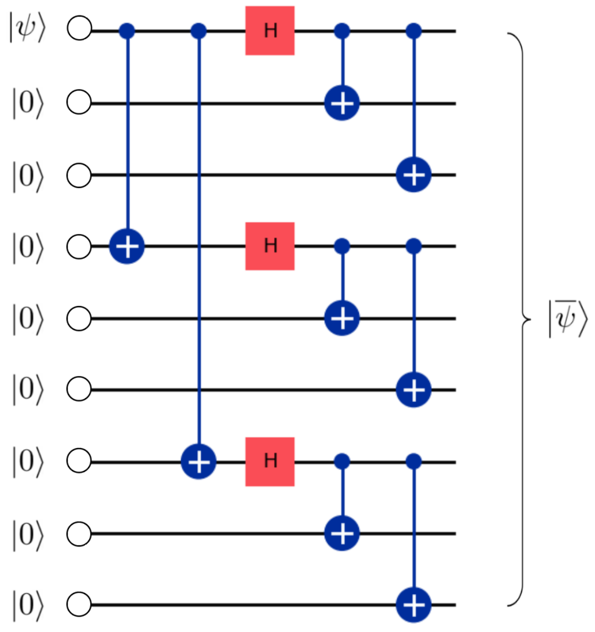


Figure 2.7: Encoding circuit for the Shor 9-qubit code, where the first qubit is in the $|\psi\rangle$ state and the rest are in the $|0\rangle$ state. Initially, the first, fourth, and seventh qubits are encoded into a 3-qubit repetition code using CNOT gates. Then, these three qubits pass through Hadamard gates before undergoing another round of quantum repetition code encoding. This process results in the Shor 9-qubit code, which can detect and correct both X and Z type errors.

2.2.7. Stabilizer Codes and Hashing Bound

Similar to classical linear stabilizer codes, a quantum stabilizer code [49] $[[n, k, d]]$ encodes k logical qubits into n physical qubits, with a rate of k/n, and requires $n-k$ auxiliary qubits to store the information of the stabilizers. Its stabilizer $\mathcal{S}$ is an Abelian subgroup of the n-fold Pauli group Π^n. The stabilizer $\mathcal{S}$ does not include the operator $-I^{\otimes n}$.

Mathematical Structure of Stabilizer Codes

For a stabilizer code , the system comprised of n qubits is a Hilbert space $H = (\mathbb{C}^2)^{\otimes n}/U(1)$, where its corresponding Pauli operators are the direct products of n single Pauli operators $\hat{O} = \prod_{i=1}^{n} \otimes \hat{O}_i$, where $\hat{O}_i \in \{\hat{I}, \hat{X}, \hat{Y}, \hat{Z}\}$.

All Pauli operators of length n can form a Pauli group $\mathcal{P}_n$, which has the following special subgroups:

Stabilizer Group $\mathcal{S}$ An Abelian subgroup, generated by linearly independent generators $S_1, \ldots, S_{n-k}$ with $-I \notin \mathcal{S}.\mathcal{S} = \langle S_1, S_2, \ldots, S_{n-k} \rangle$.

Centralizer of Stabilizer Group The centralizer of stabilizer group, generated by linearly independent generators that commute with $\mathcal{S}$. $\mathcal{C}(\mathcal{S}) = \langle S_1, S_2, \ldots, S_{n-k}, X_1, Z_1, \ldots, X_k, Z_k \rangle$.

The structures of these subgroups is as follows in Fig.2.8.

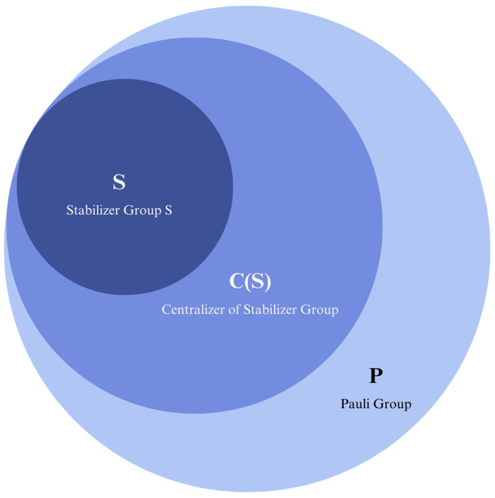


Figure 2.8: The structure of the Pauli group: At the center is the stabilizer group S, which is an Abelian subgroup of the Pauli group. $C(S)$ is the centralizer of the stabilizer group, and the outermost layer represents the entire Pauli group P.

Distance of a Stabilizer Code

The weight of a Pauli operator string is the number of non-identity (non-I) operators in the string.

For a logical operator P of a stabilizer code, its operation in the logical space under the action of the stabilizer S is equivalent, because $PS|\psi\rangle = SP|\psi\rangle$. The distance of a stabilizer code is defined as the weight of the smallest weight operator that can cause a change in the logical state starting from the code space [38]. Therefore, for a stabilizer code, its distance is:

$$d = \min_{P \in C(S)S} |P|. \tag{2.31}$$

Hashing Bound

For a Pauli channel:

$$\rho \mapsto p_I \rho + p_X X \rho X + p_Y Y \rho Y + p_Z Z \rho Z, \tag{2.32}$$

where $\mathbf{p} = (p_I, p_X, p_Y, p_Z)$ is a set of probabilities.

The **Hashing Bound Theorem** [53] states that for a Pauli channel, there exists a stabilizer code that can achieve the following rate:

$$R = 1 - H(\mathbf{p}), \tag{2.33}$$

where $H(\mathbf{p})$ is the Shannon entropy of the probability distribution $\mathbf{p} = (p_I, p_X, p_Y, p_Z)$.

2.2.8. From Classical to Quantum: Calderbank-Shor-Steane (CSS) Code

The Calderbank-Shor-Steane (CSS) Code is a special type of stabilizer code, constructed from two classical linear stabilizer codes C_1 $[n, k_1]$ and C_2 $[n, k_2]$, with the following requirements:

1. $C_2 \subseteq C_1$
2. $k_2 < k_1$
3. If C_1 and $C_2^\intercal$ can both correct errors with a maximum weight of t, then the CSS code constructed from them will be an $[[n, k_1 - k_2]]$ code capable of correcting up to one error with a weight of t.

Definition of CSS Codeword

Let N represent the total number of possible codewords for the final CSS code. Since the final CSS code encodes $k_1 - k_2$ logical qubits, the number of codewords allowed in the code space is $N = 2^{k1-k2}$. Next, select codewords $x_0, \ldots, x_{2^N-1} \in C_1$, and ensure:

$$x_i + x_j \notin C_2 \tag{2.34}$$

Where $i \neq j$. Since C_1/C_2 is a $k_1 - k_2$ dimensional space, a suitable x_i can always be found for each element. [37]

When the above conditions are met, one can use C_1 and $C_2^\intercal$ to construct a CSS code, with the codewords defined as follows:

$$x \in C_1 : |x + C_2\rangle = \frac{1}{\sqrt{2^{k_2}}} \sum_{y \in C_2} |x + y\rangle \tag{2.35}$$

The $+$ represents addition modulo 2. This CSS code can be represented by the set of its code space states: $\{|x + C_2\rangle \mid x \in C_1\}$.

These codewords are mutually orthogonal, i.e., $\langle x_i + C_2 | x_j + C_2\rangle = 0$ for $i \neq j$, which is ensured by Eq.2.34.

Stabilizer Matrix of the CSS Code

The stabilizer matrix of the CSS code is composed of the two classical parity-check matrices of C_1 and C_2:

$$H = \begin{bmatrix} H_{C_2^\perp} & 0 \\ 0 & H_{C_1} \end{bmatrix} \tag{2.36}$$

It can be observed that the stabilizers of the CSS code contain either only X operators or only Z operators.

2.3. Tensor Networks

Tensor networks [54, 55], in short, are networks composed of smaller tensors used to represent larger tensors. This representation method helps reduce the number of parameters needed to store the tensor and facilitates a more intuitive understanding of the tensor's structural characteristics.

2.3.1. Tensor Networks: A Method for Representing Large Tensors

In many scenarios, including quantum many-body systems and high-dimensional data analysis., tensors are powerful tools for representing linear mathematical structures. However, for a tensor T with n indices and a bound dimension of χ, the total number of elements in the tensor is χ^n. Clearly, as the number of indices increases, the memory cost to store a tensor grows exponentially. In practical applications, the number of indices required for tensors can be so large that storing such a tensor on a computer becomes impractical.

The good news is that if the system being dealt with has a specific structure, a large tensor can be represented and stored as a network of several smaller tensors. This can significantly reduce memory overhead, making it feasible to use tensors for many practical problems.

For a tensor T with N indices and a bound dimension of χ, if it has a specific mathematical structure and can be decomposed into k smaller tensors T_{ele} each with no more than n_{max} indices (where $n_{\text{max}} < N$), then the memory cost can be reduced from χ^N to at most $k \times \chi^{n_{\text{max}}}$. When $k(N)$, the memory overhead for processing this system is exponentially reduced: $\lim_{N\to\infty} \frac{k\times\chi^{n_{\text{max}}}}{\chi^N} = 0$. Thus, when a large tensor has a certain structure and can be represented as a tensor network, processing it becomes memory efficient. In fact, many real-world problems exhibit similar characteristics, where large-scale issues can be decomposed into structures made up of several smaller units.

For example, tensor networks are particularly useful in condensed matter physics. This is because such systems often involve entities such as spins or other more complex quantum units that are coupled together, forming systems where researchers typically aim to study macroscopic properties (large tensors) which possess specific structures (can be decomposed into tensor networks). With the aid of tensor networks, these complex condensed matter physics issues can be efficiently solved. For instance, Matrix Product States (MPS) can be used to describe one-dimensional quantum many-body systems, proving highly effective for computations of ground states and low-energy states. By appropriately choosing the dimensions of the matrices, one can effectively control the complexity and precision of the calculations. For higher-dimensional quantum many-body systems, Tensor Product States (TPS) also serve as a powerful descriptive tool, capturing complex correlations in higher-dimensional systems and suitable for addressing issues such as quantum phase transitions.

Tensor networks are also very important in handling holographic quantum error-correcting codes, as will be discussed next.

2.3.2. Quantum States as Tensors

For a pure quantum state, we can always express the quantum state in the following form based on its quantum numbers [53]:

$$|\Psi\rangle = \sum_{i,j,k...} c_{i,j,k...}|i,j,k,...\rangle \tag{2.37}$$

We notice that the coefficient $c_{i,j,k...}$ in front of each quantum state $|i,j,k...\rangle$ can be regarded as an element of a tensor. Collectively, these coefficients form a tensor T that can describe the quantum state [56–58].

$$T = \{c_{i,j,k...}\}_{\chi_i\times\chi_j\times\chi_k\times...} \tag{2.38}$$

2.3.3. State Channel Duality: Quantum Mappings as Tensors

In fact, according to the Choi–Jamiołkowski isomorphism [19, 53], there is a correspondence between quantum states and quantum channels, also known as state-channel duality. Therefore, a quantum channel (mapping) can also be represented as a tensor [53]. For a quantum map M:

$$\hat{M} = \sum_{i_1,i_2,...,i_m,j_1,j_2,...,j_n} c_{i_1,i_2,...,i_m,j_1,j_2,...,j_n}|i_1,i_2,...,i_m\rangle\langle j_1,j_2,...,j_n| \tag{2.39}$$

We can see that the coefficients $c_{i_1,i_2,...,i_m,j_1,j_2,...,j_n}$ of this mapping also form a tensor T, thus we can represent this mapping as a tensor as well.

2.3.4. Graphical Representation of Tensor Networks

For the graphical representation of a tensor network, we don't actually need to depict every element of the tensor. As a simplified representation, we focus more on the number of indices of the tensor. Assuming a tensor T has n indices, we represent the tensor T graphically as follows:

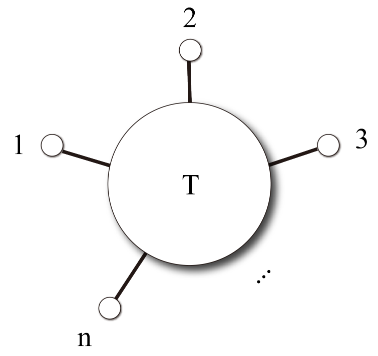


Figure 2.9: A graphical representation of a tensor T with n indices, where the white circle represents the tensor itself. The lines extending outward from the tensor represent indices, and these outward-extending lines are also known as tensor legs.

In the context of graphical representation of tensors, there are some terms to note: the white circle in the figure represents the tensor itself, and the lines extending outward represent the tensor's indices. These lines are also referred to as tensor legs.

When we attempt to connect two tensors, we are actually performing a tensor leg contraction operation [59]. In algebraic terms, leg contraction involves taking the trace over the indices that are connected between the two tensors. For instance, if we perform a contraction between the first index of the first tensor and the second index of the second tensor, algebraically, we have performed the following operation, where it's required that the bound dimensions χ corresponding to both indices are the same:

$$T_{contracted} = \sum_{i=1}^{\chi} T_{ijk\ldots} T_{lmn\ldots} \delta_{il} \tag{2.40}$$

For example, consider two tensors, each with 5 legs, where one leg from each tensor is connected to the other. The graph showing their contraction is as follows:

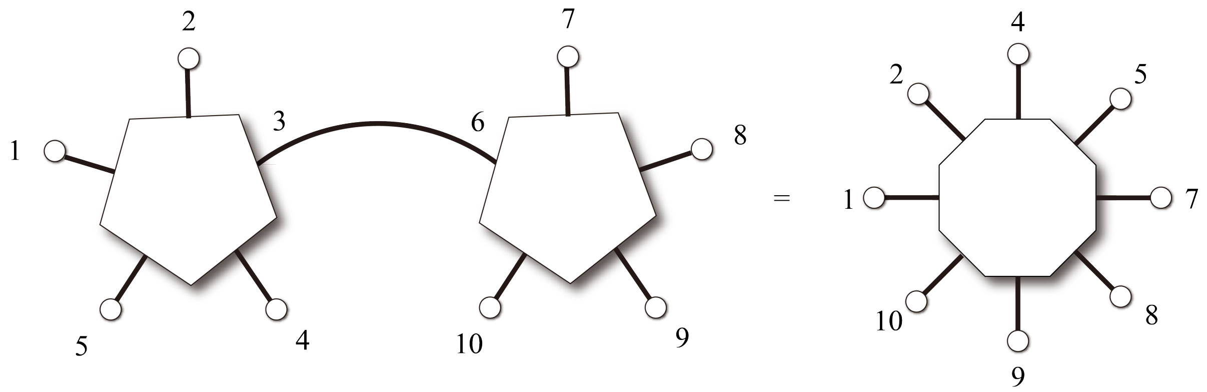


Figure 2.10: The diagram shows two tensors, each with 5 legs, undergoing contraction. The leg 3 of the tensor on the left contracts with leg 6 of the tensor on the right, resulting in a tensor with 8 legs.

2.3.5. Quantum Error Correction Codes as Tensors

As a specific type of quantum mapping, a $[[n, k, d]]$ quantum error-correcting code maps k logical qubits onto n physical qubits [5]. Therefore, our quantum error-correcting code is actually a tensor with $n + k$ legs [5], and since we are discussing a qubit system, the bound dimension is 2.

As illustrated in Fig.2.11, the orange indices represent logical qubits, and the white indices represent physical qubits. The tensor in the diagram represents a quantum error-correcting code that encodes k logical qubits into n physical qubits:

$$V = \sum_{ij} V_{i_1^P \ldots i_n^P, j_1^L \ldots j_k^L} |i_1^P \ldots i_n^P\rangle\langle j_1^L \ldots j_k^L| \tag{2.41}$$

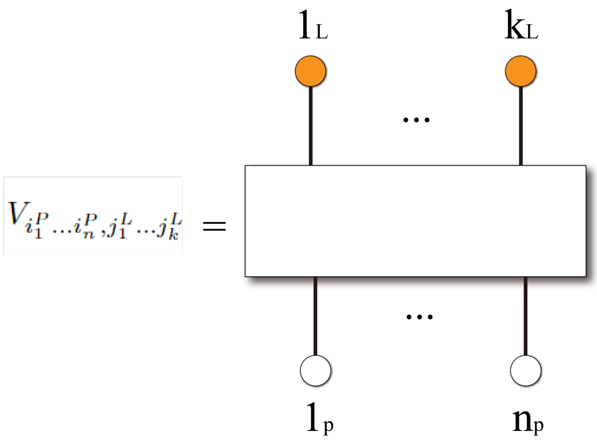


Figure 2.11: Graphical tensor representation of a [[n, k, d]] quantum error correcting code. The mapping $V_{i_1^P \ldots i_n^P, j_1^L \ldots j_k^L}$ represents a linear transformation from k logical qubits to n physical qubits, and can be represented by a tensor. The legs where the orange dots are located represent the logical qubits, and the legs with the white dots represent the physical qubits. This tensor represents the transformation from the logical code space to the physical code space $H_d^{\otimes k} \to H_d^{\otimes n}$.

2.3.6. Tensor Network Representation of Code Concatenation

To represent code concatenation in the language of tensor networks, we first select a $[[n, 1, d]]$ quantum error-correcting code, as shown in the following figure:

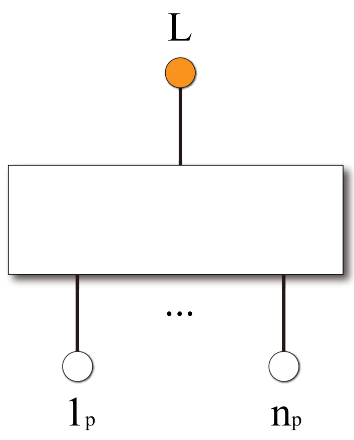


Figure 2.12: Graphical Tensor Representation of a [[n, 1, d]] Quantum Error Correcting Code

We can use this $[[n, 1, d]]$ error-correcting code as a ”building block” by connecting the physical qubit legs of the original $[[n, 1, d]]$ error-correcting code tensor to the logical qubit leg of additional $[[n, 1, d]]$ error-correcting codes. This process further encodes the original physical qubits into logical qubits through error correction, resulting in an overall larger error-correcting code.

Looking back at the previously mentioned Shor 9-qubit code, it is actually derived from a 3-qubit repetition code through concatenation:

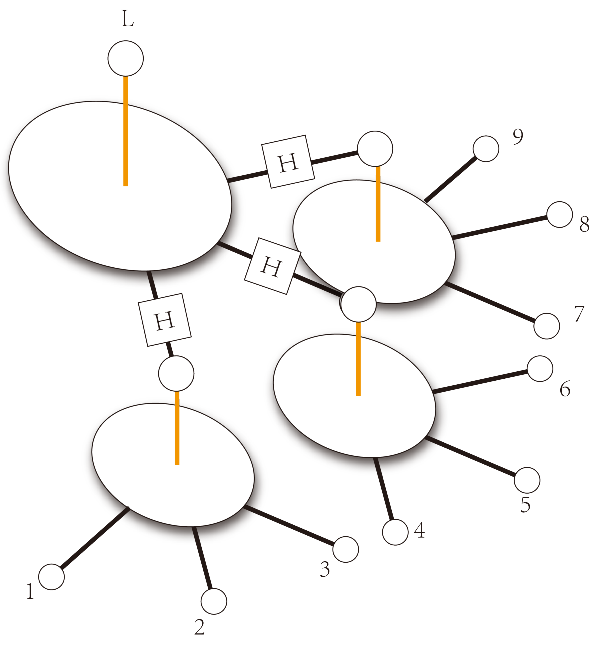


Figure 2.13: The tensor network representation of the Shor 9-qubit code is obtained by taking a 3-qubit quantum repetition code, applying Hadamard gates to its three physical legs, and then concatenating it with three additional 3-qubit quantum repetition codes.

2.4. AdS/CFT Correspondence and Holography

The AdS/CFT correspondence, also known as holographic duality or gauge/gravity correspondence, links quantum field theory (QFT) and gravity [60, 61]. Specifically, it associates the quantum behavior of strongly correlated many-body systems with the classical gravitational dynamics in an extra dimension [22, 62, 63]. It also plays a key role in theoretical debates about black holes. Below, we will start by introducing the theoretical challenges of extremal black hole, D-branes, and the properties of Anti-de Sitter space, leading into an introduction of the AdS/CFT Correspondence, and its connection to holography.

2.4.1. Thermodynamics of Black Holes

The work of Stephen W. Hawking and Jacob D. Bekenstein [64, 65] revealed that black holes, as thermodynamic objects, possess temperature and entropy as follows:

$$T_H = \frac{\hbar c^3}{8\pi k_B G M}, \qquad S_{BH} = \frac{4GM^2}{\hbar c} = \frac{c^3 A_{\text{hor}}}{4\hbar G}, \tag{2.42}$$

It can be observed that the entropy of a black hole is proportional to the area of its event horizon, indicating that the information of the black hole is encoded on its ”surface” (i.e., the event horizon). This is consistent with the holographic principle.

2.4.2. Challenges of Extremal Black Hole and D-branes

Ashoke Sen considered a string in a highly excited state that winds many times around a compact dimension. Such a string would be heavier than a normal string because energy is required to wrap the string around this compact direction. Consequently, a string with a winding number of 2 in the compact direction will be heavier than one with a winding number of 1. As the winding number of the string is increased to a very large value, its mass also becomes very large. However, since it is confined to a compact space, it can only be a black hole. Moreover, since the winding of the string generates charge, such black holes are referred to as extremal black holes [66], which are a type of charged black hole.

Ashoke Sen used the Einstein field equations to calculate the area of the event horizon for these extremal black holes. Surprisingly, the result showed that the area of the event horizon vanishes to zero, which directly contradicts the holographic principle. He suggested that this might be because the field equations do not take into account quantum fluctuations, which would expand such a black hole and create an extended event horizon. The entropy is proportional to the area of this extended event horizon.

Jin Dai, Leigh, Joseph Polchinski [67], and Hořava [68] introduced the concept of D-branes, where open strings can end with Dirichlet boundary conditions. Thus, the 'D' in D-branes stands for Dirichlet. The description of D-branes includes additional dimensions, which can be either compactified or non-compactified. Using D-branes, Andrew Strominger and Cumrun Vafa [69] combined strings and D-branes to construct an extremal black hole with a sizable and definite classical event horizon, allowing string theory to provide a total amount of information for extremal black holes that agrees with the Hawking formula Eq.2.42. This highlights the crucial role of D-branes in string theory.

2.4.3. A Container for Black Holes: Anti-de Sitter Space

A d-dimensional anti-de Sitter space can be viewed as a submanifold in d-dimensional Minkowski spacetime, which satisfies the following equation:

$$\sum_{i=1}^{d-1}(x_i)^2 - (x_d)^2 - (x_0)^2 = -R^2 \tag{2.43}$$

Where R is a non-zero constant, known as the radius of the spacetime. This metric indicates that the spatial curvature of Anti-de Sitter space is negative. As a curved spacetime, Anti-de Sitter space generates a gravitational pull towards its center, which causes any object near its boundary, including the event horizons of black holes, to experience a gravitational force pulling them away from the boundary and back towards the center. The ”repulsive force” between such objects and the boundary of Anti-de Sitter space is strong enough that the event horizons of black holes cannot extend to the boundary of Anti-de Sitter space. This makes it possible to study a non-evaporating black hole, turning Anti-de Sitter space into a ”container” that can ”hold” black holes.

Understanding the geometric properties of Anti-de Sitter Space can be helpful by considering a 2+1-dimensional Anti-de Sitter Space, as shown in the figure 2.14. The 2+1-dimensional Anti-de Sitter Space appears cylindrical in the figure 2.14, with the vertical direction representing time. Slicing through Anti-de Sitter Space at a specific time horizontally, as shown in the figure, yields a cross-section where objects at the center appear larger, while those closer to the boundary appear smaller. Due to the curvature of this spacetime, objects closer to the boundary of Anti-de Sitter Space experience stronger centripetal gravitational forces. In this space, no objects can interact with its boundary, which ensures the validity of the holographic principle in Anti-de Sitter Space, as it ensures that no objects ”pass through” the boundary.

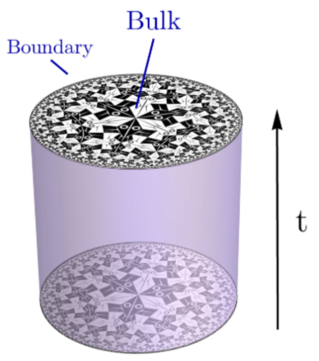


Figure 2.14: Illustration of 2+1-dimensional Anti-de Sitter Space, where the horizontal circles represent the two spatial dimensions, and the vertical dimension represents time. Slicing through Anti-de Sitter Space at a specific time point yields a two-dimensional space with negative curvature. The ”cylindrical” boundary of this Anti-de Sitter Space corresponds to a conformal field theory. Figure from [70]

Additionally, AdS space corresponds to an isometry group, which means that the metric of the manifold is consistent with the metric of the embedding space, implying that AdS space possesses unique geometric properties.

2.4.4. AdS/CFT Correspondence Conjecture

Juan Maldacena discussed the properties of open strings ending on the D3 brane, which can exist independently or as part of a stack of D3 branes. He explored the interaction rules of open strings ending on the D3 brane, which he found to be identical to the rules governing gluons in QCD. His astonishing discovery was that the two different-dimensional descriptions of D branes, as outlined by Maldacena, are equivalent. One description is a 3+1 dimensional QCD that does not include gravity. The dual description involves a 4+1 dimensional Anti-de Sitter Space. This is called AdS/CFT Correspondence [22, 71, 72].

Initially, it seems counterintuitive: how can an Anti-de Sitter Space be equivalent to a lower-dimensional CFT? This can be explained using the concept of the holographic principle. The extra dimension in Anti-de Sitter Space (the fourth spatial dimension in Juan Maldacena's example) does not correspond to movement within the CFT, but rather to a "scaling" operation. Thus, the information in Anti-de Sitter Space can be equivalently described within the CFT. This is the holographic principle.

2.5. Holographic Quantum Error Correction Codes

In recent years, there has been a gradual connection established between quantum information and the AdS/CFT correspondence. Surprisingly, these two areas have been found to mutually benefit each other. Discussing the AdS/CFT correspondence within the context of quantum information has led to new insights into it. Similarly, the AdS/CFT correspondence holds potential in aiding the construction of effective quantum error-correcting codes.

2.5.1. Ryu-Takayanagi (RT) formula

A well-known formula that reveals the connection between the AdS/CFT correspondence and quantum information, the Ryu-Takayanagi formula [73], establishes a representation for the entanglement entropy of a subsystem A on a holographic CFT within the AdS space. It indicates that S_A, the entanglement entropy of subsystem A, is proportional to the area of a $d-1$-dimensional minimal surface γ_A homologous to A:

$$S_A = \frac{|\gamma A|}{4G}, \tag{2.44}$$

$|\gamma_A|$ is the area of γ_A, and G is the gravitational constant in the AdS space-time. The Ryu-Takayanagi formula is an extension of the Bekenstein-Hawking entropy [64, 74] formula in the context of quantum information.

2.5.2. Holographic Codes

The AdS/CFT correspondence in quantum information reveals a mapping of quantum information between the two dual spaces [75–78], where information from the bulk (AdS) is mapped to the boundary (CFT) and vice versa. More specifically, under the holographic principle, the information in an $n+1$-dimensional AdS space is holographically projected onto an $(n-1)+1$-dimensional CFT. Here, the motions in the $n-1$ dimensions are directly corresponded, while the motion in the nth spatial dimension of the AdS space is represented as scaling in the CFT. Due to the conformal invariance of the CFT, some symmetries within the AdS space are also preserved in the CFT. Consequently, Noether's theorem [79] ensures the correspondence of conserved quantities and physical equivalence between the two dual descriptions. Thus, information is equivalent under both descriptions; starting from AdS space, information on the CFT can be derived, and starting from the CFT, information in AdS space can be "reconstructed." Therefore, inspired by the AdS/CFT correspondence, tensor networks can be used to construct toy models of the AdS/CFT duality, creating quantum error-correcting codes that mimic the properties of the AdS/CFT duality. This type of code enables the reconstruction of quantum information in the bulk area through the quantum information on the boundary, making such holographic error-correcting codes a promising candidate for quantum error correction.

Causal Wedge in Holography

Causality in flat spacetime can be described using light cones, while for more general spaces, the causal wedge [80–82] is a powerful tool for describing causal relationships. They are not the same concept; the shapes and extents of causal wedges and light cones may differ. When discussing causality in the context of the AdS/CFT correspondence, we need to use causal wedges for description.

The concept of a causal wedge in holography can be elucidated with an example:

On a Poincaré disk with negative curvature, there exists a field operator $\psi(x)$ in the bulk region. The bulk region is a timeslice of AdS spacetime at a specific point in time, characterized by negative spatial curvature.. On the CFT boundary, there are two operators O_A and O_B, with their corresponding causal wedges being W_A and W_B. If $\psi(x)$ is located within both W_A and W_B, this means that the field operator $\psi(x)$ can be reconstructed by the boundary operators O_A and O_B. This is because the causality regarding the CFT boundary operators O_A and O_B only exists within their causal wedges. Since the bulk region of interest is a slice of AdS spacetime at a specific point in time, the slices of the causal wedges are the regions W_A and W_B as shown in the figure 2.15. However, this does not necessarily mean that the intersection of the boundary operators $O_A \cap O_B$ can always reconstruct $\psi(x)$, because $\psi(x)$ may not lie within $W_A \cap W_B$.

In the context of quantum information, such causal wedges are also referred to as entanglement wedges. The core idea is that to fully reconstruct a field operator in the bulk, the entanglement wedge of the CFT boundary operators used for the reconstruction must include this bulk field operator [83].

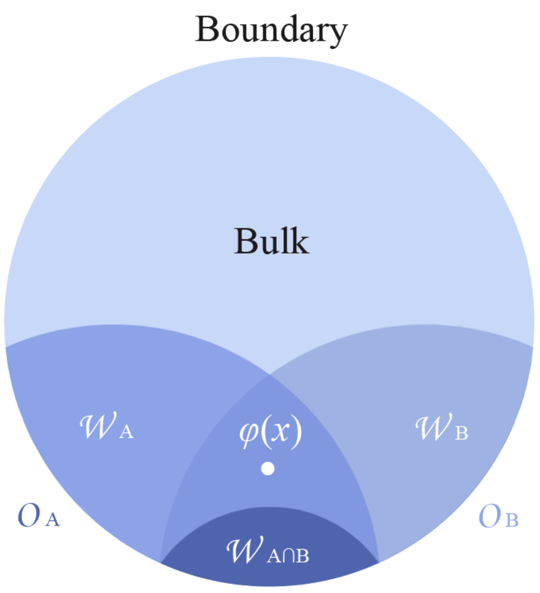


Figure 2.15: An example of a causal wedge: A field operator $\psi(x)$ located in the bulk region can be encompassed by the causal wedges of different CFT boundary operators O_A and O_B. Consequently, the information of $\psi(x)$ can be independently reconstructed by O_A and O_B, meaning that the information of $\psi(x)$ is redundantly stored on the CFT boundary. It's important to note, however, that the causal wedges of the boundary operators $O_{A\cap B}$ do not necessarily include $\psi(x)$.

Constructing Holographic Codes Using Tensor Networks

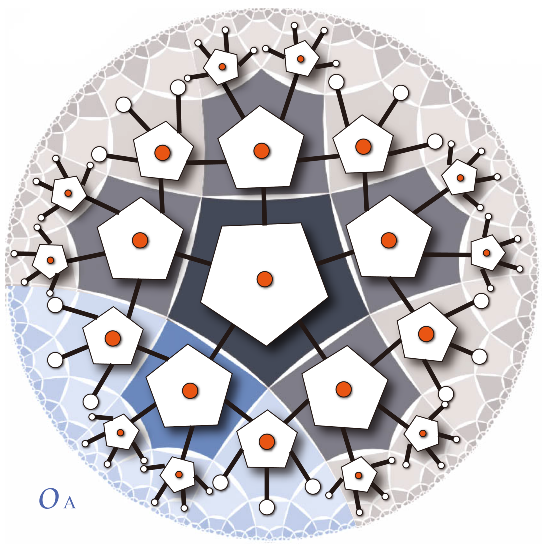


Figure 2.16: An example of constructing a discrete holographic code using tensor networks involves using the $[[5,1,3]]$ code as a ”building block” tensor, holographically stacking with a $[p,q]=[5,4]$ configuration on a negatively curved Poincaré disk, and connecting the tensor legs of adjacent tensors. The orange circles represent the logical qubits of each ”building block” tensor, located within the bulk space, and also serve as the logical qubits of the entire holographic network code. Their information is redundantly stored on the boundary, represented by the white circles on the boundary as physical qubits.[3, 70]

For a field operator $\psi(x)$ located near the center of the bulk, its reconstruction can be achieved through multiple different wedges. This means that the removal of a boundary operator, when its corresponding wedge does not penetrate deeply into the central region of the bulk, allows the information of the field operator $\psi(x)$ to be obtained through other CFT boundary operators that have not been removed. Therefore, the information of the field operator $\psi(x)$ in the bulk is redundantly stored on the CFT boundary, which aligns with the properties of quantum error-correcting codes. Quantum error-correcting codes based on this principle are known as holographic quantum error-correcting codes.

To discuss holographic error correction in discrete spaces, tensor networks serve as a useful tool for constructing toy models of holographic quantum error-correcting codes. The network structure and the properties of the tensors themselves (such as isometry, which will be discussed in subsequent sections) can to some extent mimic the behavior of entanglement wedges in continuous holographic error correction and discretely mimic fractal characteristics on the boundary.

To construct such discrete holographic quantum error-correcting codes, one starts by choosing a Poincaré disk with negative curvature. On this disk, tensors that preserve inner product transformations (which will be elaborated in the later HaPPY code section) are holographically stacked to form a holographic tensor network [70]. This network represents a mapping from the logical qubits in the bulk region to the physical qubits on the CFT boundary, thereby establishing a tensor network-based discrete holographic quantum error-correcting code. An example is illustrated in Fig. 2.16.

The Thresholds for Holographic Codes

Thresholds are a crucial metric for any error-correcting code. Ref.[84] discusses the threshold characteristics in holographic quantum error-correcting codes. They demonstrated that holographic CFTs possess algebraic thresholds, related to the confinement-deconfinement phase transition. In the continuous limit ($N\to\infty$), the logical error rate converges to a step function, indicating that, as holographic codes, AdS/CFT has an error threshold for thermal noise consistent with the deconfinement phase transition temperature.

2.5.3. Inflation Pattern: Vertex Inflation and Edge Inflation

Holographic codes have two different inflation patterns [70]: Vertex Inflation and Edge Inflation. Taking the $[p,q]=[5,4]$ tiling as an example:

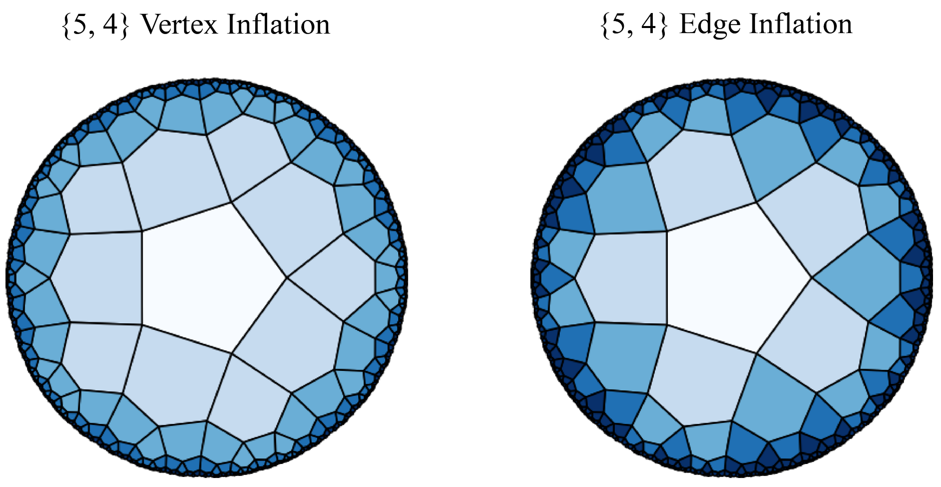


Figure 2.17: Vertex Inflation and Edge Inflation

The difference between them is as follows [85]: Vertex Inflation requires that when stacking the next layer of tensors, each tensor must share an edge with a tensor from the previous layer and also share edges with tensors within the same layer. This ensures that all vertices inside the stack are shared by $q = 4$ polygons. On the other hand, for Edge Inflation, the rule is that when adding a new layer of tensors, we only need to extend from an edge of a tensor from the previous layer and place a new tensor in the next layer, without ensuring that all vertices are shared by q polygons. The differences between these two inflation patterns are illustrated in the following diagram, where different colors represent tensors from different layers. Tiling patterns other than $\{5, 4\}$ are also allowed [86].

2.5.4. HaPPY Code

As one of the most typical codes in holographic codes, the HaPPY code [3] serves as an excellent example for understanding the properties of holographic codes. **Isometries and Perfect Tensors**

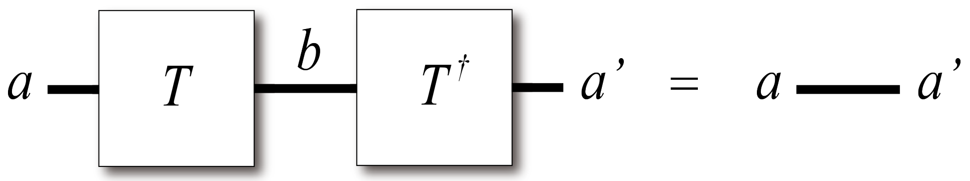


Figure 2.18: Tensor notation, here showing that T is an isometry

Definition of isometries: H_A and H_B are two Hilbert spaces, which can have different dimensions. An isometry from H_A to H_B is a linear map $T : H_A \to H_B$ that preserves the inner product from H_A to H_B.

For our construction use case of the holographic code, we assume that both H_A and H_B have finite dimensions. Naturally, we can conclude that an isometry from H_A to H_B exists if and only if dim$(A) <$ dim(B). If dim$(A) =$ dim(B), then the isometry T is a unitary transformation. Specifically, this means that T is an isometry from H_A to H_B, and $T^\dagger$ is an isometry from H_B to H_A.

Consider a general linear map T, which linearly maps states from H_A to H_B as

$$T : |a\rangle \to \sum_b |b\rangle T_{ba}. \tag{2.45}$$

If T is an isometry, it must preserve the inner product, hence we have:

$$\langle a'|a\rangle = \sum_{b',b} T^{\dagger}_{a'b'} T_{ba} \langle b'|b\rangle \tag{2.46}$$

Hence:

$$\sum_{b} T^{\dagger}_{a'b} T_{ba} = \delta_{a'a}. \tag{2.47}$$

From this, we can see that when T is an isometry from H_A to H_B, then $T^{\dagger}T$ is the identity operator on H_A.

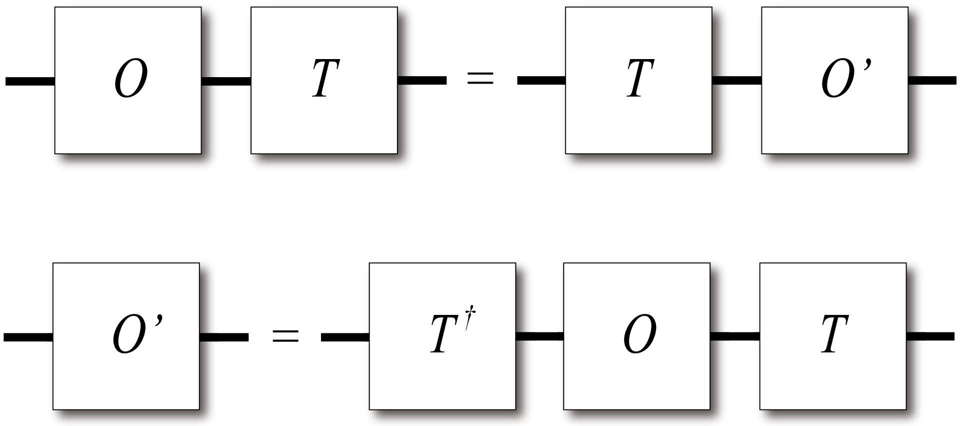


Figure 2.19: Operator Pushing

Furthermore, when there is an isometry T from H_A to H_B, and we consider the correspondence of operators between H_A and H_B, suppose there is an operator O in H_A. The operator O is mapped by the isometry T to an operator O' in H_B, then we have:

$$TO = TOT^{\dagger}T = (TOT^{\dagger})T = O'T \tag{2.48}$$

Hence:

$$O' = TOT^{\dagger} \tag{2.49}$$

Through this formula, we can determine the expression for O'. This process can be visually understood as pushing the operator O, originally in H_A, through T to H_B, resulting in O'.

Definition of Perfect Tensors: For a tensor $T_{a_1,a_2,\ldots,a_{2n}}$ with $2n$ indices, given any bipartition of these indices into a set A and its complement A^c, satisfying $|A| \leq |A^c|$, if T is proportional to an isometry tensor from A to A^c, then T is called a perfect tensor.

We can regard the perfect tensor T as a quantum pure state within the H_{2n} space, which can be written as:

$$|\psi\rangle = \sum_{a_1,a_2,\ldots,a_{2n}} T_{a_1 a_2 \ldots a_{2n}} |a_1 a_2 \ldots a_{2n}\rangle. \tag{2.50}$$

This quantum state, composed of $2n$ qubits, takes any subsystem of n qubits to be maximally entangled with the remaining n qubits. Such quantum states are known as absolutely maximally entangled (AME) states [87, 88]. Every perfect tensor defines an AME state, and conversely, every AME state defines a perfect tensor.

Construction of HaPPY Code

To construct the HaPPY code, we start from a $[[5,1,3]]$ perfect code, which perfectly maps 1 logical qubit onto 5 physical qubits, with a distance of 3. The stabilizer group of this perfect code has 4 generators, denoted as $S = \langle S_1, S_2, S_3, S_4 \rangle$, with the generators as follows:

$$\begin{aligned} S_1 &= X \otimes Z \otimes Z \otimes X \otimes I \\ S_2 &= I \otimes X \otimes Z \otimes Z \otimes X \\ S_3 &= X \otimes I \otimes X \otimes Z \otimes Z \\ S_4 &= Z \otimes X \otimes I \otimes X \otimes Z \end{aligned} \tag{2.51}$$

The logical operators $\bar{X}$ and $\bar{Z}$ of this perfect code are as follows:

$$\begin{aligned} \overline{X} &= X \otimes X \otimes X \otimes X \otimes X \\ \overline{Z} &= Z \otimes Z \otimes Z \otimes Z \otimes Z \end{aligned} \tag{2.52}$$

After determining the perfect tensor we use to construct the HaPPY code, we proceed to create the required topology for the HaPPY code. We start by placing a $[[5,1,3]]$ perfect code at the center of a Poincaré disk with negative curvature. Then, we stack tensors on this Poincaré disk using a $[p,q] = [5,4]$ tessellation method, as illustrated in Fig. 2.20:

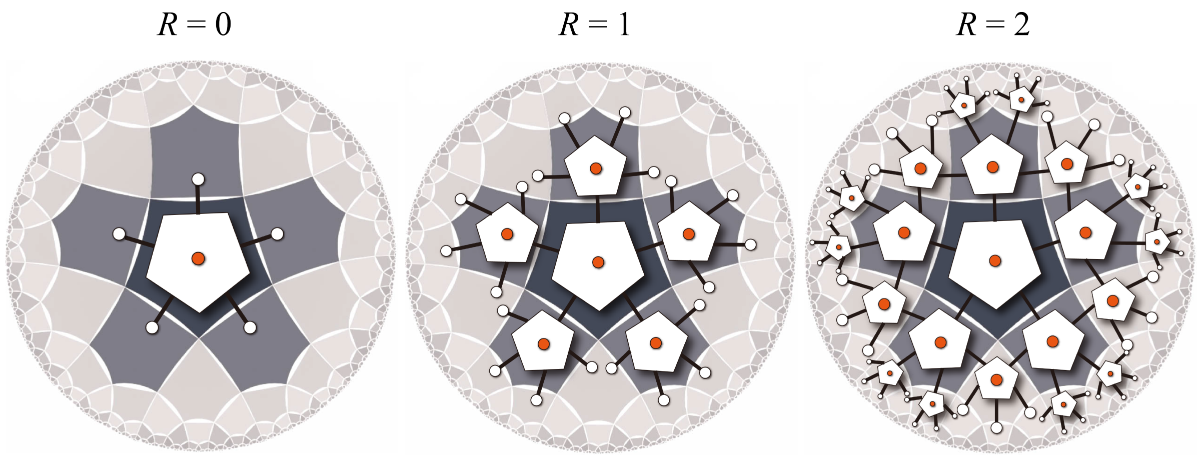


Figure 2.20: The growing process of the HaPPY code from $R = 0$ to $R = 2$, following the rules of edge inflation. Place the perfect code $[[5,1,3]]$ on the negatively curved Poincaré disk. This forms a HaPPY code with radius $R = 0$. Then, using the edge inflation mode, holographically stack a new layer of the $[[5,1,3]]$ perfect code on the radius $R = 0$ HaPPY code with $[p,q] = [5,4]$, resulting in the $R = 1$ HaPPY code. By continuing to holographically stack a layer of the $[[5,1,3]]$ perfect code on the $R = 1$ HaPPY code with $[p,q] = [5,4]$, the $R = 2$ HaPPY code can be obtained.

Rate of HaPPY Code

The HaPPY code can have different rates, where the rate of a $[[n,k,d]]$ code is defined as k/n. For instance, in the construction of the HaPPY code we previously discussed, we built a max rate HaPPY code because it encodes one logical qubit per tensor as input. In fact, having a logical qubit on every tensor is not necessary. We can change the code's rate by altering the encoding density of logical qubits in the tensor network. As shown in the following diagram, on the left is a HaPPY code that encodes logical qubits only at even R levels, also known as the Pentagon/Hexagon code. On the right is a HaPPY code that encodes a logical qubit only at the central tensor, also known as the One qubit code or zero rate HaPPY code. Because as R increases, the rate of the One qubit code tends toward zero.

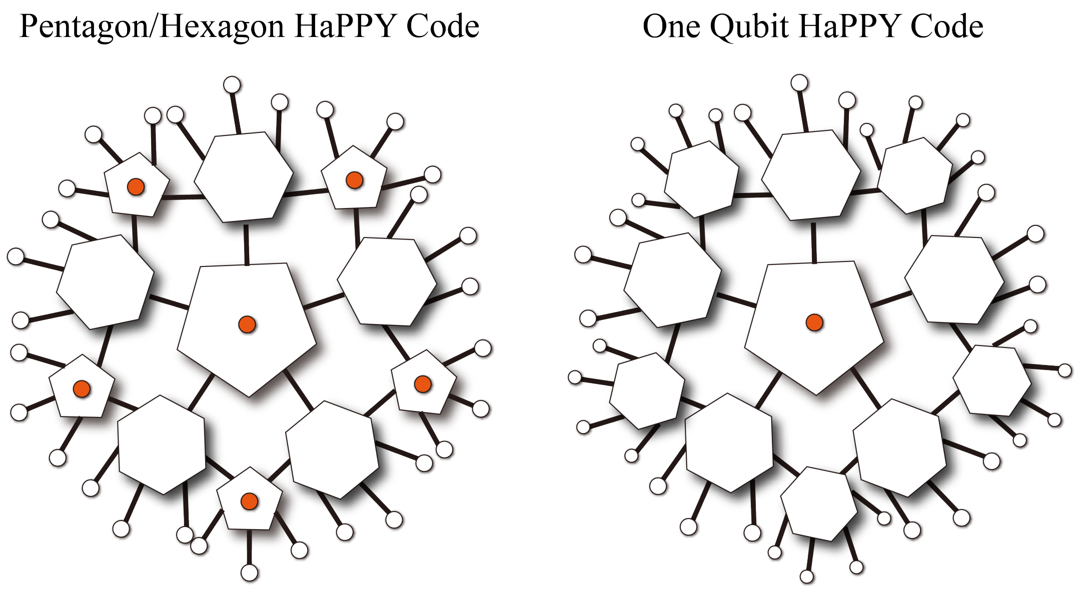


Figure 2.21: Tensor network representations of the Pentagon Hexagon HaPPY code and the zero rate HaPPY code.

2.5.5. Holographic Steane Code

Beyond the HaPPY code, there are many other excellent holographic codes, where a key step is the selection of tensors. Here, we introduce the holographic Steane code [89], which utilizes the Steane code [90] tensor. As a type of quantum error-correcting code, the Steane code has many outstanding features, giving us reason to expect that the holographic Steane code will possess desirable properties.

Block Perfect Tensors

When examining the tensor of the Steane code, we naturally hope that it could be a perfect tensor, similar to the $[[5,1,3]]$ tensor used in the HaPPY code, but this is not the case. However, the good news is that the tensor corresponding to the Steane code can form an isometry mapping under some limited bipartitions, which we refer to as block perfect tensors [89].

Recalling the concept of perfect tensors, for their index sequence $\overline{J} = \{j_1, j_2, \ldots, j_{2m}\}$, given a bipartition $\{A|\overline{A}\} = \Pi[\overline{J}]$, where Π is any permutation, as long as $|A| < |\overline{A}|$, it can form an isometry from H_A to $H_{\overline{A}}$. This requirement is quite stringent for a tensor T, and in reality, we can relax this requirement to obtain a broader class of tensors. These tensors can form an isometry under a limited set of permutations, which we refer to as block perfect tensors.

Construction of Holographic Steane Code

To construct the holographic Steane code, let's first look at the seed tensor we use: the Steane tensor. Below are the stabilizer group generators and logical operators of the Steane tensor:

Index label:	1	2	3	4	5	6	L	7
S_1	X	X	I	I	I	X	I	X
S_2	I	X	X	X	I	I	I	X
S_3	I	I	I	X	X	X	I	X
S_4	Z	Z	I	I	I	Z	I	Z
S_5	I	Z	Z	Z	I	I	I	Z
S_6	I	I	I	Z	Z	Z	I	Z
S_x	X	X	X	X	X	X	X	X
S_z	Z	Z	Z	Z	Z	Z	Z	Z

(2.53)

The Steane tensor is not a perfect tensor because it cannot guarantee the formation of an isometry under all permutations. However, as long as the ”input” indices of the Steane tensor are sequentially adjacent, for example, 6, L, 7, with the remaining indices serving as ”output,” it can be verified that the Steane tensor will provide an isometry under these conditions.

By using the Steane tensor as the seed tensor and employing a $[p, q] = [7, 4]$ tessellation method on the Poincaré disk, holographically tessellating Steane tensors can form the holographic Steane code. As shown in the Fig. 2.22.

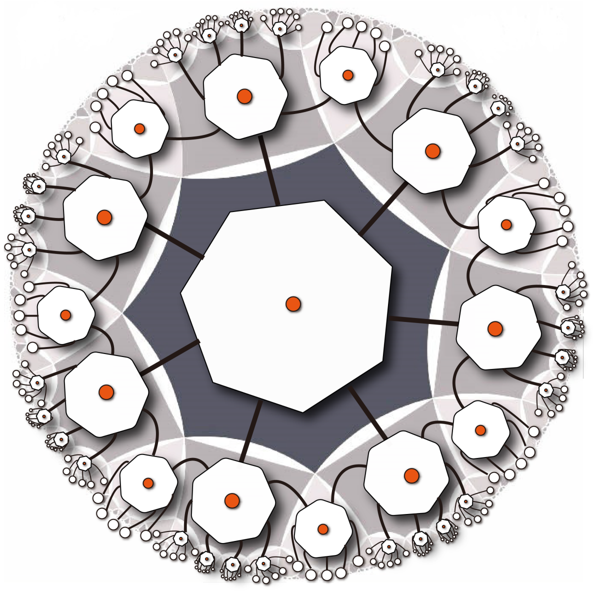


Figure 2.22: Max rate holographic Steane code, following the edge inflation rule.

2.5.6. Hyper-Invariant Tensor Network (HTN) Code

Indeed, while both the HaPPY code and the holographic Steane Code are quantum error-correcting codes proposed based on the AdS/CFT correspondence, they do not fully exhibit all the key properties of the AdS/CFT correspondence [4]. A significant feature of the AdS/CFT duality is the correspondence between the n-point correlation functions on the CFT and the geometry in the AdS space, such as the direct connection between the two-point correlation functions on the CFT boundary and the geodesics in the AdS space. A new class of holographic codes, which excellently reflect this property, is the Hyper-Invariant Tensor Network (HTN) Code [4, 91].

The HTN code is a type of holographic quantum error-correcting code based on Vertex Inflation. Its stacking method on the Poincaré disk can vary. We introduce a typical HTN code, the $[p, q] = [4, 5]$ HTN code, and explain how it is constructed.

Construction of $[p, q] = [4, 5]$ HTN code

The $[p, q] = [4, 5]$ HTN code uses a seed tensor that is block perfect. The generators of its stabilizer group and the logical operators are as follows:

Index label:	1	2	3	4	L
S_1	X	X	X	X	I
S_2	I	Z	I	Z	I
S_3	Z	I	Z	I	I
S_x	I	X	I	X	X
S_z	I	I	Z	Z	Z

(2.54)

Then, this seed tensor is holographically stacked on the Poincaré disk through vertex inflation, as shown in the following Fig.2.23.

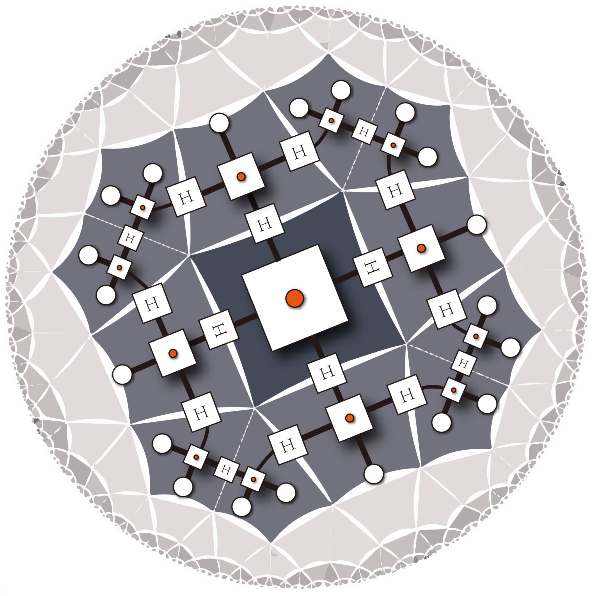


Figure 2.23: The $[4,5]$ HTN code at $L=1$ [92] , following the vertex inflation rule.

Rate of HTN Code

Unlike most other holographic codes, the HTN code has relatively strict requirements regarding the geometric properties of the holographic stacking [92]. Therefore, when discussing non-max rate HTN codes, we cannot simply treat the logical leg of the seed tensor as a normal leg and stack it onto the Poincaré disk as a polygon with one more edge than the original seed tensor. Instead, when handling non-max rate HTN codes, we first ”discard” the logical leg on some tensors. Originally, this tensor had gauge degrees of freedom, but since we ”discard” the logical index of this tensor, the gauge degree of freedom can be set to a logical operator. For example, if we choose the Z logical operator as the gauge we wish to fix, we are essentially expanding the original tensor's $\langle S1, S2, S3\rangle$ to $\langle S1, S2, S3, Z\rangle$. Thus, we transform the original $[[n,k]] = [[4,1]]$ error-correcting code into a $[[n,k]] = [[4,0]]$ mapping tensor.

By freely choosing the tensors on which to perform gauge fixing operations (also known as projecting onto an eigenstate of a certain logical operator), we can start with a max rate HTN code and freely reduce the rate of the HTN code to achieve our desired code. In particular, when we project all tensors except for the central tensor onto an eigenstate of one of their logical operators, we effectively construct a zero rate HTN code (One qubit HTN code). More details on this can be found in [92].

2.5.7. Holographic Reed-Muller Code

In the field of fault-tolerant quantum computing, the transversality of gate operations is a highly desirable property. For most codes, the transversality of Clifford operations is not uncommon. However, for typical non-Clifford operations, like the T gate, most codes do not have a transversal operation to implement a logical T gate and must resort to methods like magic state distillation to achieve logical T gate operations. The Reed Muller code [23, 93] allows for the transversal operation of the T gate, making it one of the codes we are interested in. We want to see if interesting properties emerge when using it as a seed tensor to construct a holographic error-correcting code [5].

Construction of Holographic Reed Muller Code

The stabilizer group generators and logical operators used by the Reed Muller code are as follows, which are key to constructing the seed tensor:

Index label:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z	I	Z
S_2	I	Z	Z	I	I	Z	Z	I	I	Z	Z	I	I	Z	Z
S_3	I	I	I	Z	Z	Z	Z	I	I	I	I	Z	Z	Z	Z
S_4	I	I	I	I	I	I	I	Z	Z	Z	Z	Z	Z	Z	Z
S_5	I	I	Z	I	I	I	Z	I	I	I	Z	I	I	I	Z
S_6	I	I	I	I	Z	I	Z	I	I	I	I	I	Z	I	Z
S_7	I	I	I	I	I	Z	Z	I	I	I	I	I	I	Z	Z
S_8	I	I	I	I	I	I	I	I	I	Z	Z	I	I	Z	Z
S_9	I	I	I	I	I	I	I	I	I	I	I	Z	Z	Z	Z
S_{10}	I	I	I	I	I	I	I	I	Z	I	Z	I	Z	I	Z
S_{11}	X	I	X	I	X	I	X	I	X	I	X	I	X	I	X
S_{12}	I	X	X	I	I	X	X	I	I	X	X	I	I	X	X
S_{13}	I	I	I	X	X	X	X	I	I	I	I	X	X	X	X
S_{14}	I	I	I	I	I	I	I	X	X	X	X	X	X	X	X

(2.55)

With the Reed Muller code as the seed tensor, we use edge inflation to holographically stack these seed tensors on the Poincaré disk. When aiming to create a max rate holographic Reed Muller code, we employ a $[p, q] = [15, 4]$ method for holographic stacking, as illustrated in the following Fig.2.24:

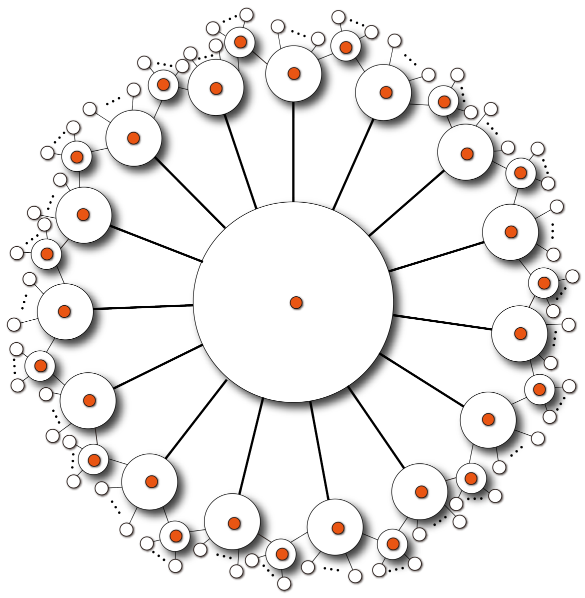


Figure 2.24: Holographic Reed Muller code, following the edge inflation rule.

2.6. Quantum Lego

From the examples of constructing holographic quantum error-correcting codes via tensor networks, we can see that by assembling small error-correcting code tensors into a tensor network, we can systematically create various quantum error-correcting codes. Therefore, we can broaden our perspective beyond holographic tensor networks to view the construction of quantum error-correcting codes through tensor networks more widely.

”Small” tensors act like LEGO bricks, and the process of building tensor networks from these ”small” tensors is akin to constructing a large LEGO structure from individual LEGO bricks. Hence, we aptly refer to this method of constructing quantum error-correcting codes through tensor networks as quantum LEGOs [5].

A crucial aspect of a tensor network is the ”flow” of operators within it. As mentioned earlier, for an isometry, operators can be ”pushed” from one side of the isometry to the other. Thus, operators can be ”pushed” from their original positions to other locations within the tensor network that composes a quantum error-correcting code. Let’s first introduce the protocol for operator pushing.

2.6.1. Operator Push Protocol

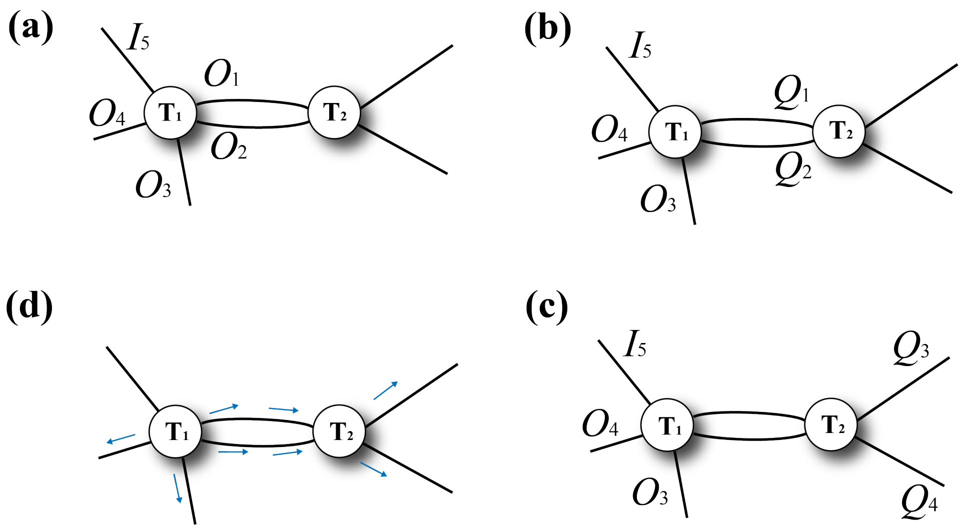


Figure 2.25: (a) Operator O_i is located on the legs of tensor T_1. (b) Operator O_i located on two connected tensor legs is equivalently replaced by operator Q_i on the leg of tensor T_2. (c) Operator Q_i within the tensor network is equivalently replaced by operator Q_j on the boundary through the application of the UPS method. (d) The trajectory of the operator being pushed through the tensor network, forming an ”operator flow”.

Let’s limit our discussion to operators acting on the tensor network that belong to $SU(2)^{\otimes n}$, where n is the total number of tensor legs in the tensor network. Thus, we can choose to represent these operators acting on each $SU(2)$ subspace by drawing them on the tensor legs in the tensor network, indicating that the $SU(2)$ operator acts on the space represented by that index in the tensor network. As shown in Figure (a), we place the operators O_1, O_2, O_3, and O_4 on the legs of the left tensor, while the other legs, not depicted with operators, currently represent the identity operator.

When examining the $SU(2)^{\otimes n}$ operators acting on the depicted tensor network, we find that this $SU(2)^{\otimes n}$ operator actually has many equivalent forms of expression. Therefore, for practical quantum error-correcting codes, we need all non-trivial $SU(2)$ operators to act on the boundary of the tensor network. Thus, we seek methods to ”push” the $SU(2)$ operators to the boundary. We note that the O_3 and O_4 operators are already on dangling legs, indicating they are already at the boundary of the tensor network and do not require further action. However, for O_1 and O_2, since they are on legs contracted with the right tensor, we need to first ”move” O_1 and O_2 to the legs of the right tensor. This ”movement” operation must follow certain rules, which we introduce as the Matching Protocol.

Matching Protocol

For a pair of contracted tensor legs, they are effectively ”glued” together into a pair of Bell pairs, entering a maximally entangled state. The quantum state of the subsystem composed of these two tensor legs can be written in the following form:

$$\left|\Phi^+\right\rangle = \sum_{i=0}^{d-1} |ii\rangle \tag{2.56}$$

Assuming in this subsystem, we originally have an operator O_e on the first leg and the identity operator on the second leg, we need to find a matching pair (O_e, Q_e) such that the action of $O_e \otimes Q_e$ on this subsystem is trivial, as illustrated below.

$$\langle\Phi^{+}\rvert\, O_e \otimes Q_e = \langle\Phi^{+}\rvert\,. \tag{2.57}$$

To better understand this matching protocol, let's consider O_e and Q_e as Hermitian and unitary operators (such as Pauli operators). Then, this expression can be rewritten as:

$$\langle\Phi^{+}\rvert\, O_e \otimes I = \langle\Phi^{+}\rvert\, I \otimes Q_e. \tag{2.58}$$

Thus, by employing this matching protocol, we can find the corresponding Q_e for O_e, thereby ”moving” O_e onto another tensor. It's worth noting that for Pauli operators, O_e and Q_e are equal. This means that for the matching protocol operation with Pauli operators, we are simply moving the Pauli operator from one tensor along the contracted tensor leg to the corresponding tensor leg of another tensor.

Therefore, looking back at figure 2.25 (b), we use the matching protocol to move the operators O_1 and O_2 to the corresponding legs of the right tensor, resulting in Q_1 and Q_2.

To further push Q_1 and Q_2 to the boundary of the tensor network, the next step is to find a way to let Q_1 and Q_2 ”pass through” the right tensor to reach the boundary. To achieve this, we need to use what are called UPS operators.

Unitary Product Stabilizer (UPS)

Definition: For any state $\lvert V\rangle \in \mathcal{H}$, a unitary U that satisfies $U\lvert V\rangle = \lvert V\rangle$ is called a unitary stabilizer of $\lvert V\rangle$. Additionally, if they also satisfy $\mathcal{H} = \mathcal{H}_{\mathbb{Q}_N}^{\otimes d}$ for some prime d and $U = \bigotimes_i U_i$, then U is also a unitary product stabilizer (UPS).

In the subsystem where a tensor in the tensor network resides, the action of the UPS on this subspace is trivial with respect to the code space. Therefore, by selecting UPS that matches Q_1 and Q_2, and applying it to the right tensor, we can push Q_1 and Q_2 to the boundary, resulting in Q_3 and Q_4, as shown in figure 2.25 (c).

Operator Flow

After the operations depicted in figures 2.25 (a), (b), and (c), we successfully find an $SU(2)^{\otimes n}$ operator equivalent to the initial operator, which has non-trivial operators only on the boundary of the tensor network. Understanding this process visually, we find that the operators seem to ”flow” from their original positions to the boundary of the tensor network. The direction of operator flow in the above example is shown in figure 2.25 (d).

3 Methods

3.1. Quantum Lego-Based UPS Generator Program

To study the properties of a quantum error-correcting code, it's essential first to obtain a complete expression of its stabilizer generators and the expressions for the generators of logical operators within the corresponding centralizer group. With holographic codes, as the radius increases, the number of physical qubits grows exponentially. This means that the number of stabilizer generators we need to calculate, as well as the length of Pauli operators for each stabilizer generator, also increase exponentially with the radius, making it a very labor-intensive task. Before this work, there was no program capable of automatically and rapidly handling the operator pushing for quantum legos, primarily due to the complexity of digitizing tensor networks and the intricate logic behind operator pushing.

This work developed an automated operator pushing program based on quantum legos [5] that can generate holographic tensor networks and quickly provide a complete set of stabilizer group generators and the corresponding centralizer group generators. It can even infer the boundary expression from any local bulk operator.

3.1.1. Architecture Diagram of the Program

For the Tensor Network UPS Generator Program, its architecture was designed with three layers: the Object Layer (which includes tensor legs, tensors, and tensor networks), the Operator Push Manager Layer, and the Boundary Reading Layer, as illustrated in the following figure 3.1.

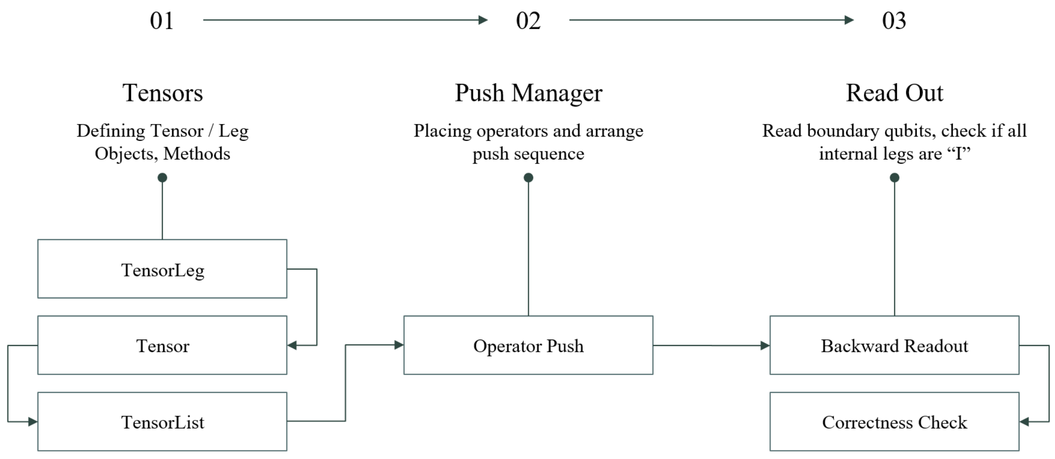

01
02
03
Tensors
Defining Tensor / Leg Objects, Methods
Push Manager
Placing operators and arrange push sequence
Read Out
Read boundary qubits, check if all internal legs are "I"
TensorLeg
Tensor
TensorList
Operator Push
Backward Readout
Correctness Check

Figure 3.1: Architecture of the automated UPS generation program.

As shown in the figure 3.1, the Quantum Lego-Based UPS Generator program is primarily divided into three layers, explained as follows:

1. **Object layer, which includes tensor legs, tensors, and tensor networks.**
   The object layer defines the components required to construct the operator push tensor network, ranging from the smallest to largest scale: tensor legs, tensors, and tensor networks. A detailed explanation will be in provided in the section of Tensor Legs, Tensor Classes, and Tensor Networks.
2. **Operator Push Manager, a function that manages the order of operator pushes.**
   For a holographic tensor network, the Operator Push Manager first receives one or a set of local UPS operators that need to be pushed to the boundary. It then pushes them to the boundary in a specific order, ensuring that most tensors only need to undergo a single push operation. A more detailed explanation will be provided in the later section on the Operator Push Manager.
3. **Read Out Layer**
   Readout function retrieves operators on the boundary using the backtracking method and compiles them into the final boundary operator result. It also includes a correctness check, traversing all internal legs of the tensor network to ensure that all operators have been pushed to the boundary. More detailed explanations will follow in subsequent sections.

3.1.2. Tensor Legs, Tensor Classes, and Tensor Networks

The Object Layer defines all components necessary for operator pushing within the tensor network, including tensor legs class, tensor class, and the tensor network itself. Numerous methods and functions have been developed corresponding to these objects to facilitate the operator pushing. Below, we introduce these classes and some of their basic methods and functions:

1. **TensorLeg**

Table 3.1: Tensor Leg Object's Main Variables and Their Functions

Variables	Type	Description
$operator$	String	Stores the operator located on this tensor leg. Currently, only Pauli operators ("I", "X", "Y", "Z") are supported
$connection$	Tuple	(target tensor ID, corresponding target tensor leg ID). The first element indicates the ID of the target tensor to which this current leg of the current tensor is connected, and the second element indicates which leg of the target tensor this current leg is connected to. If the current tensor leg is not connected (i.e., a dangling leg), then this variable is set to None.
$clifford\_gate$	String	Stores the Clifford gate existing on this tensor leg, such as the Hadamard gate. 'None' or 'I' indicates that there is no Clifford gate on this tensor leg.
$logical$	Boolean	Used to indicate whether this leg is a logical leg.
$blocked$	Boolean	To prevent the operator from being pushed back to the center of the holographic code, this program blocks the corresponding tensor leg after each push operation to prevent the operator from being pushed back.

2. **Tensor**

Table 3.2: Tensor Object's Main Variables and Their Functions

Variables	Type	Description
$tensor\_id$	Integer	Used to store the ID of this tensor, serving as the tensor's unique identifier.

$legs$	List	Used to store the tensor leg objects of this tensor. The index of a leg in the legs list corresponds to the leg ID.
ups_list	List	Used to store a complete set of UPS for the current tensor, treating the tensor as a state. Hence, this ups_list also includes stabilizer generators and complete logical operators, i.e., a set of generators for C(S), where each generator is stored as a string.
$starting\_tensor$	Boolean	Indicates whether this tensor is the starting tensor for initiating operator push. If true, and the current tensor is not located at the center of the bulk, then some legs of this tensor need to be blocked in advance to prevent pushing the operator outside the causal wedge.
$layer$	Integer	Used to store the current tensor's layer in the holographic tensor network.
$stabilizer\_list$	List	Used to separately store the stabilizer group generators.
$logical\_z\_list$	List	Used to separately store the logical Z operation operators. Typically, this list only has one element because usually, we deal with seed tensors that only have one logical leg.
$logical\_x\_list$	List	Used to separately store the logical X operation operators. Similarly, this list usually only has one element.
$incomplete\_logical$	Boolean	For some special tensor network codes, the UPS generators are insufficient to support complete logical operators (this occurs at layer L=2 and beyond for the $[5, 4]$ HTN code). In such cases, this tensor requires special handling.

3. **TensorNetworks**
   The tensor network object is a object used to store all tensors. Currently, a list is used to hold all tensors in the existing code. In future versions released to the open-source community, consideration will be given to replacing the list with a dictionary object to enhance performance.

For these objects, the program has developed many methods and functions to efficiently serve the purpose of operator pushing. Here, we list some basic methods and functions and explain their functions, with a more comprehensive description to be provided in the documentation for the open-source community.

1. **Basic Methods of TensorLeg Objects**

Table 3.3: Basic Methods of TensorLeg Objects

Methods	Description
$operator\_set$	Assigns an operator to this leg.

2. **Basic Methods of Tensor Objects**

Table 3.4: Basic Methods of Tensor Objects

Methods	Description
$add\_stabilizer$	Add a stabilizer UPS to the tensor.
set_leg	Set the information for a specific leg
add_leg	Add a leg to this tensor
$apply\_ups$	Apply a UPS to this tensor, where the single Pauli operators in the UPS act sequentially on the corresponding tensor legs
$pauli\_push$	Push the operator to the next tensor's leg according to the properties of the tensor leg and its connected corresponding tensor leg, following the matching protocol

$get\_connections$	Obtain the connection information of all tensors connected to the current tensor
$ups\_decision$	In the process of using UPS to push operators, appropriately select the suitable UPS from the UPS group to cancel the Pauli operators on the input legs
$operator\_push\_decision$	In this program, tensors are programmed to autonomously decide which method to use for effective operator pushing; this is a push decision function

3. **Basic Functions for TensorNetworks**

Table 3.5: Basic Functions of TensorNetworks

Functions	Description
$get\_tensor\_from\_id$	Given a tensor ID, return the tensor corresponding to that tensor ID.
$remove\_tensor$	Given a tensor ID, remove this tensor from the tensor network and update the topological information of the tensor network.

Above are the three main types of objects used in this program, along with some of the most basic and commonly used methods and functions.

3.1.3. Programmatic Construction of Holographic Tensor Networks

The number of tensors in a holographic tensor network grows exponentially with the radius, making manual input of the tensor network's topological structure into the program highly inefficient. Therefore, the program also needs an automated method to generate specific topological structures.

In this program, there are two methods to generate arbitrarily large holographic tensor networks: one uses the external package Hypertiling to generate max rate holographic codes, and the other uses a method developed in-house to generate tensor networks with any $q = 4$ edge inflation. The latter allows for the creation of holographic codes of various rates and also permits the construction of heterogeneous holographic codes.

For the first approach, which involves obtaining topological information from an external hypertiling package and constructing a tensor network, the logic is relatively simple. However, it is important to pay attention to the order of the legs and their connection directions. Below is the pseudocode for obtaining topological information from an external hypertiling package and constructing a holographic tensor network (usually only for max rate codes):

Algorithm 1 Importing Topology from External Hypertiling Package

```
1: procedure Import_Topology_from_Hypertiling($p, q, n$)
2:     $hypertiling\_obj \leftarrow hypertiling(p, q, n)$ # Create hypertiling object
3:     $topology \leftarrow hypertiling\_obj$.get_nbr() # Get neighbors information of all tiling objects
4:     $tensor\_network \leftarrow []$ # Create an empty TensorNetwork object
5:     # Set the TensorNetwork's topology using Hypertiling's topology
6:     for $connection$ in $topology$ do
7:         create_connection($connection, tensor\_network$)
8:     end for
9:     #Reorder leg sequence according to the requirements of the actual holographic code
10:    for $tensor$ in $tensor\_network$ do
11:        sort_leg_order($tensor, tensor\_network$)
12:    end for
13:    return $tensor\_network$
14: end procedure
```

For the second method, which does not rely on any external package and creates arbitrary q=4 tiling, it is broadly applicable to various non-max rate holographic codes. The pseudocode for this method is as follows:

Algorithm 2 Create $q = 4$ Layer for Tensor Network

```
1: procedure Create_$q$_eq_$4$_Layer_for_Tensor_Network
2: # This function adds a new layer of tensors to a given TensorNetwork with $q = 4$ tiling
3:     Create a $new\_tensor$
4:     Connect the $new\_tensor$ with the first $dangling\_leg$ of the first $tensor$ of the $outermost\_tensors$
5:     Connect the $new\_tensor$ with the last $dangling\_leg$ of the last $tensor$ of the $outermost\_tensors$
6:     for $current\_tensor$ in $outermost\_tensors$ do
7:         for $dangling\_leg$ in $tensor$ do
8:             Create a $new\_tensor$ connected with the $dangling\_leg$ of the $current\_tensor$
9:             if $new\_tensor$'s connection is with the last $dangling\_leg$ of the $current\_tensor$ then
10:                Connect the $new\_tensor$ to the corresponding $neighbor\_tensor$
11:            end if
12:        end for
13:    end for
14: end procedure
```

The function described above can holographically grow one more layer of tensors $(q = 4)$ on the outermost layer, given a holographic tensor network with $q = 4$ edge inflation. Of course, the initialization of the holographic tensor network, such as the tensors of the 0th layer and the growth of the 1st layer, requires special handling. With this function, we can freely generate any $q = 4$ edge inflation holographic code and can vary the types of connected tensors as well as the encoding rate.

3.1.4. Self-Push Operators: Autonomous Tensors

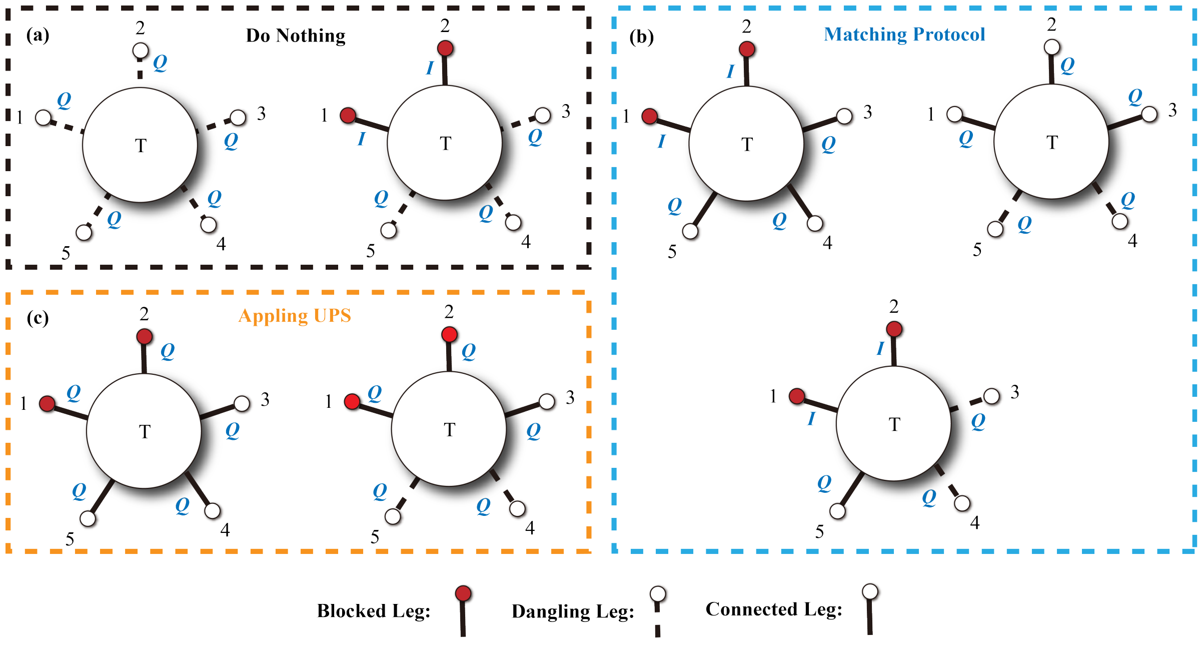


Figure 3.2: (a) When non-identity operators appear only on the tensor's dangling legs and all connected legs are already blocked, no action is required on the tensor. (b) When there are non-identity operators on connected but unblocked legs, and only identity operators exist on connected and blocked legs, use the matching protocol to push operators at that time, and block these connected legs after pushing. (c) When the tensor has at least one blocked leg with a non-identity operator, prioritize applying UPS, then consider whether to use the matching protocol.

As mentioned in the tensor lego section, the operator push process requires the use of the matching protocol and local UPS methods. If these operations are performed directly at the level of the tensor network, the code logic becomes very complex. Therefore, we opt to let tensor objects possess the ability to execute local operator pushes on their own, which we refer to as: Autonomous Tensors.

For an autonomous tensor, the main focus is on the operators and connections on its own legs, as shown in the Figure 3.2. An autonomous tensor needs to take different actions depending on various situations.

1. **Non-identity operators appear only on the tensor's dangling legs and all connected legs are already blocked:** In this case, no action is required on the tensor.
2. **When there are non-identity operators on connected but unblocked legs, and only identity operators exist on connected and blocked legs:** Use the matching protocol to push operators at that time, and block these connected legs after pushing.
3. **When the tensor has at least one blocked leg with a non-identity operator:** Prioritize applying UPS, then consider whether to use the matching protocol.

Based on the above rules, an autonomous tensor can manipulate the operators on its legs and the state of its legs to locally handle operator pushes. Of course, the actual program considers additional rules and features, such as unblocking a leg and arranging for a tensor to undergo another round of local operator pushes. These extra rules are related to the specific tensor network structure and the stabilizer group of the seed tensor.

3.1.5. Operator Push Manager

For the entire tensor network, although each tensor is now an autonomous tensor capable of handling localized operator pushes, there still needs to be an Operator Push Manager to manage the initiation sequence of these autonomous tensor operator pushes.

The main function of the Operator Push Manager is to place one or several local UPS operators that are waiting to be resolved into boundary representations on a fully initialized tensor network, and to reasonably arrange the initiation sequence of the autonomous tensors to ensure that all operators are pushed to the boundary. Therefore, the Operator Push Manager needs to call the autonomous tensors in an effective order.

For the tensor network of a holographic code, the Operator Push Manager first calls the central tensor, allowing the central autonomous tensor to make a round of operator push decisions. Then, using methods defined in the tensor class, the Operator Push Manager retrieves the neighbor tensors connected to the central tensor and adds these neighbor tensors to a queue, which stores the autonomous tensors that will be called in the next round. As the Operator Push Manager processes the autonomous tensors passed from the previous round, it records the neighbors of each tensor being called that have not yet been called and places them in the queue for the next round of calls.

The general logic of the Operator Push Manager is as follows:

Algorithm 3 Operator Push Manager

```
1: procedure Operator_Push_Manager
2:     Call the central tensor $T_c$, and add $T_c$ to a set $Processed\_Tensors$
3:     Retrieve neighbors of $T_c$ and add these neighbors $\{T_{nbr}\}$ to a queue $Q$
4:     while not $Q$.empty() do
5:         Call autonomous tensor $T \leftarrow Q$.pop(), and $Processed\_Tensors$.add($T$)
6:         for each neighbor $T_{nbr}$ of $T$ do
7:             if $T_{nbr}$ not in $Q$ and $T_{nbr}$ not in $Processed\_Tensors$ then
8:                 $Q$.add($T_{nbr}$)                ▷ Add to queue if not already included
9:             end if
10:        end for
11:    end while
12: end procedure
```

3.1.6. Boundary Operator Readout and Correctness Verification

After completing the operator push, there are two main modules for reading boundary operators. The first module reads the boundary operators using a backtracking method, while the second module traverses all internal legs to check whether the operator push has been correctly completed.

For boundary operators readout, a backtracking method is used to sequentially traverse the operators on each dangling leg along the boundary, ultimately returning the Pauli operators for the entire boundary. The reason for using the backtracking method instead of directly traversing each tensor leg is to ensure that the order of the read Pauli operators corresponds to the geometry of the holographic code.

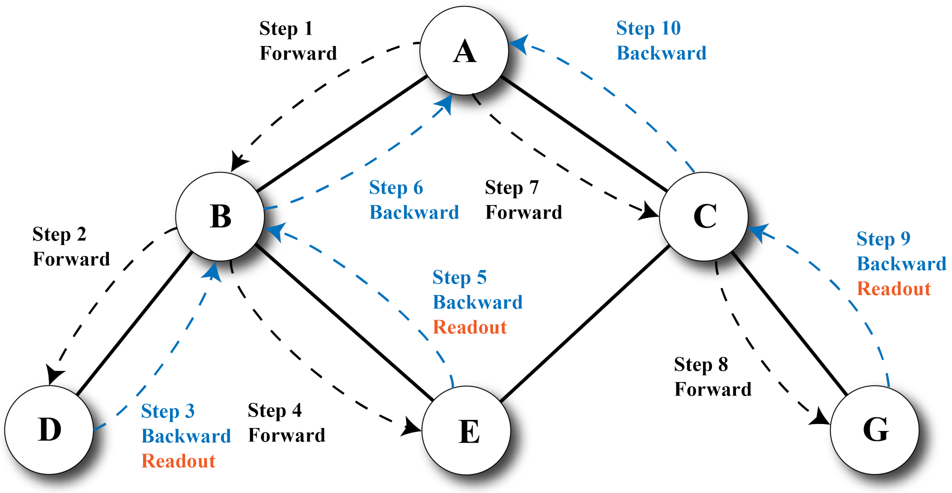


Figure 3.3: A schematic diagram of the Backtracking Boundary Operator Reading Algorithm illustrates the process of traversing each tensor using the backtracking method. Every time a boundary tensor is encountered, the logical operators on the dangling legs of the boundary tensor are read.

3.2. Erasure Decoder

In quantum communication and computing, the information stored in qubits can be lost. In the context of quantum error correction, this loss of quantum information is referred to as an erasure error. Quantum error-correcting codes combat erasure errors by creating redundancy, allowing the original quantum information to potentially be reconstructed from the remaining quantum information. Let's first introduce the quantum erasure channel.

3.2.1. Recoverability of Logical Information under Erasure Channel

When a string of qubits passes through a quantum erasure channel, we receive information about which qubits have been erased. Our primary focus is on the positions of these erased qubits. When assessing the recoverability of quantum information for a certain logical qubit, we typically need to determine if this logical information can be reconstructed using the qubits that have not been erased. This problem is equivalent to whether we can obtain an equivalent set of complete logical operators $\{Lx, Lz\}$ through the stabilizer group of the entire error-correcting code $\langle S_1, S_2, ..., S_{n-k} \rangle$, such that the support of this complete set of logical operators $\{Lx, Lz\}$ on the erased qubits is trivial [89] (i.e., $\{Lx, Lz\}$ acts as the identity operator on the positions of the erased qubits).

3.2.2. Erasure Vector and Filtered Pauli Strings

In our Monte Carlo simulations, we need to randomly generate quantum erasure errors with certain probabilities and express these erasure errors mathematically using an erasure vector. Given a quantum error-correcting code with n physical qubits, our random erasure vector is also a binary symplectic vector [94] of length n. For the ith element of the erasure vector, if it is 1, it means that the ith qubit has experienced a quantum erasure error; conversely, if this element is 0, it indicates that no quantum erasure error has occurred at this position.

Since our criterion for judging the recoverability of logical information is whether there exists an equivalent logical operator whose support on the erased qubits is trivial, we are more interested in the subspace of erased qubits. Therefore, we need to filter our error-correcting code's logical operators and stabilizer group generators through the erasure vector [89].

The filtering rule is as follows: for the ith qubit, if the ith bit of the erasure vector is 1, then retain the operator at the ith position of the logical operator and stabilizer group generators. If the ith bit of the erasure vector is 0, then delete the operator at the ith position of the logical operator and stabilizer group generators. Thus, for an erasure vector of weight m (containing m ones), we filter the original logical operators and stabilizer group generators of length n into shorter ones of length m ($m \leq n$). An example of the filtering process is illustrated in the diagram.

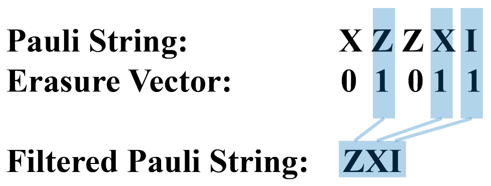


Figure 3.4: An example of Pauli operator filtering: given the Pauli string $XZZXI$ and the erasure vector 01011, the filtered Pauli string according to the filtering rules is ZXI.

3.2.3. Binary Representation of Pauli Strings

Before actually assessing the recoverability of a quantum error-correcting code under an erasure channel using an erasure decoder, we first need to convert the Pauli strings into binary vector form to facilitate subsequent mathematical processing [94].

Given a Pauli string S_p of length l, composed of single Pauli operators (I, X, Y, Z), we convert each single Pauli operator into a binary vector of length 2, as follows:

Pauli Operator:	Binary Representation
I	(0, 0)
X	(1, 0)
Y	(1, 1)
Z	(0, 1)

(3.1)

For a Pauli string of length l composed of single Pauli operators, we create its binary vector representation in the following way:

1. Create an empty vector S_{Binary} of length $2l$.
2. For i ranging from 1 to l, for the ith single Pauli operator in the original Pauli string, we follow the rule for converting single Pauli operators into binary vectors. We place the first bit of the binary vector representing this single Pauli operator at the ith position of S_{Binary}, and the second bit of the binary vector at the $(i + l)$th position of S_{Binary}, continuing until the entire traversal is completed.

3.2.4. The Mathematical Method for Judging the Recoverability of Logical Information

Once we have the filtered logical operators $\{X^{\text{filtered}}, Z^{\text{filtered}}\}$ and the similarly filtered complete set of stabilizer generators $\{S_1^{\text{filtered}}, \ldots, S_{n-k}^{\text{filtered}}\}$, the judgment of the recoverability of logical information is equivalent to finding whether there exists a set of combinations of stabilizer generators $\{\lambda_i\}$ that make the following equation hold:

$$\ell^{(filtered)} + \sum_i \lambda_i S_i^{(filtered)} = 0 \mod 2. \tag{3.2}$$

where $\ell^{(filtered)}$ is either X^{filtered} or Z^{filtered}.

The question of the existence of solutions for this Boolean linear equation system can be transformed into a matrix form. We can write the augmented matrix corresponding to this set of equations:

$$M = (\{S_i^{(filtered)}\}|\{\ell_j^{(filtered)}\}) \tag{3.3}$$

Where $\{\ell_j^{\text{(filtered)}}\}$ is the set of logical operators whose recoverability we wish to assess.

Thus, the problem of the Boolean equation system having a solution is transformed into a matrix transformation problem for the augmented matrix. We use binary mod-2 Gaussian elimination to transform the matrix M into Reduced Row-Echelon Form (rref), and then we compare the rank of the stabilizer matrix S and the augmented matrix M. If the ranks are equal, then the equation system has a solution, and the logical information is recoverable. If they are not equal, then the equation system does not have a solution, and the logical information cannot be recovered.

3.2.5. Architecture of the Erasure Decoder

Overall, our quantum erasure error decoder is primarily used for determining recoverability and needs to natively support multiple Monte Carlo simulations to obtain the recoverability probability P_{rec} at a specific erasure probability P_e. Therefore, in a single Monte Carlo simulation, we first generate an erasure vector of length n according to the erasure probability P_e, then use this erasure vector to perform filtering operations on the logical operators and stabilizer generators. Next, we use Boolean linear algebra methods to judge the recoverability of logical information under this erasure error. Through multiple Monte Carlo simulations, we tally the number of successful recovery instances n_{succ} and the total number of simulations N, to estimate the recoverability probability $P_{rec} = n_{succ}/N$.

3.3. Integer Optimization Decoder

For decoding Pauli errors, the minimum weight decoder is a viable option. This minimum weight decoder selects and returns the lowest weight element from the set of all possible Pauli strings that could cause a given syndrome γ. The integer optimization decoder [95] is a minimum weight decoder based on integer optimization methods and can be applied to the decoding of any stabilizer code.

3.3.1. From Syndrome to possible Errors: Pseudoinverse Matrix

First, let's explore how to infer the Pauli errors that can cause the same syndromes from the syndromes obtained by stabilizer measurements. Initially, our binary form of the Pauli error vector is e_0, which results in the syndrome γ when acted upon by the stabilizer matrix S.

$$\gamma = Se_0 \tag{3.4}$$

We know that the stabilizer matrix S is not a square matrix. So, in the strict sense of linear algebra, it does not have an inverse matrix. However, we can generate a pseudoinverse matrix F of the stabilizer matrix S such that SF equals the identity matrix I. In this case, our guess for the Pauli error e that can cause the same syndrome can be written as:

$$e = F\gamma \tag{3.5}$$

So that:

$$Se = SF\gamma = \gamma \tag{3.6}$$

From this, we can see that our guessed Pauli error e will cause the same syndrome as the actual error e_0.

3.3.2. Generation of the Pseudoinverse Matrix for Holographic Codes

When it comes to calculating the pseudoinverse matrix, methods such as the Moore–Penrose inverse might come to mind first. Unfortunately, since the elements within our binary representation of the S matrix are not integers but Boolean values, and the elements of S commute with each other, we cannot directly use existing pseudoinverse formulas to solve for the F matrix directly. Moreover, the S matrix in holographic tensor network codes is usually huge, making it impractical to search for the pseudoinverse matrix through brute force. In the actual decoding process, we need to cleverly generate the columns of the pseudoinverse matrix using the operator push program mentioned earlier.

First, we identify from which tensor the stabilizer UPS in the jth row of the S matrix originates and find the local UPS expression s_i^{local} of this tensor's stabilizer, where i indicates which stabilizer generator this is for the tensor. Since this local tensor is much smaller compared to the entire tensor network, we then search for a Pauli string f_i^{local} of size matching the local tensor. This Pauli string f_i^{local} is chosen such that it anti-commutes only with the selected local stabilizer and commutes with the rest of the stabilizers of this tensor. We refer to this f_i^{local} as the anti-commuter solely for s_i^{local}. Next, we use the operator push program to push this local anti-commuting operator to the boundary to obtain its boundary expression f_j^{boundary}, and then we insert this f_j^{boundary} as the jth column into the F matrix.

The reason why choosing a local tensor's anti-commuter and pushing it to the boundary can obtain an anti-commuter for the entire tensor network is because, when searching for the anti-commuter, we need to consider the commutativity with stabilizer generators other than the selected one. During the process of pushing operators to the boundary, we also use local UPS for operator pushing. Thus, in the process of pushing operators from local to global, we ensure the preservation of commutativity and anti-commutativity.

3.3.3. Combinatorial Optimization Problem: Minimizing the Weight of Possible Errors

After obtaining our guessed error e through the pseudoinverse matrix, it's important to clarify that this error e may not be equivalent to the original, actual error e_0. For simplicity, let's assume a certain error-correcting code encodes only one logical qubit. The errors that can produce the same syndrome as e_0 actually belong to different cosets of the stabilizer subgroup S: e_0S, Xe_0S, Ye_0S, Ze_0S. Therefore, we need to select the most likely coset among these as the final output of the decoder.

Based on the common depolarizing error model, the probability of a Pauli error occurring on a qubit is p, with the occurrence probabilities of the three Pauli errors X, Y, and Z each being $p/3$. We consider a single Pauli error as having a weight, and it's clear that Pauli errors with lower weights are more likely to occur than those with higher weights because $p^n > p^{n+1}$ (with $0 < p < 1$).

Therefore, the most likely Pauli error to occur in the entire system would be the one among those that can cause the same syndrome but has the smallest weight.

We can act on our guessed Pauli error e_0, inferred through the pseudoinverse matrix, with the entire tensor network's stabilizer generators and logical operators to obtain a new optimization target e'. Then, we find the optimal combination that minimizes the weight of e'. Therefore, the decoding problem becomes a combinatorial optimization problem:

$$e' = e + \sum_{\ell} \lambda_\ell s_\ell + \sum_m \mu_m L_m, \tag{3.7}$$

The optimization task is to find the appropriate combinations of $\{\lambda_\ell\}$ and $\{\mu_m\}$ such that the weight of e' is minimized.

3.3.4. From Combinatorial Optimization Problems to Integer Optimization Problems

As is well-known, combinatorial optimization problems are exceedingly challenging because the most naive approach, brute-force search, requires exploring 2^n combinations for n Boolean parameters. This is particularly daunting for holographic codes, which inherently scale exponentially. Therefore, at this step, we need to leverage external integer optimization decoders that employ smarter strategies to significantly save on optimization time. In this work, we chose to use the Gurobi optimizer. To this end, we need to equivalently transform the combinatorial optimization problem into a mixed-integer linear optimization problem for Gurobi to optimize.

First, we set the coefficients $\{\lambda_\ell\}$ and $\{\mu_m\}$ required for the combinatorial optimization. For the jth element of our optimization target e' in its binary vector form, we select every element $\{s_{ij}\}_{i=1}^{n-k}$ from the jth row of the matrix composed of binary stabilizer vectors $\{s_i\}_{i=1}^{n-k}$ and every element $\{L_{mj}\}_{m=1}^{2k}$ from the jth row of the matrix composed of binary logical operators $\{L_m\}_{m=1}^{2k}$, and multiply them by the coefficients $\{\lambda_\ell\}$ and $\{\mu_m\}$, respectively, to obtain $\{\lambda_i s_{ij}\}_{i=1}^{n-k}$ and $\{\mu_m L_{mj}\}_{m=1}^{2k}$. Then, we sum these elements to get the optimization variable corresponding to the jth element of e' in its binary vector form.

It's important to note that each optimization variable corresponding to an element of e' in its binary vector form is a sum of values, and hence, these optimization variables are not binary. This is inconsistent with our mod-2 addition. Therefore, in the integer optimizer, for each optimization variable to be able to represent a Boolean value, we need to introduce an additional variable and add some constraints to ensure that the output is a Boolean vector.

Ultimately, since we consider the weight of each Pauli error as 1, our final optimization objective is:

$$Obj = \sum_{j=1}^{n}\{or(\sum_{i=1}^{n-k}\{\lambda_i s_{ij}\} + \sum_{m=1}^{2k}\{\mu_m L_{mj}\}\ mod\ 2, \sum_{i=1}^{n-k}\{\lambda_i s_{i(j+n)}\} + \sum_{m=1}^{2k}\{\mu_m L_{m(j+n)}\}\ mod\ 2)\} \tag{3.8}$$

By optimizing this objective with Gurobi and then returning the corresponding coefficients, we can calculate the minimum weight possible Pauli error, min(e').

3.3.5. Judging the Success of the Decoding Result

The minimum weight possible error min(e') we obtain may be the same as or different from the original, actual error e_0. The method to determine whether they are equivalent in logical space is to check if min(e') can be obtained from e_0 through the combined action of stabilizer generators.

This problem is equivalent to whether the following equation has a solution:

$$min(e') = \sum_i \lambda_i s_i + e_0\ (mod\ 2) \tag{3.9}$$

Upon rearranging this equation, we find that it is equivalent to the problem of solvability of the Boolean linear equation system we previously encountered in the context of the quantum erasure decoder:

$$0 = \sum_i \lambda_i s_i + e_0 + min(e')\ (mod\ 2) \tag{3.10}$$

We only need to use a portion of the code from the quantum erasure decoder to determine whether this equation has a solution. If a solution exists, then the decoding attempt by the integer optimization decoder is considered successful; otherwise, it is deemed a failure.

3.4. Code Distance Calculator

For a quantum error-correcting code of the form $[[n, k, d]]$, its stabilizer group possesses $n-k$ generators, and the logical space has $2k$ logical operators. The code distance d can sometimes be directly inferred from the properties inherent in the code's construction. However, the code distance d is not always so intuitively obtainable. For holographic codes, to date, no particularly expedient method has been found to calculate the code distance. Therefore, this work has developed a more universally applicable code distance calculator based on integer optimizers to compute the code distances of holographic codes.

For a given stabilizer group $S =< s_1, s_2, ..., s_{n-k} >$ and a given logical operator ℓ, we obtain a logical operator ℓ' equivalent to ℓ by combining the stabilizers:

$$\ell' = \ell + \sum_i \lambda_i s_i \tag{3.11}$$

Here, λ_i are variables considered in the combinatorial optimization. We then input this combinatorial optimization problem into the minimization optimization process mentioned in the previous section on integer optimization decoders. This process yields an ℓ' with the minimum weight. The weight of this minimal weight operator is the smallest weight that can perform the logical operation within the logical space, which corresponds to the distance of this logical operation.

3.5. Variable Weight Integer Optimization Decoder

This is another original method in this work besides the tensor network operator push program. In traditional integer optimization decoders, the decoder only uses stabilizer measurement results (syndromes) to infer the most likely errors. Under depolarizing noise, such decoding strategies are effective. However, for decoding under biased noise channels, traditional integer optimization decoders may not fit the noise model well under biased noise since they consider all types of Pauli errors to have equal weight. To achieve better decoding performance and threshold behavior under biased noise models, this work improved the integer optimization decoder by flexibly defining the weights of X, Y, Z errors according to different noise biases r_x, r_y, r_z, thereby achieving better decoding results.

3.5.1. From Biased Noise Models to Weights

Before discussing the relationship between weights and noise models, let's look at a specific example.

For a noise model where $P_y = P_x^2 = P_z^2$, we can consider that the probability of a Y error occurring on a qubit is equivalent to the simultaneous occurrence of two X errors or two Z errors. Therefore, in the integer optimization representation, the weight of Y, $wt(Y)$, is twice that of X or Z: $wt(Y) = 2wt(X) = 2wt(Z)$.

From the above example, it's evident that the weights and the probabilities of various Pauli errors under a noise model have an exponential relationship.

$$P_i \propto \text{exp}(wt(i)),\ 0 < \text{base} < 1 \tag{3.12}$$

To generalize this relationship, we choose a common base P_{base} and then link the weights with P_x, P_y, and P_z:

$$P_x = P_{\text{base}}^{\text{wt}(X)}, \tag{3.13}$$

$$P_z = P_{\text{base}}^{\text{wt}(Z)}, \tag{3.14}$$

$$P_y = P_{\text{base}}^{\text{wt}(Y)} \tag{3.15}$$

By selecting P_{base} appropriately, we can align the probability of each type of Pauli error with its respective weight in the error model. This allows us to tune the decoder to the specific characteristics of the noise, which is essential for optimizing the decoding under biased noise conditions.

We know the total probability of a Pauli error occurring on a qubit is:

$$P_{\text{total}} = P_x + P_y + P_z \tag{3.16}$$

And the probability of each Pauli error in relation to the bias is:

$$P_x = r_x P_{\text{total}}, \tag{3.17}$$

$$P_y = r_y P_{\text{total}}, \tag{3.18}$$

$$P_z = r_z P_{\text{total}} \tag{3.19}$$

In order for the weights $wt(X)$, $wt(Y)$, $wt(Z)$ to return to 1 when the noise model reverts to a depolarizing model, we choose:

$$P_{\text{base}} = \frac{P_{\text{total}}}{3} \tag{3.20}$$

Thus, we derive the following relationships from the noise model to the weights:

$$wt(X) = \log_{\frac{P_{\text{total}}}{3}}(r_x P_{\text{total}}), \tag{3.21}$$

$$wt(Z) = \log_{\frac{P_{\text{total}}}{3}}(r_z P_{\text{total}}), \tag{3.22}$$

$$wt(Y) = \log_{\frac{P_{\text{total}}}{3}}(r_y P_{\text{total}}) \tag{3.23}$$

3.5.2. From Variable Weights to Integer Optimization Models

Now that we understand how to determine the weights of each Pauli error from the noise model, we can describe how to reflect these weights in the optimization objective of the integer optimizer.

We know that the original binary Pauli error vector is actually composed of two parts: the first part represents a binary vector for X errors, and the second part represents a binary vector for Z errors:

$$\vec{e} = \{\vec{e_x}|\vec{e_z}\}$$

Thus, we can consider $\vec{e_x}$ and $\vec{e_z}$ separately, and similarly, we can have a binary vector $\vec{e_y}$ representing Y errors. According to Boolean algebra, we know that the elements of $\vec{e_y}$ can be obtained from the elements of $\vec{e_x}$ and $\vec{e_z}$:

$$\vec{e_y}_i = and(\vec{e_x}_i, \vec{e_z}_i) = \vec{e_x}_i + \vec{e_z}_i - or(\vec{e_x}_i, \vec{e_z}_i)$$

Therefore, we set the optimization objective as:

$$OptObj = \sum_{i=1}^{n}(c_x \vec{e_x}_i + c_z \vec{e_z}_i + c_y \vec{e_y}_i)$$

We can derive the relationship between the weights and the coefficients c_x, c_y, c_z as follows:

$$c_x = wt(X) \tag{3.24}$$

$$c_z = wt(Z) \tag{3.25}$$

$$c_y = wt(Y) - wt(X) - wt(Z) \tag{3.26}$$

In this way, we can integrate the weights with the mathematical expression of the optimization objective inside the integer optimization decoder, thereby obtaining an integer optimization decoder with variable weights.

3.6. Tensor-Network Decoder

The previous chapters introduced decoders based on integer optimization and their variants. The advantage of such integer optimization decoders is that they save memory, but their computational complexity grows exponentially with the number of physical qubits $O(2^n)$. For holographic codes, the number of physical qubits, n, grows exponentially with the radius R. Therefore, decoding holographic codes with large radii often requires considerable computational time. As a result, this work also employs tensor network decoders [2, 96–98] to decode holographic codes, whose computational time complexity is O(n log n), which significantly speeds up the decoding process, and it is an approximation of the maximum-likelihood decoder. The trade-off is a larger memory overhead compared to the integer-optimization decoders from previous sections.

As a type of tensor network code, holographic codes are naturally suited to being decoded using tensor network decoders. Next, we introduce the basic principles of tensor network decoders and how to apply tensor network decoders on holographic codes.

3.6.1. Maximum-Likelihood Decoding Based on Tensor Networks

For a Pauli error e occurring on physical qubits, we can obtain its corresponding syndrome $\vec{s}$ through stabilizer checking. Then, based on this syndrome, we use the pseudo-inverse matrix F of the stabilizer matrix S to find a pure error $e(\vec{s})$ that causes the same syndrome as the original Pauli error e. In fact, the pure error $e(\vec{s})$ is obtained by mapping the original Pauli error e to the orthogonal complement of the column vectors of matrix S through the matrix FS. Thus, the possible space for the pure error $e(\vec{s})$ can be divided into several subspaces, which are the cosets formed by the stabilizer group and logical operators multiplied by the Pauli error e.

The task of the decoder is to find the pure error $e(\vec{s})$ so that $e(\vec{s})e$ resides in the S space. In this case the pure error $e(\vec{s})$ can be obtained from e through the action of some stabilizers, making them equivalent in the logical space.

The maximum likelihood decoder needs to return a pure error $e'(\vec{s})$ that is most likely to return the system to the code space among all logical subspaces. We apply different logical operators L to the original pure error $e(\vec{s})$ obtained via the pseudo-inverse matrix to place it in different logical subspaces. Therefore, we want to calculate the probability of the occurrence of the pure error $e(\vec{s})$ within a coset SL:

$$\chi(L,\vec{s}) = \sum_{S\in\mathcal{S}} \text{Prob}[e(\vec{s})|SL] \tag{3.27}$$

In this equation, the syndrome $\vec{s}$ is given, and $L \in \mathcal{L}$., so the decoder actually needs to find $\bar{L}$ that maximizes $\chi(L,\vec{s})$, that is, $\bar{L} = \arg\max_L \chi(L,\vec{s})$. It is worth noting that χ as a probability is normalized:

$$\sum_{L\in\mathcal{L}} \sum_{\vec{s}} \chi(L,\vec{s}) = 1 \tag{3.28}$$

We denote the four Pauli operators as $\sigma^0 = I$, $\sigma^1 = X$, $\sigma^2 = Y$, and $\sigma^3 = Z$. In this work we have only considered the case of single Pauli errors. Therefore, we assume that all Pauli errors are independent events, and hence we have:

$$\text{Prob}(\sigma^{a_1} \otimes \cdots \otimes \sigma^{a_n}) = \prod_{i=1}^{n} p(\sigma^{a_i}), \tag{3.29}$$

where

$$p(\sigma^{a_i}) = \begin{cases} 1-p & \text{if } a_i = 0 \\ pr_x & \text{if } a_i = 1 \\ pr_y & \text{if } a_i = 2 \\ pr_z & \text{if } a_i = 3. \end{cases} \tag{3.30}$$

Where p is the probability of a Pauli error occurring, and r_x, r_y, r_z are the probability weights for Pauli X, Y, and Z errors occurring, respectively, satisfying $r_x + r_y + r_z = 1$.

Next, we need to construct a mapping from the Pauli errors and their noise model in the physical qubits space to the corresponding probability $\chi(L,\vec{s})$ in the logical space. As holographic codes are stabilizer codes based on tensor networks, this mapping can be intuitively represented using tensor network methods. Initially, stabilizer codes can be represented as tensors. In tensor network decoders, their corresponding tensor representation is as follows:

$$T(L)_{(g_1,\ldots,g_n)} = \begin{cases} 1 & \text{if } \sigma^{g_1} \otimes \cdots \otimes \sigma^{g_n} \in LS \\ 0 & \text{otherwise,} \end{cases} \tag{3.31}$$

It is worth noting that, unlike the tensors mentioned earlier that represent quantum states and mappings, the tensor $T(L)$ here is not an isometry but rather represents the structure of different logical subspaces. This is very useful when calculating probabilities.

When decoding codes based on tensor networks, we can conveniently contract these $T^1(L)$ seed tensors with some seed tensors T^0 that do not contain logical qubits, thereby obtaining the structure of the logical subspaces of the entire tensor network code.

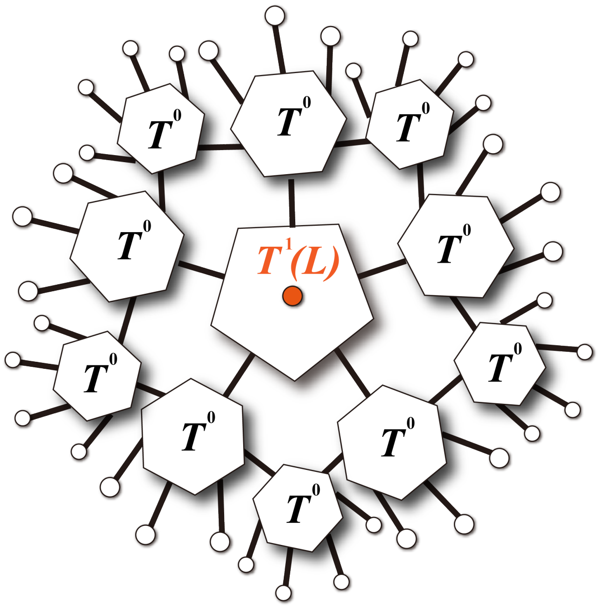


Figure 3.5: Diagram of a zero rate holographic code under a tensor network decoder.

Now, assuming we have obtained the tensor $T(L)$ that represents the logical subspace structure of the entire code, the corresponding $\chi(L,\vec{s})$ can be expressed as:

$$\chi(L,\vec{s}) = \sum_{r_1,\ldots,r_n \in \{0,1,2,3\}} T(L)_{(r_1,\ldots,r_n)} \prod_{i=1}^{n} p(\sigma^{e_i}\sigma^{r_i}), \tag{3.32}$$

Where $p(\sigma^{e_i}\sigma^{r_i})$ is a one-dimensional tensor with a bound dimension of 4, which is a vector:

$$p(\sigma^{e_i}\sigma^{r_i}) = \begin{cases} (1-p, pr_x, pr_y, pr_z) & \text{if } \sigma_{e_i} = I \\ (pr_x, 1-p, pr_z, pr_y) & \text{if } \sigma_{e_i} = X \\ (pr_y, pr_z, 1-p, pr_x) & \text{if } \sigma_{e_i} = Y \\ (pr_z, pr_y, pr_x, 1-p) & \text{if } \sigma_{e_i} = Z \end{cases} \tag{3.33}$$

These $p(\sigma^{e_i}\sigma^{r_i})$ vectors are connected to the i-th physical qubit of the code, thus the tensor network representation of $\chi(L,\vec{s})$ is illustrated as follows in the diagram:

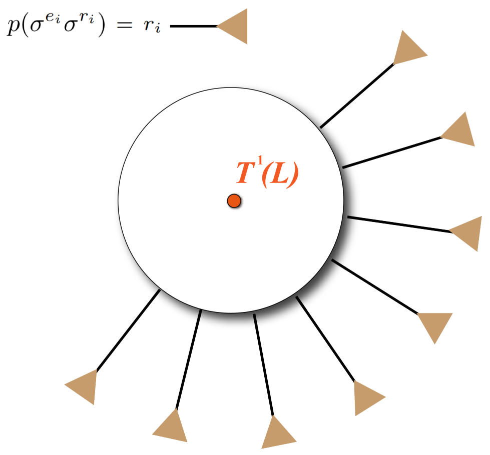


Figure 3.6: To obtain χ, the p vector is connected to each boundary dangling leg and contracted, thus the final result of this contraction is χ.

Since the number of physical qubits n is typically not small, directly representing the entire code's $T(L)$ is impractical. Therefore, we can represent $T(L)$ through the contraction of seed tensors within the original tensor network of the holographic code. Take the HaPPY code with $R = 1$ as an example.:

$$T(L)_{(r_1,\dots,r_{25})} = T^1(L)_{(g_1,\dots,g_5)} T^0_{(g_1,r_1,\dots,r_5)} \cdots T^0_{(g_5,r_{21},\dots,r_{25})} \tag{3.34}$$

By appropriate tensor contraction, we can obtain the final result $\chi(L, \vec{s})$ with reasonable memory overhead.

3.6.2. Backtracking Contraction Algorithm

For the contraction of a large tensor network, the order of contraction is often very important because we aim to avoid generating excessively large tensors during the contraction process. This work proposes a contraction algorithm based on backtracking, which can keep the size of intermediate tensors within a reasonable range during the contraction process of holographic codes.

First, let's introduce the traditional backtracking method. For a simple tree structure, we start from the root node and perform a depth-first search until we reach the bottom-most nodes of the tree. Then, we backtrack to the previous level node, recording information about the edges passed during this return. If there are any lower-level nodes that haven't been visited yet, we perform depth-first searches on them, repeating the process described above.

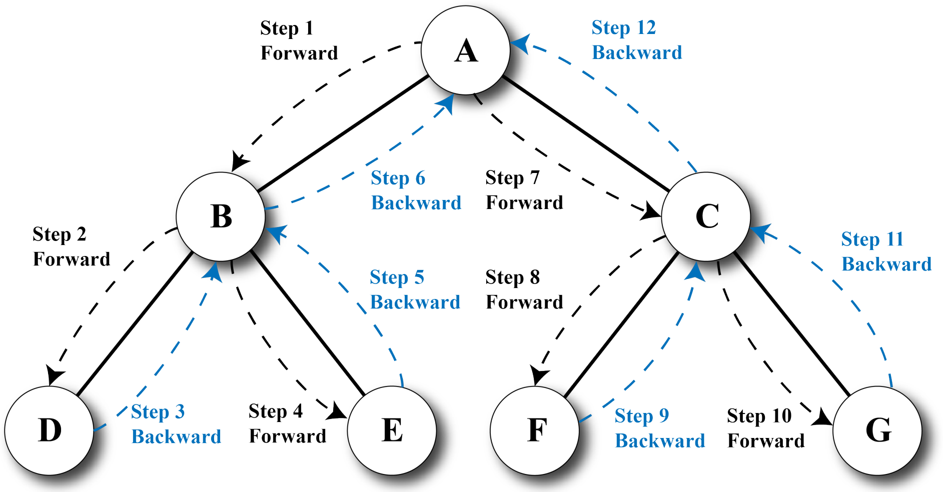


Figure 3.7: Backtracking algorithm for simple tree structures.

For holographic codes, the structure is not a simple tree; some nodes may have more than one parent node, but this does not affect the logic of backtracking. In fact, for the special tree-like structure of holographic codes, it is only necessary to ensure that each backtracking edge is the same as the one previously used to reach that node. Additionally, each search towards the boundary should only visit nodes that have not been visited before.

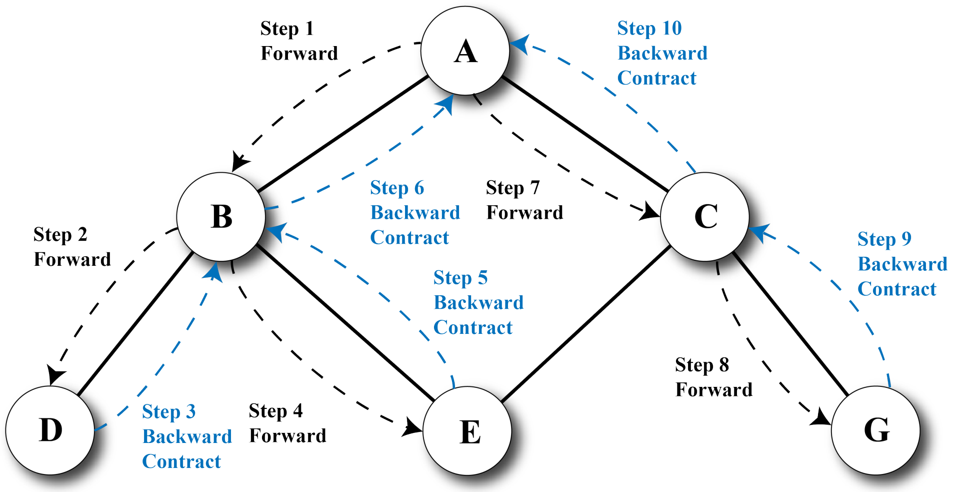


Figure 3.8: Backtracking algorithm for holographic tree structures.

Given a holographic tensor network structure, the pseudocode for the Backtracking Contraction Algorithm is as follows. It implements iterative calls to sequentially return edges, allowing subsequent tensor contractions to be performed in the order of these edges:

Algorithm 4 Backtracking Contraction Algorithm

```
1: procedure CollectEdges($tensor\_network, starting\_tensor\_id$)
2:     $edges \leftarrow []$
3:     $visited \leftarrow$ set()
4:     RecurseCollect($tensor\_network, starting\_tensor\_id,$ None$, visited, edges$)
5:     return $edges$
6: end procedure

7: procedure RecurseCollect($tensor\_network, current\_id, prev\_id, visited, edges$)
8:     $current\_tensor \leftarrow tensor\_network(current\_id)$
9:     $visited$.add($current\_id$)
10:    for $neighbor\_id$ in neighbors of $current\_tensor$ do
11:        if $neighbor\_id \notin visited$ and $neighbor\_tensor$.layer $>$ $current\_tensor$.layer then
12:            RecurseCollect($list, neighbor\_id, current\_id, visited, edges$)
13:        end if
14:    end for
15:    if $prev\_id$ is not None then
16:        $edges$.add(edge from $prev\_id$ to $current\_id$)
17:    end if
18: end procedure
```

For holographic codes, tensor contractions are performed according to the edge order obtained from the previously described Backtracking Contraction Algorithm. The maximum number of tensor legs in the intermediate process tensors grows linearly with the radius of the holographic code, significantly reducing memory overhead.

3.7. Biased Noise Threshold Study

The threshold of holographic codes under biased noise is a key focus of this research. Two representative types of holographic codes have been thoroughly tested for thresholds under biased noise in this work: one is the HaPPY code, which is based on non-CSS perfect tensors, and the other is the holographic Steane code, based on self-dual CSS block perfect tensors.

3.7.1. Selection of Data Points for Biased Noise

We select the distribution of r_x, r_y, r_z combinations on a ternary plot. Initially, we consider the typical depolarizing noise point, which is located at the center of the ternary plot, meaning $r_x = r_y = r_z$. The distribution of other biased noise points is shown in the diagram:

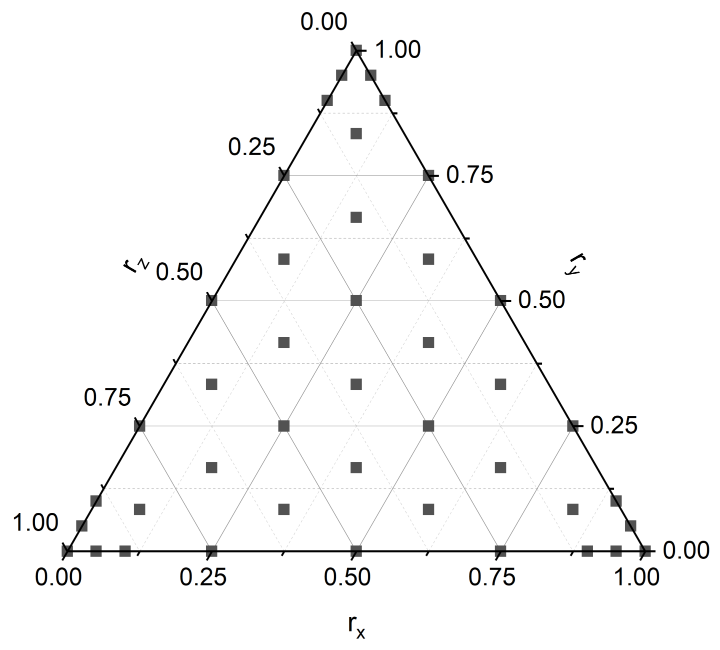


Figure 3.9: Distribution of bias points (rx, ry, rz) tested in the biased noise threshold study.

Hashing Bound Comparison Data Points

This work plans to experiment with different bias in a specific bias direction ($\eta = r_z/(r_x + r_y)$ with $r_x = r_y$) and compare them to the hashing bound. The selected biases are $\eta = 1/2$, 1, 3, 10, 30, 100, 300, 1000, and ∞.

3.7.2. Arrangement of Monte Carlo Experiments

To obtain the threshold of holographic codes at each biased noise point, this work employs Monte Carlo experiments to estimate the recovery probability p_{rec} for holographic codes of different sizes at various error probabilities p. Curves are then plotted to derive threshold estimates for each biased noise point.

Monte Carlo experiments using a tensor-network decoder.

1. **Large-Granularity Rapid Search**: Monte Carlo experiments are conducted only for holographic codes at $R = 0$ and $R = 1$, with $10,000$ Monte Carlo trials. This approach provides an approximate range of thresholds, facilitating a more focused and precise search in the next step.
2. **Localized Fine-Granularity Precision Verification**: In this part of the precise, small-range search, Monte Carlo experiments are conducted within the range of $[th_{approx} - 0.10, th_{approx} + 0.10]$ with a step size of $p_{step} = 0.01$. Tests are carried out for holographic codes of sizes $R = 0$, $R = 1$,

$R = 2$, and $R = 3$. The number of Monte Carlo trials for each size of the holographic code is consistent, with $N_{MC} = 50,000$.

For each combination of biased noise parameters (r_x, r_y, r_z) at a specific error probability p, our workflow primarily involves the following steps in each Monte Carlo cycle: Randomly generate errors according to the parameters (p, r_x, r_y, r_z), then generate syndromes based on the stabilizer matrix. The syndromes and bias parameters are then fed into the decoder. The result of the decoding is passed to a layer that judges whether the decoding was successful. If the decoding is successful, it is counted as a successful decode. After N Monte Carlo simulations, the total number of successful decodings, N_{succ}, is counted. This count is then used to estimate the recovery probability, P_{rec}.

For a holographic code of a given size, we test all bias noise points. For each bias noise point, we increment the error probability p from p_{start} to p_{end} in steps of p_{step}, thereby plotting the P_{rec} vs. p curve.

This work plans to conduct the aforementioned tests on HaPPY codes and holographic Steane codes with radii $R = \{0, 1, 2, 3\}$.

3.8. Threshold Study of HTN Codes

Prior to this work, while there were some preliminary results regarding the erasure noise threshold for HTN codes, there hadn't been any particularly comprehensive large-scale threshold testing for HTN code. The main difficulty was the lack of an automated process capable of performing operator push operations before this work, making it challenging to scale the codes sufficiently to observe threshold behavior. With the operator push program developed in this work, studying the thresholds of HTN codes under various noise models has become feasible.

The research on HTN is divided into several main parts. The first part involves studying the threshold of the [5, 4] HTN code under quantum erasure errors. The second part examines the distance of the [5, 4] HTN code at different sizes. The third part investigates the threshold of the [5, 4] HTN code under depolarizing noise and pure Pauli noise.

3.8.1. Study of the Threshold of HTN Codes under Quantum Erasure Channels

This work investigates the [5,4] HTN codes, categorized by rate, including the max rate HTN, zero rate HTN (gauge fixing), and constant rate HTN.

Max Rate HTN

The previous chapters introduced the max rate HTN code and its construction methods. We placed this code under a quantum erasure channel and then calculated the recoverability of the qubits encoded by the central tensor, thus investigating the existence of a threshold for the maximum rate HTN code.

Zero Rate HTN

For the zero rate HTN codes, this work utilizes the original max rate HTN code, projecting the logical legs of all non-central tensors onto the eigenvalues of a specific logical operator (X, Y, Z) to fix the gauge, thereby obtaining zero rate HTN codes. Consequently, this study examines the recoverability performance of the logical qubits encoded in the central tensor of the X gauge fixed HTN code, Z gauge fixed HTN code, and Y gauge fixed HTN code under a quantum erasure channel, and investigates the existence of thresholds.

Constant Rate HTN

For the constant rate HTN codes, this work tested a type of constant rate HTN code proposed in Ref.[92]. The logical qubit encoding rule for this constant rate HTN code starts from the central tensor, which occupies two geodesics. These two geodesics are highlighted, and then the process moves into the next layer of tensors to encode the logical qubits. If two unhighlighted geodesics intersect at a tensor, a logical qubit is encoded on that tensor; otherwise, the gauge degrees of freedom of that tensor are fixed. This procedure ensures that each logical qubit is supported independently by four physical qubits at the boundary. This constant rate HTN code can be referred to as the "geodesic type" HTN. Additionally, this work also reduced the rate on the basis of the "geodesic type" HTN by projecting half of the logical qubits onto the eigenstate of a specific logical operation (gauge fixed), creating the "geodesic 1/2 type"

HTN. We will investigate the existence of thresholds under quantum erasure channels for these in subsequent studies.

3.8.2. Study of HTN Code Distances

Prior to this work, there was no research on the code distances of the [5,4] HTN codes. This work uses an operator push program to obtain the complete stabilizer group generators and logical operators for HTN codes at various sizes. Then, using a code distance calculator based on integer optimization, it calculates the minimum weight representation of each logical operator to determine the code distances, and further research and discussion are conducted.

3.8.3. Study of the Threshold of Zero Rate HTN under Pauli Errors

This work places the gauge-fixed zero-rate HTN codes under depolarizing and pure Pauli noise channels, using a variable-weight integer optimization decoder to determine their thresholds.

3.9. Threshold Study of Holographic Reed Muller Code

Prior to this work, there had been no threshold studies on holographic Reed Muller codes. Therefore, this work only conducts some basic research on holographic Reed Muller codes, including: obtaining the complete stabilizer generators and logical operators for zero rate holographic Reed Muller codes from $R = 0$ to $R = 4$ through an operator push program; placing zero rate holographic Reed Muller codes under a quantum erasure channel to study their thresholds.

3.10. Heterogeneous Holographic Codes Threshold Study

Study of heterogeneous holographic codes permits the use of different seed tensors within a homogeneous holographic framework. By incorporating various tensors into a homogeneous holographic code, a heterogeneous holographic code can be obtained. For heterogeneous codes, since different codes possess different transversal gate operations, it is expected that the number of transversal gate operations can be expanded (with an extra round of error correction [23]), extending to heterogeneous concatenated codes. Previous work [23–25] has demonstrated that concatenated codes, achieved through tree-like concatenation of Steane codes with Reed Muller codes that possess a transversal T gate, have thresholds and can execute transversal T gates, albeit with the necessity of error correction after each transversal T gate operation. Therefore, studying the thresholds of heterogeneous holographic codes under erasure error channel is meaningful.

3.10.1. The Heterogeneous Codes Studied and Their Construction

This study constructs heterogeneous codes primarily using three types of codes as seed tensors: HaPPY code, Steane code, and Reed Muller code.

1. **HaPPY code + Steane code**

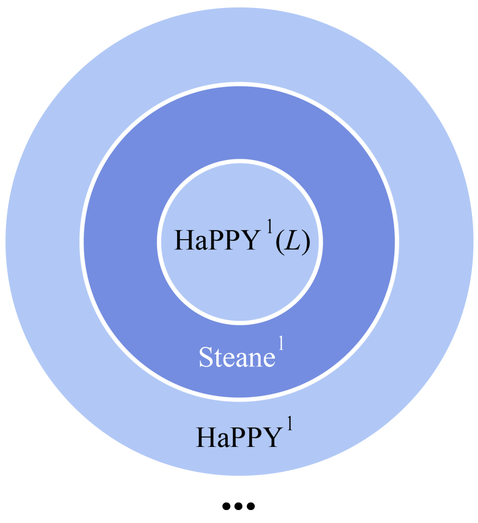


Figure 3.10: HaPPY+Steane

For the first attempt at working with heterogeneous holographic codes in this study, the choice to combine HaPPY code and Steane code is not aimed at expanding the number of transversal gate operations, but rather to use this as a case study to preliminarily understand the threshold behavior of heterogeneous holographic codes under quantum erasure noise. For this heterogeneous holographic code, we have placed HaPPY code in odd layers and Steane code in even layers, forming a max rate heterogeneous HaPPY+Steane holographic code.

2. **HaPPY code + Reed Muller code**

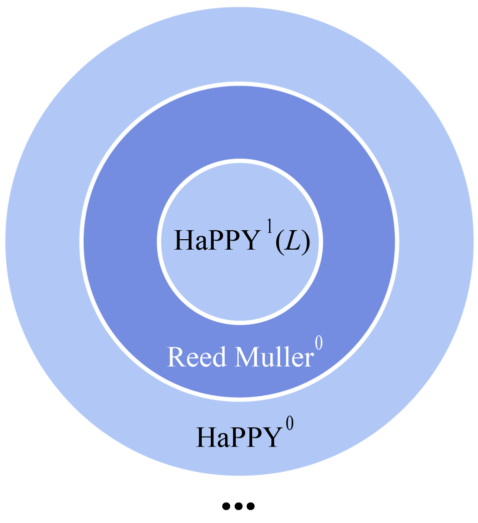


Figure 3.11: HaPPY+qRM

The second attempt in this study at working with heterogeneous holographic codes involves combining HaPPY code and Reed Muller code to form a heterogeneous holographic code. In this configuration, HaPPY code is placed in odd layers, and Reed Muller code is placed in even layers, with the structure assembled according to a $q = 4$ edge inflation stacking method to create the heterogeneous holographic code.

3. **Steane code + Reed Muller code**

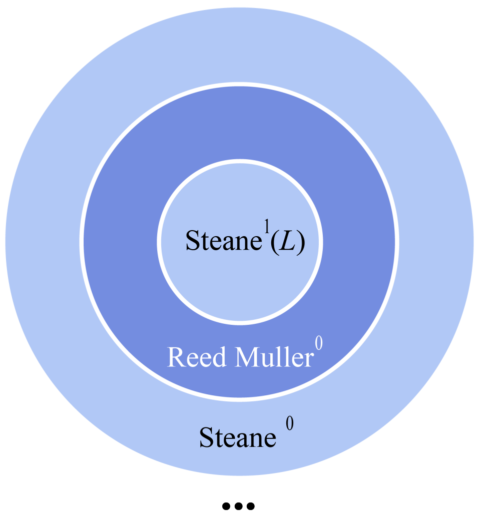


Figure 3.12: Steane+qRM

The third attempt in this study at working with heterogeneous holographic codes involves combining Steane code and Reed Muller code to form a heterogeneous holographic code. In this arrangement, Steane code is placed in odd layers, and Reed Muller code is placed in even layers. Similarly to the previous configurations, this heterogeneous holographic code is assembled using a $q = 4$ edge inflation stacking method.

4

Novel Results

4.1. Holographic Codes under Biased Noise Channel

This work investigated the thresholds of the HaPPY code and the holographic Steane code under different biased noise conditions (r_x, r_y, r_z) using a tensor network decoder. And, in the $r_x = r_y$ direction, the bias η is defined as $\eta = r_z/(r_x + r_y)$. The thresholds of the HaPPY code and the Steane code under different biases η in this direction are compared with the Hashing Bound.

4.1.1. HaPPY Code under Biased Noise

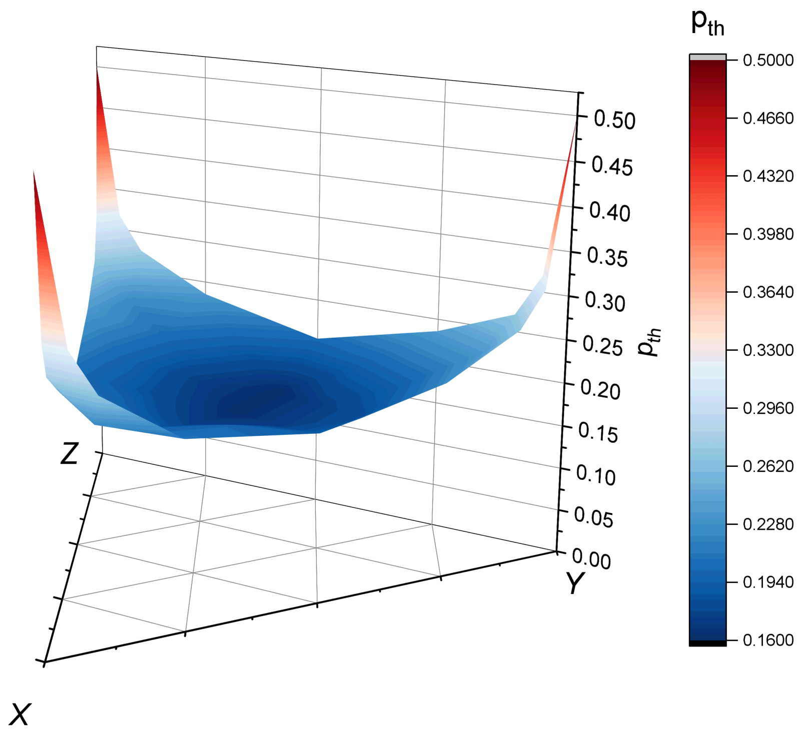


Figure 4.1: Thresholds of the zero-rate HaPPY code under biased noise.

The threshold performance of the Zero rate HaPPY code under biased noise is illustrated in the figure 4.1. It has a threshold of about 16.3% under the depolarizing channel [95], and as the bias increases, its threshold also gradually increases (this threshold figure agrees with previous work on the same code) [95]. Under pure Pauli noise channels, the Zero rate HaPPY code can achieve a 50% threshold, consistent with its threshold performance under quantum erasure error channels.

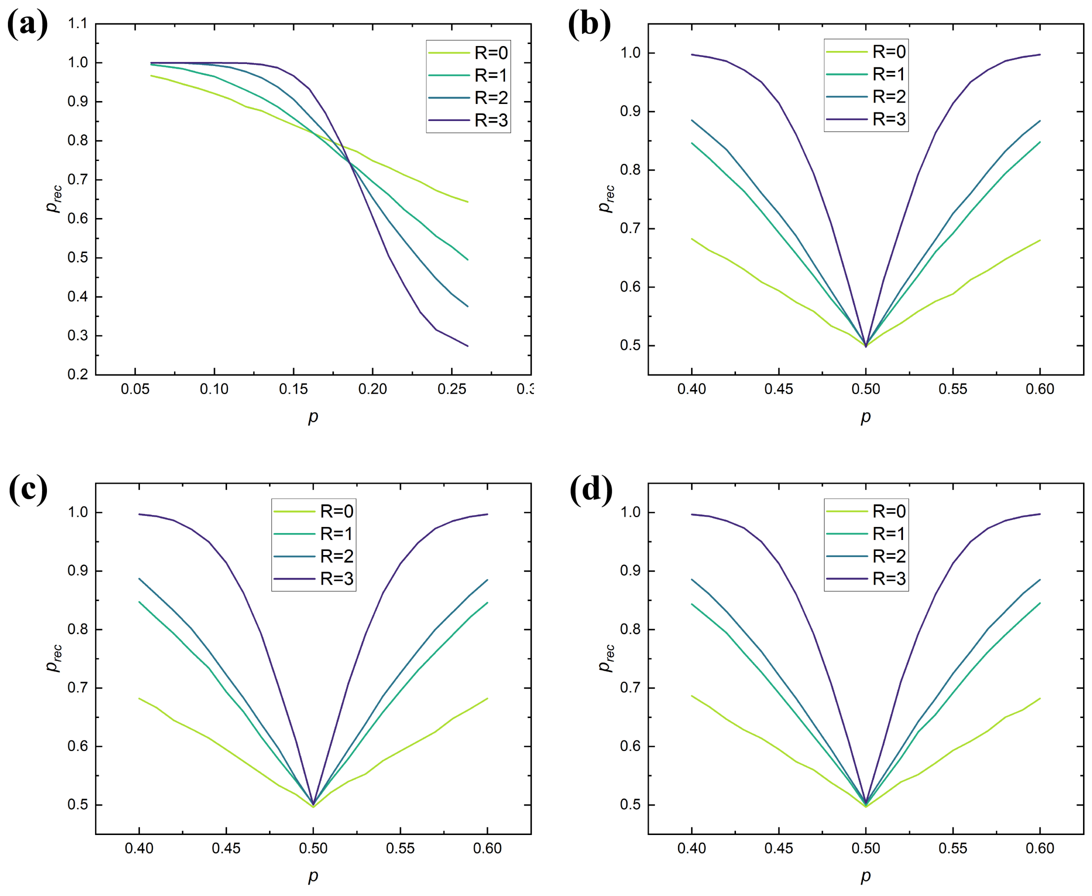


Figure 4.2: (a) The recovery rate curve of the zero rate HaPPY code under the depolarizing noise channel. (b) The recovery rate curve of the zero rate HaPPY code under pure X Pauli noise, with a threshold of $\sim 50\%$. (c) The recovery rate curve of the zero rate HaPPY code under pure Y Pauli noise, with a threshold of $\sim 50\%$. (d) The recovery rate curve of the zero rate HaPPY code under pure Z Pauli noise, with a threshold of $\sim 50\%$.

Figure 4.2 shows the recoverability versus error rate curves for the Zero rate HaPPY code under pure Pauli noise channels. It is observed that when using the tensor network decoder, the Zero rate HaPPY code under pure Pauli errors maintains a minimum recoverability of 50%. This minimum is only reached when the error rate is exactly 50%, which aligns with the quantum no-cloning theorem. Additionally, the graphs clearly illustrate that the Zero rate HaPPY code has a 50% threshold under pure Pauli noise.

4.1.2. Holographic Steane Code under Biased Noise

The threshold performance of the holographic Steane code under biased noise channels is depicted in the figure 4.3. Under the depolarizing channel, the threshold of the holographic Steane code is approximately 19%. As the bias increases towards pure Pauli noise, the threshold gradually decreases, with the threshold under pure Pauli noise being about 10%. Interestingly, if the bias increases not towards a single type of Pauli noise but towards a mix of two types, such as $(r_x, r_y, r_z) = (0.5, 0, 0.5)$, the threshold of the holographic Steane code gradually increases with the bias, reaching a maximum of about 21.9%.

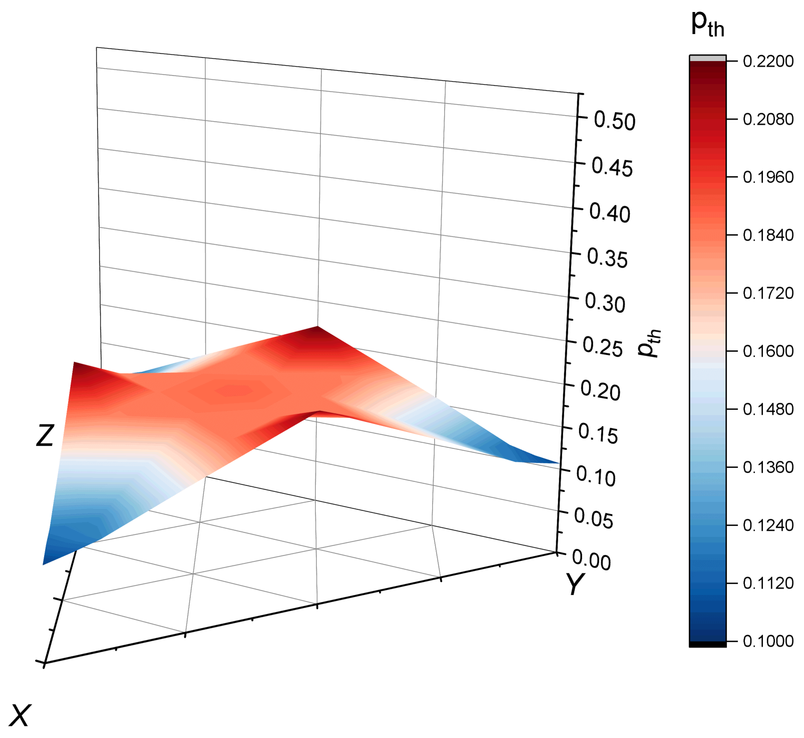


Figure 4.3: Threshold of the holographic Steane code under biased noise.

The reason for the threshold decrease of the holographic Steane code with increasing bias is understandable. The holographic Steane code is a self-dual CSS code, whose stabilizer group can be symmetrically divided into S_x and S_z. S_x contains only X operators, while S_z contains only Z operators, and the stabilizer matrices for S_x and S_z are identical within their respective subspaces. Under pure X or Z errors, only half of the stabilizers could potentially be triggered, while the other half provides no useful information, hence the anticipated drop in threshold. Under pure Y errors, S_x and S_z would give the same syndrome, thus beyond informing the decoder that this is a Y error, no additional information is provided, making it understandable why the threshold under pure Y noise is as low as under pure X or Z noise.

4.1.3. Comparison to the Hashing Bound

We fix the direction of increasing bias from the depolarizing point $(r_x, r_y, r_z) = (1/3, 1/3, 1/3)$ to the pure Z error bias point $(r_x, r_y, r_z) = (0, 0, 1)$. Thus, our bias ratio η is defined as $\eta = r_z/(r_x + r_y)$, where $r_x = r_y$. We then test the thresholds of the HaPPY code and the holographic Steane code at biases of $\eta = 1/2, 1, 3, 10, 30, 100, 300, 1000, \infty$ and compare these thresholds with those achievable by random coding according to the Hashing Bound. [1, 53, 99–101]. The results are as shown in figure 4.4.

From the figure, it can be observed that the threshold of the zero rate HaPPY code surpasses the hashing bound starting at $\eta = 30$, whereas, in contrast, the threshold of the holographic Steane code decreases as η increases.

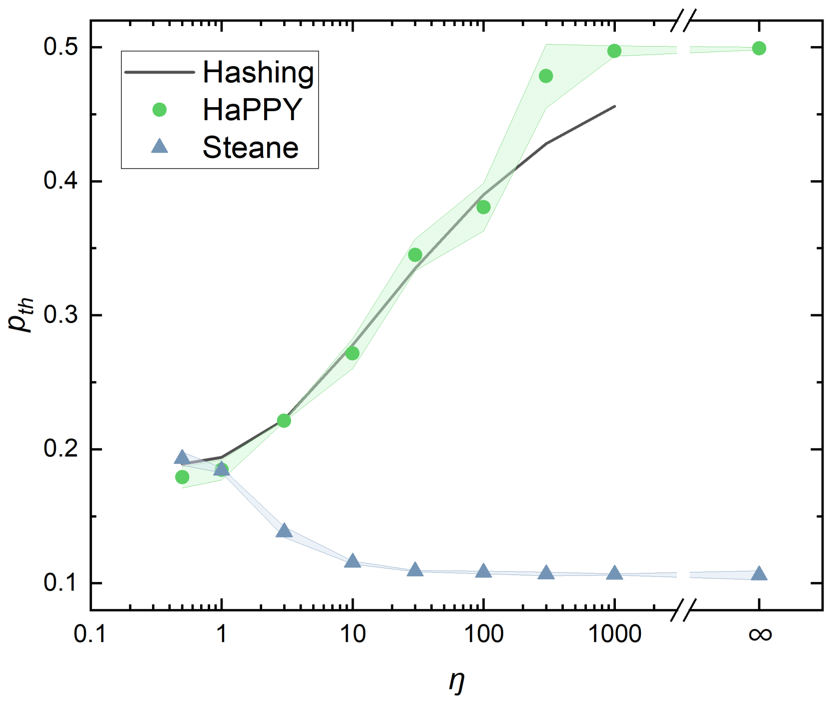


Figure 4.4: Comparison of thresholds for the HaPPY code and holographic Steane code under different biases η with the hashing bound.

4.2. Distance and Threshold of HTN Codes

This work has obtained the quantum erasure threshold results for the {5,4} HTN code under various rates and different methods of gauge fixing, as well as the threshold results for the zero rate HTN code under Z gauge fixing in depolarizing channels and pure Pauli noise channels.

4.2.1. Distance of HTN Codes

This work used integer optimizers to calculate distances, obtaining the bit distance and word distance of the central tensor qubit of the {5,4} HTN code (Max Rate), as well as the distances for the X gauge fixed zero rate HTN code, Z gauge fixed zero rate HTN code, and Y gauge fixed zero rate HTN code.

		Max Rate HTN central qubit (bit)		**Max Rate HTN central qubit (word)**	
L	n	d_z	d_x	d_z	d_x
0	4	2	2	2	2
1	20	6	2	2	2
2	76	14	6	2	2
3	284	36	16	2	2
4	1060	92	42	2	2

Table 4.1: Distances of Max Rate HTN

		Zero Rate X Gauge Fixed		Zero Rate Z Gauge Fixed		Zero Rate Y Gauge Fixed	
L	*n*	d_z	d_x	d_z	d_x	d_z	d_x
0	4	2	2	2	2	2	2
1	20	2	2	6	2	6	2
2	76	2	2	10	6	9	6
3	284	2	2	14	14	14	14
4	1060	2	2	30	26	32	16

Table 4.2: Distances of Gauge Fixed Zero Rate HTN

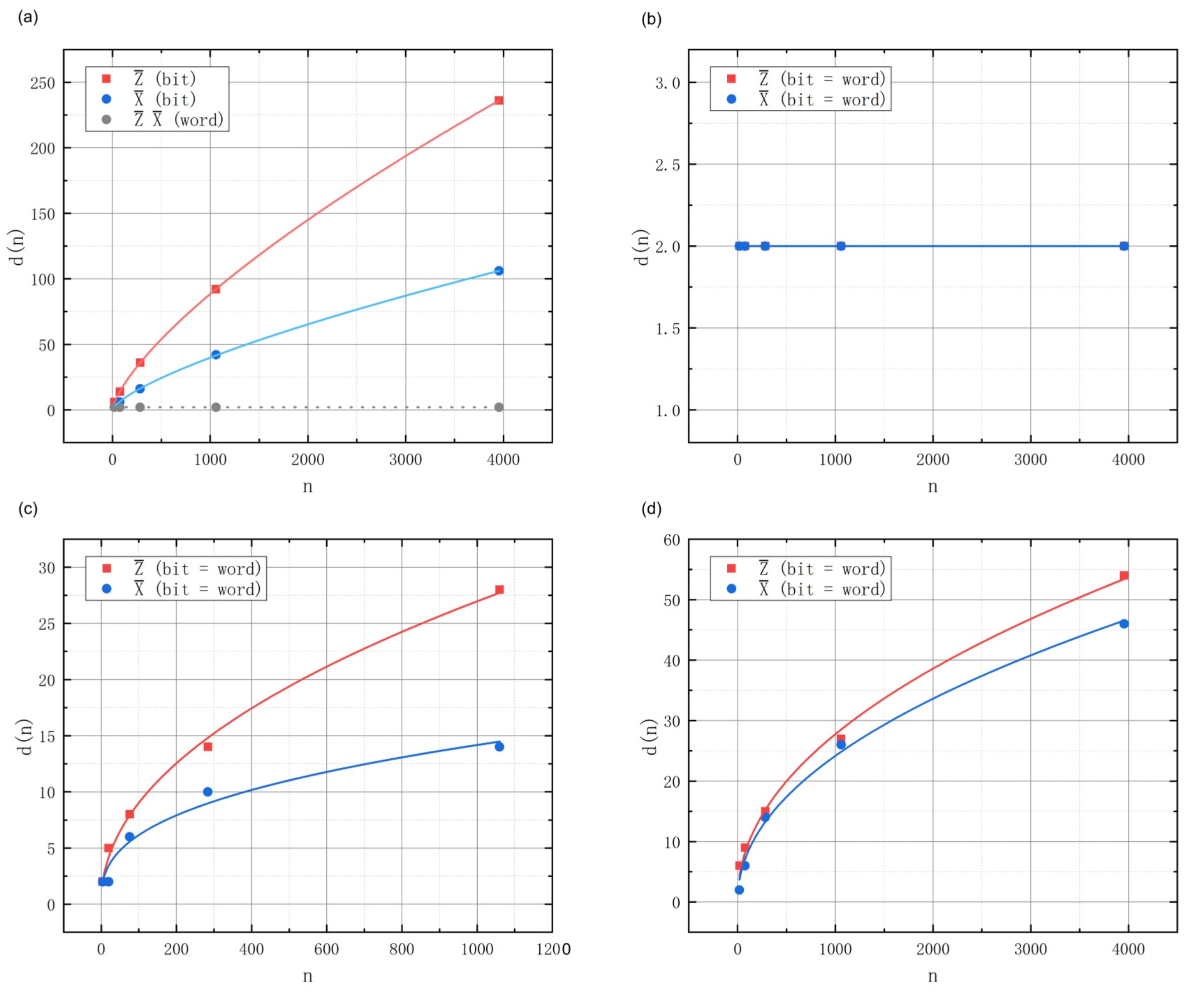


Figure 4.5: Distances in HTN codes are as follows: (a) Under max rate HTN, the bit distance of the central logical qubit (the weight of the minimum weight operator needed to change the state of the central logical qubit while keeping other logical qubits unchanged) increases with n as $d(n) \approx n^{0.558}$ and $\approx n^{0.658}$, while the word distance (the weight of the minimum weight operator needed to change the state of the central logical qubit) remains constantly at 2. (b) For the X gauge fixed zero rate HTN code, obtained after fixing the X gauge of all tensors except the central tensor, the distance remains constant at 2. (c) For the Y gauge fixed zero rate HTN code, obtained after fixing the Y gauge of all tensors except the central tensor, the distance is $d(n) \approx n^{0.385}$ and $\approx n^{0.476}$. (d) For the Z gauge fixed zero rate HTN code, obtained after fixing the Z gauge of all tensors except the central tensor, the distance is $d(n) \approx n^{0.463}$ and $\approx n^{0.48}$.

4.2.2. Erasure Threshold of Zero Rate Gauge-Fixed HTN Codes

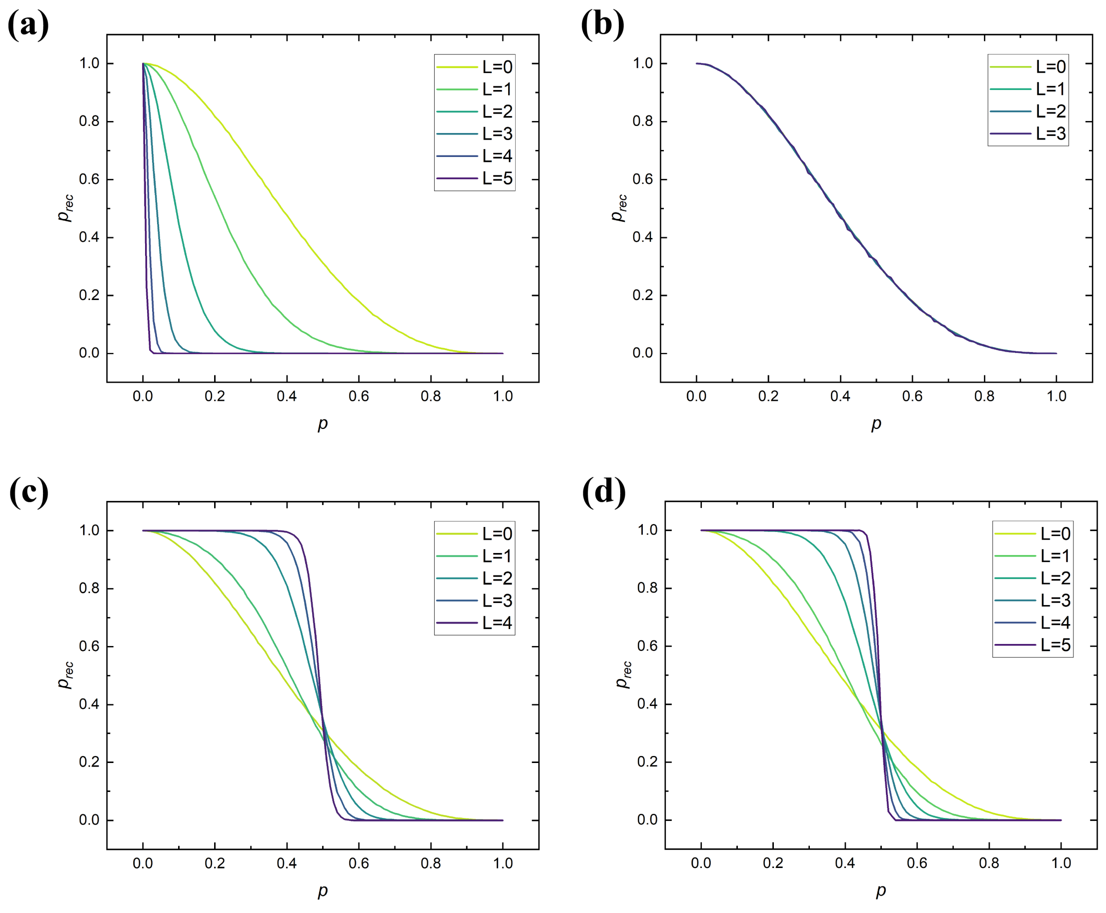


Figure 4.6: (a) The recovery rate of the logical qubit for the central tensor in a max-rate HTN code varies with the error rate p. It can be observed that there is no threshold. (b) The recovery rate of the logical qubit for an X gauge-fixed zero-rate HTN code (where all X gauges except the central tensor are fixed) varies with the error rate p, and there is no threshold. (c) The recovery rate of the logical qubit for a Y gauge-fixed zero-rate HTN code (where all Y gauges except the central tensor are fixed) varies with the error rate p, and the threshold is $p_{th} = 49.83\% \pm 0.09\%$. (d) The recovery rate of the logical qubit for a Z gauge-fixed zero-rate HTN code (where all Z gauges except the central tensor are fixed) varies with the error rate p, and the threshold is $p_{th} = 49.95\% \pm 0.01\%$.

First, as a comparison, Fig.4.6 (a) shows that the central logical qubit of a max-rate HTN code does not have a threshold. Next, for three different gauge-fixed zero-rate HTN codes, the Y gauge-fixed and Z gauge-fixed zero-rate HTN codes have a saturation erasure threshold of approximately 50%, while the X gauge-fixed zero-rate HTN code does not have a threshold. It is worth noting that the recovery rate curve does not change with the increase in the code radius of the HTN code.

The erasure thresholds for different zero-rate HTN codes are summarized in the table 4.3 below:

Table 4.3: Erasure Threshold of Zero Rate Gauge-Fixed HTN Codes

Code Type	**Erasure Threshold**	**Code Rate**
X gauge-fixed HTN code	No threshold	$\sim 0\%$
Y gauge-fixed HTN code	$\sim 50\%(49.83\% \pm 0.09\%)$	$\sim 0\%$
Z gauge-fixed HTN code	$\sim 50\%(49.95\% \pm 0.01\%)$	$\sim 0\%$

4.2.3. Erasure Threshold of Constant Rate Gauge-Fixed HTN Codes

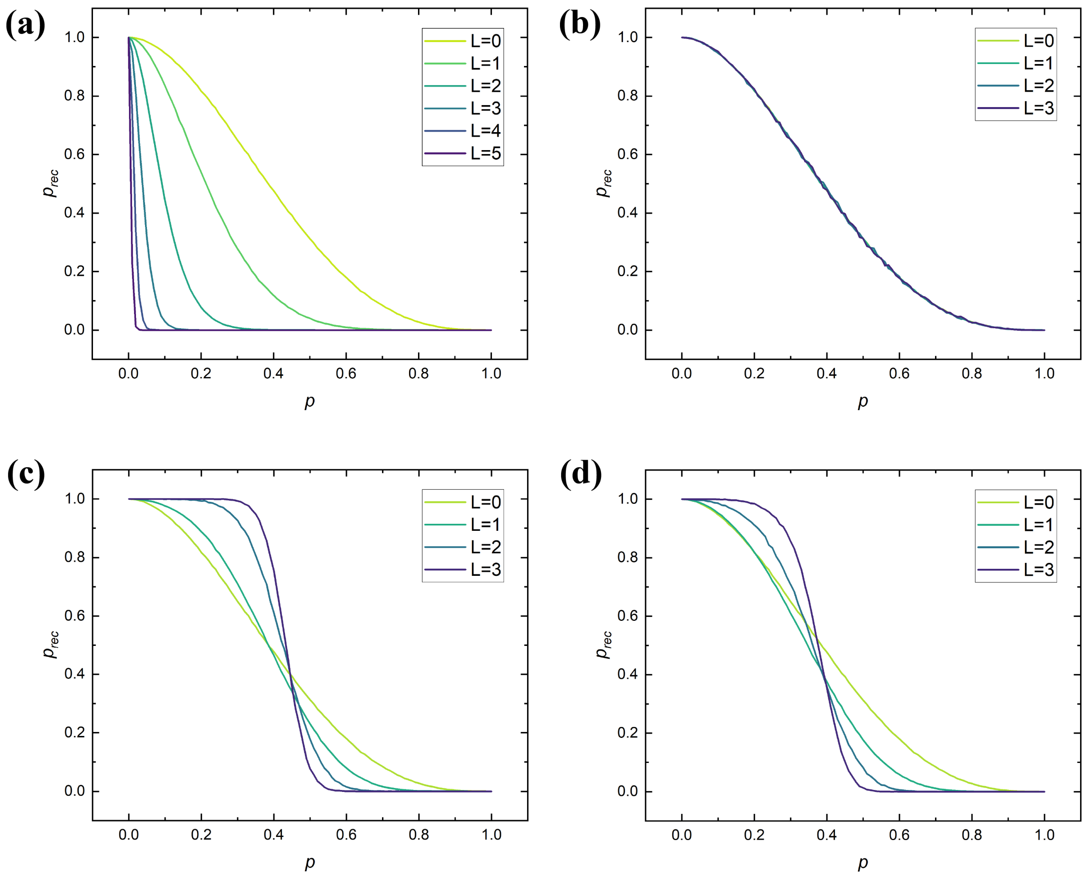


Figure 4.7: (a) The recovery rate of the logical qubit for the central tensor in a max-rate HTN code varies with the error rate p, showing that there is no threshold. (b) The recovery rate of the logical qubit for the central tensor in an X gauge-fixed geo-half HTN code varies with the error rate p, showing that there is no threshold. (c) The recovery rate of the logical qubit for the central tensor in a Y gauge-fixed geo-half HTN code varies with the error rate p, with a threshold of $p_{th} = 44.02\% \pm 2.89\%$. (d) The recovery rate of the logical qubit for the central tensor in a Z gauge-fixed geo-half HTN code varies with the error rate p, with a threshold of $p_{th} = 39.24\% \pm 0.60\%$.

As a constant rate HTN code, the geo-half HTN code exhibits different threshold behaviors under different gauge fixings. Fig4.7 (b) shows the recovery rate p_{rec} curve of the logical qubit for the central tensor in an X gauge-fixed geo-half HTN code under erasure errors, indicating that it does not change with the code radius. Fig4.7 (c) and Fig4.7 (d) show the Y gauge-fixed and Z gauge-fixed geo-half HTN codes, respectively, with erasure thresholds of $p^Y_{th} = 44.02\% \pm 2.89\%$ and $p^Z_{th} = 39.24\% \pm 0.60\%$.

Table 4.4: Erasure Threshold of Constant Rate Gauge-Fixed HTN Codes

Code Type	Erasure Threshold	Code Rate
X gauge-fixed geo-half HTN code	No threshold	$\sim 12.5\%$
Y gauge-fixed geo-half HTN code	$44.02\% \pm 2.89\%$	$\sim 12.5\%$
Z gauge-fixed geo-half HTN code	$39.24\% \pm 0.60\%$	$\sim 12.5\%$

4.2.4. Threshold of Zero Rate Z Gauge-Fixed HTN Codes under Pauli Noise Channel

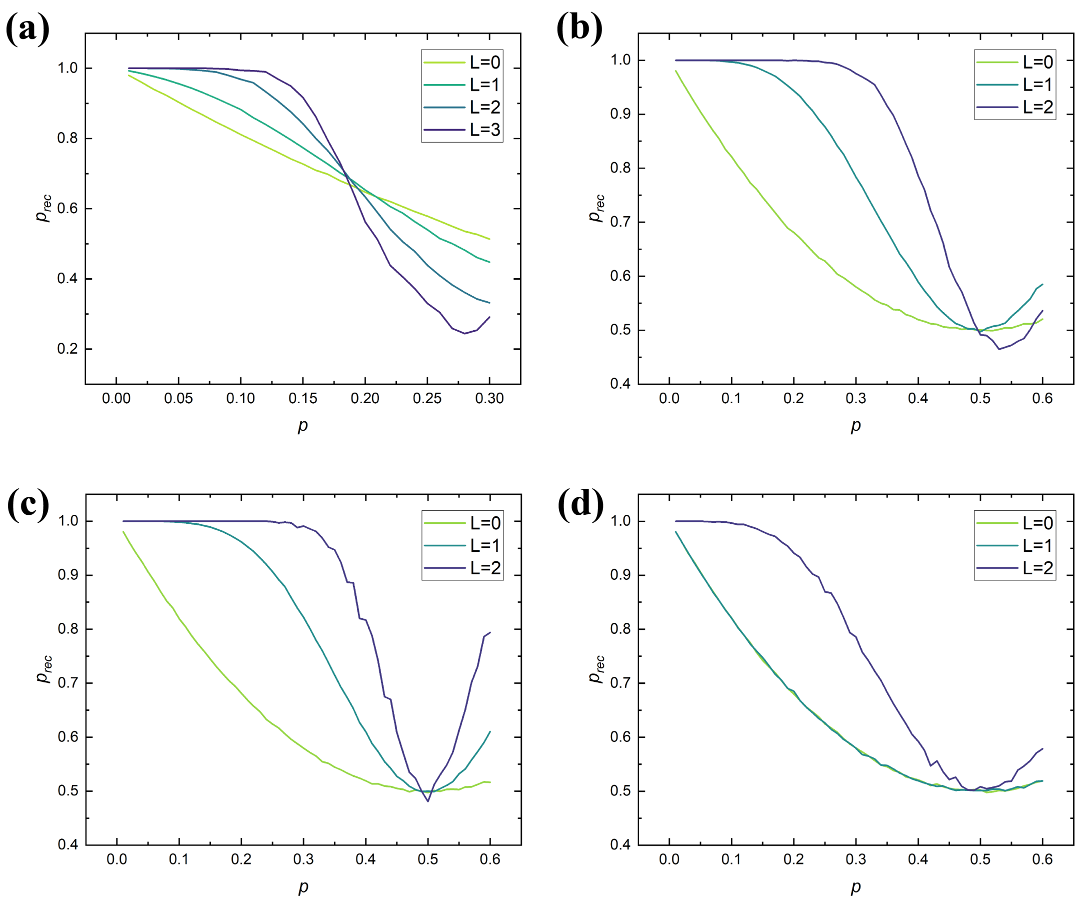


Figure 4.8: (a) The recovery rate curve of the Z gauge-fixed zero-rate HTN code under depolarizing noise, with a threshold of $p_{th} = 19.10\% \pm 0.94\%$. (b) The recovery rate curve of the Z gauge-fixed zero-rate HTN code under pure X noise, with a preliminary estimated threshold of $\sim 50\%$. (c) The recovery rate curve of the Z gauge-fixed zero-rate HTN code under pure Y noise, with a preliminary estimated threshold of $\sim 50\%$. (d) The recovery rate curve of the Z gauge-fixed zero-rate HTN code under pure Z noise, with a preliminary estimated threshold of $\sim 50\%$.

In this work, three types of gauge-fixed zero-rate HTN codes (X gauge-fixed, Y gauge-fixed, Z gauge-fixed) were subjected to depolarizing and pure Pauli noise (pure X, pure Y, pure Z) to study their threshold behaviors. It was found that the X gauge-fixed and Y gauge-fixed zero-rate HTN codes do not have thresholds under Pauli noise. Only the Z gauge-fixed zero-rate HTN code exhibits a threshold.

Since the HTN code's tensor network includes Hadamard gates, a tensor network decoder has not yet been adapted for this purpose. Therefore, Pauli error decoding for HTN was performed using a variable-weight integer optimization decoder.

As shown in Fig 4.8 (a), the Z gauge-fixed zero-rate HTN code has a threshold of $p_{th} = 19.10\% \pm 0.94\%$ under depolarizing noise. Additionally, as shown in Fig 4.8 (b), (c), and (d), due to the performance limitations of the variable integer optimization decoder, only the recovery rate curves for HTN codes with up to three different radii can be obtained. Therefore, it can only be preliminarily concluded that the Z gauge-fixed zero-rate HTN code has an approximate threshold of 50% under pure X, pure Y, and pure Z Pauli noise.

A notable point of discussion is the recovery rate curve of the Z gauge-fixed zero-rate HTN code under pure Z noise. It can be observed that the recovery rates for the Z gauge-fixed zero-rate HTN codes with sizes $L = 0$ and $L = 1$ are essentially the same under pure Z noise. The current qualitative explanation for this phenomenon is that the fixed Z gauge at the $L = 1$ layer does not anti-commute with Z errors, thus it cannot provide effective error correction information. However, for the HTN code with a radius of $L = 2$, the Z gauge at the $L = 1$ layer has undergone a Hadamard gate and starts to anti-commute with Z errors, leading to an improvement in the recovery rate.

The results are summarized as follows:

Table 4.5: Threshold of Zero Rate Z Gauge-Fixed HTN Codes under Pauli Noise Channel

Code Type	**Noise Type**	**Threshold**
Z gauge-fixed geo-half HTN code	Depolarizing noise	$19.10\% \pm 0.94\%$
Z gauge-fixed geo-half HTN code	Pure X noise	Preliminarily estimated $\sim 50\%$
Z gauge-fixed geo-half HTN code	Pure Y noise	Preliminarily estimated $\sim 50\%$
Z gauge-fixed geo-half HTN code	Pure Z noise	Preliminarily estimated $\sim 50\%$

4.3. Erasure Threshold of Holographic Reed Muller Codes

The seed tensor of the holographic Reed Muller code has 16 legs, one of which is a logical leg. This determines that the operator push workload for the holographic Reed Muller will be substantial. This work used an automatic operator push program to obtain the stabilizer group generators and complete logical operators for the holographic Reed Muller code up to $R = 3$. The holographic Reed Muller codes for $R = 0$, $R = 1$, and $R = 2$ were then tested under a quantum erasure noise channel to determine their thresholds. The results are shown in the following figure 4.9:

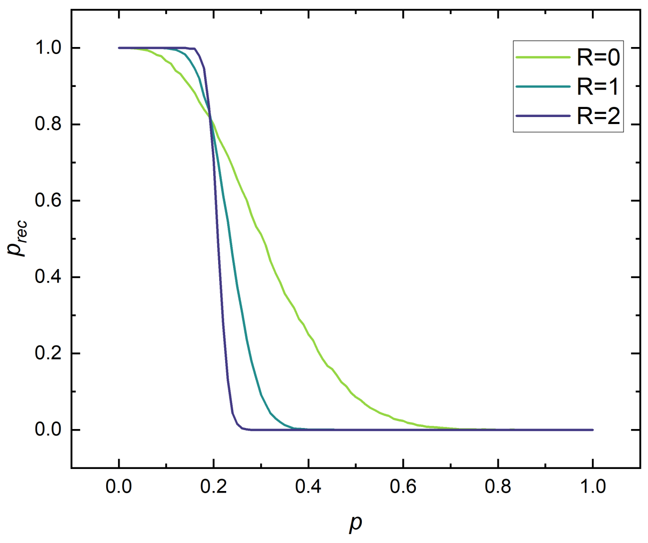


Figure 4.9: Erasure Threshold of Holographic Reed Muller Codes

We preliminarily determined that the threshold of the holographic Reed Muller code under a quantum erasure channel is approximately $19.24\% \pm 0.12\%$.

4.4. Threshold of Heterogeneous Holographic Codes

In this work, we placed several heterogeneous holographic quantum error-correcting codes under a quantum erasure channel to observe how their threshold behaviors compare and contrast with those of homogeneous holographic quantum error-correcting codes. We found that for holographic codes with tensors arranged heterogeneously according to the parity of the layers, the thresholds are no longer unique but instead display distinct even and odd thresholds.

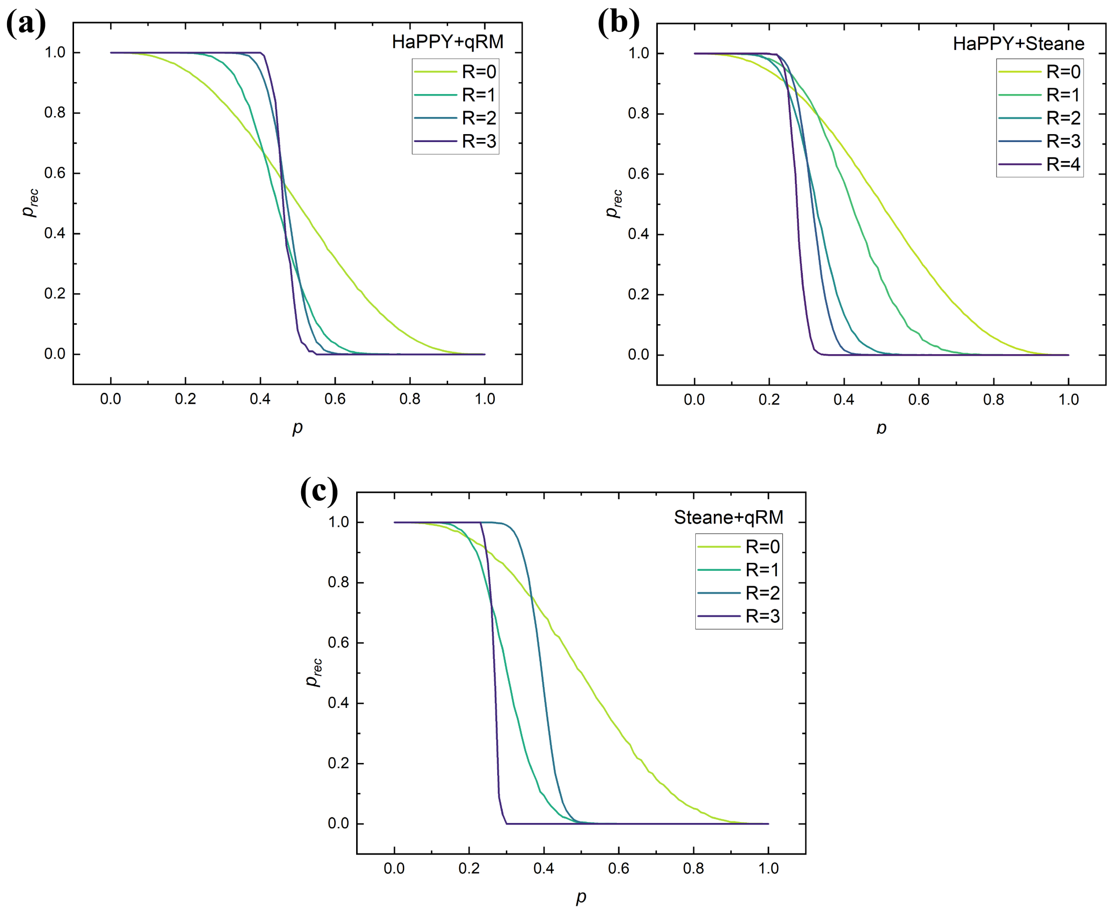


Figure 4.10: (a) The recovery rate curve for the HaPPY+Reed Muller heterogeneous holographic error-correcting code, where even layers contain HaPPY code tensors and odd layers contain Reed Muller code tensors, indicates a preliminary threshold with an odd-even effect: $p_{th}^{even} = 46.38\%$, $p_{th}^{odd} = 46.78\%$. (b) The recovery rate curve for the HaPPY+Steane heterogeneous holographic error-correcting code, where even layers contain HaPPY code tensors and odd layers contain Steane code tensors, indicates a preliminary threshold with an odd-even effect: $p_{th}^{even} = 24.64\%$, $p_{th}^{odd} = 25.68\%$. (c) The recovery rate curve for the Steane+Reed Muller heterogeneous holographic error-correcting code, where even layers contain Steane code tensors and odd layers contain Reed Muller code tensors, indicates a preliminary threshold with an odd-even effect: $p_{th}^{even} = 25.95\%$, $p_{th}^{odd} = 36.65\%$.

For the three types of heterogeneous holographic codes, the seed tensors for odd and even layers are of different types (HaPPY, Steane, Reed Muller). Therefore, it is intuitive that they have two different thresholds for odd and even layers. Monte Carlo experiments have preliminarily verified this, showing that all these heterogeneous holographic codes exhibit two distinct thresholds for odd and even layers. Due to limited computational power, the heterogeneous holographic codes involving Reed Muller could not be extended to sufficiently high radii, so only preliminary conclusions can be given here.

5 Conclusion

This thesis has studied the principles and error-correction properties of holographic quantum error correction codes. We have made here several notable contributions to the larger error-correction community:

- We have proposed and developed an automated holographic code operator generation program based on the quantum lego operator push [5], and in conjunction with various decoders [2, 95], has studied the thresholds of holographic codes under several different noise channels, notably erasure, depolarizing, and biased-noise channels.
- We executed a full biased-noise study of the zero-rate HaPPY and holographic Steane codes using the tensor network deocder. We confirmed that the zero-rate HaPPY code exhibits a maximum threshold of 50% under pure Pauli noise and that its threshold exceeds the Hashing bound under conditions of high bias, which is very similar to the XZZX and XY surface codes [1, 99]. In contrast to the HaPPY code, the threshold of the holographic Steane code decreases under high bias, and a straightforward explanation for this behavior is intuitively provided.
- Furthermore, the depolarizing noise and erasure thresholds for the HTN code have been determined, showing very promising resilience of the code against both error channels, roughly aligning with the known theoretical and numerical estimates for Toric and traditional CSS surface codes [102–104]. These codes also support weight-2 transversal logical operations in the logical-X gauge [92].
- Finally, we have developed a range of heterogeneous holographic codes to explore the creation of larger fault-tolerant logical gate sets. Due to the substantial size of the check and logical operators, we conducted our initial studies using the erasure decoder. While these results are preliminary, they intriguingly suggest a high resilience to erasure that predominantly depends on whether the concatenation layer tested is even or odd.

There are many potential future research directions:

1. *Extension for constant-rate holographic codes.* It would of course be useful and interesting to investigate whether constructions of constant-rate holographic codes can exceed or attain the Hashing bound.
2. *Biased-noise explorations of the HTN code, as well as other holographic codes.* This thesis did not give enough time to fully investigate other holographic code constructions and their resilience to biased noise; principle among these lies the HTN code, for which a specialized tensor network decoder will need to be designed in order to perform constractions with the Hadamard gates on the edges, as well as for storing the very large stabilizers in the vertex-inflated HTN code.

In conclusion, we have demonstrated here that holographic quantum codes indeed exhibit many useful and desirable properties for practical quantum computing However, much more exploration is needed to see precisely in what architectural conditions a holographic quantum code could prove useful.

References

[1] J Pablo Bonilla Ataides et al. “The XZZX surface code”. In: *Nature communications* 12.1 (2021), p. 2172.

[2] Terry Farrelly et al. “Parallel decoding of multiple logical qubits in tensor-network codes”. In: *Physical Review A* 105.5 (2022), p. 052446.

[3] Fernando Pastawski et al. “Holographic quantum error-correcting codes: Toy models for the bulk/boundary correspondence”. In: *Journal of High Energy Physics* 2015.6 (2015), pp. 1–55.

[4] Matthew Steinberg, Sebastian Feld, and Alexander Jahn. “Holographic codes from hyperinvariant tensor networks”. In: *Nature Communications* 14.1 (2023), p. 7314.

[5] ChunJun Cao and Brad Lackey. “Quantum lego: Building quantum error correction codes from tensor networks”. In: *PRX Quantum* 3.2 (2022), p. 020332.

[6] Peter W Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[7] Peter W Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332.

[8] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[9] Richard Feynman. “There’s plenty of room at the bottom.” In: *Resonance: Journal of Science Education* 16.9 (2011).

[10] Richard P Feynman. “Simulating physics with computers”. In: *Feynman and computation*. CRC Press, 2018, pp. 133–153.

[11] Richard Feynman. “Quantum mechanical computers”. In: *Optics news* 11.2 (1985), pp. 11–20.

[12] Davide Castelvecchi. “Quantum computers ready to leap out of the lab in 2017”. In: *Nature* 541.7635 (2017).

[13] Colm A Ryan et al. “Hardware for dynamic quantum computing”. In: *Review of Scientific Instruments* 88.10 (2017).

[14] Lieven MK Vandersypen and Mark A Eriksson. “Quantum computing with semiconductor spins”. In: *Physics Today* 72.8 (2019), pp. 38–45.

[15] Medina Bandic, Sebastian Feld, and Carmen G Almudever. “Full-stack quantum computing systems in the NISQ era: algorithm-driven and hardware-aware compilation techniques”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 1–6.

[16] Salonik Resch and Ulya R Karpuzcu. “Quantum computing: an overview across the system stack”. In: *arXiv preprint arXiv:1905.07240* (2019).

[17] Michael Brooks. “Beyond quantum supremacy: the hunt for useful quantum computers”. In: *Nature* 574.7776 (2019), pp. 19–22.

[18] John Preskill. “Quantum computing in the NISQ era and beyond”. In: *Quantum* 2 (2018), p. 79.

[19] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[20] Menno Veldhorst et al. “A two-qubit logic gate in silicon”. In: *Nature* 526.7573 (2015), pp. 410–414.

[21] Daniel Gottesman. “An introduction to quantum error correction and fault-tolerant quantum computation”. In: *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*. Vol. 68. 2010, pp. 13–58.

[22] Juan Maldacena. “The large-N limit of superconformal field theories and supergravity”. In: *International journal of theoretical physics* 38.4 (1999), pp. 1113–1133.

[23] Tomas Jochym-O’Connor and Raymond Laflamme. “Using concatenated quantum codes for universal fault-tolerant quantum gates”. In: *Physical review letters* 112.1 (2014), p. 010505.

[24] Christopher Chamberland, Tomas Jochym-O’Connor, and Raymond Laflamme. “Overhead analysis of universal concatenated quantum codes”. In: *Physical Review A* 95.2 (2017), p. 022313.

[25] Christopher Chamberland, Tomas Jochym-O’Connor, and Raymond Laflamme. “Thresholds for universal concatenated quantum codes”. In: *Physical review letters* 117.1 (2016), p. 010501.

[26] Edward Witten. “Anti de Sitter space and holography”. In: *arXiv preprint hep-th/9802150* (1998).

[27] Martin Bossert. *Channel coding for telecommunications*. John Wiley & Sons, Inc., 1999.

[28] Rodger Ziemer and William H Tranter. *Principles of communications: system modulation and noise*. John Wiley & Sons, 2006.

[29] Richard W Hamming. “Error detecting and error correcting codes”. In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.

[30] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Vol. 16. Elsevier, 1977.

[31] Dave K Kythe and Prem K Kythe. *Algebraic and stochastic coding theory*. CRC Press Boca Raton (FL), 2012.

[32] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

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

[34] Amin Shokrollahi. “LDPC codes: An introduction”. In: *Coding, cryptography and combinatorics*. Springer. 2004, pp. 85–110.

[35] Larry Hardesty. “Explained: Gallager codes”. In: *MIT News* (2010).

[36] Robert Gallager. “Low-density parity-check codes”. In: *IRE Transactions on information theory* 8.1 (1962), pp. 21–28.

[37] John Watrous. *Lecture 17: General quantum errors; CSS codes*. 2006. url: `https://cs.uwaterloo.ca/~watrous/QC-notes/QC-notes.17.pdf`.

[38] Barbara M Terhal. “Quantum error correction for quantum memories”. In: *Reviews of Modern Physics* 87.2 (2015), p. 307.

[39] Karl Kraus. “Lecture notes in physics”. In: *States, Effects and Operations. Fundamental Notions of Quantum Theory* 190 (1983).

[40] Benjamin Schumacher. “Sending entanglement through noisy quantum channels”. In: *Physical Review A* 54.4 (1996), p. 2614.

[41] Charles H Bennett, David P DiVincenzo, and John A Smolin. “Capacities of quantum erasure channels”. In: *Physical Review Letters* 78.16 (1997), p. 3217.

[42] Markus Grassl, Th Beth, and Thomas Pellizzari. “Codes for the quantum erasure channel”. In: *Physical Review A* 56.1 (1997), p. 33.

[43] James L Park. “The concept of transition in quantum mechanics”. In: *Foundations of physics* 1.1 (1970), pp. 23–33.

[44] Asher Peres. “Reversible logic and quantum computers”. In: *Physical review A* 32.6 (1985), p. 3266.

[45] Emanuel Knill and Raymond Laflamme. “Concatenated quantum codes”. In: *arXiv preprint quant-ph/9608012* (1996).

[46] Benjamin Rahn, Andrew C Doherty, and Hideo Mabuchi. “Exact performance of concatenated quantum codes”. In: *Physical Review A* 66.3 (2002), p. 032304.

[47] John Preskill. “Reliable quantum computers”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (1998), pp. 385–410.

[48] Peter W Shor. “Scheme for reducing decoherence in quantum computer memory”. In: *Physical review A* 52.4 (1995), R2493.

[49] Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

[50] Bryan Eastin and Emanuel Knill. “Restrictions on transversal encoded quantum gate sets”. In: *Physical review letters* 102.11 (2009), p. 110502.

[51] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Physical Review A* 71.2 (2005), p. 022316.

[52] Sergey Bravyi and Jeongwan Haah. “Magic-state distillation with low overhead”. In: *Physical Review A* 86.5 (2012), p. 052329.

[53] Mark M Wilde. *Quantum information theory*. Cambridge university press, 2013.

[54] Román Orús. “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of physics* 349 (2014), pp. 117–158.

[55] Jacob Biamonte and Ville Bergholm. “Tensor networks in a nutshell”. In: *arXiv preprint arXiv:1708.00006* (2017).

[56] Pietro Silvi et al. “The Tensor Networks Anthology: Simulation techniques for many-body quantum lattice systems”. In: *SciPost Physics Lecture Notes* (2019), p. 008.

[57] Román Orús. “Advances on tensor network theory: symmetries, fermions, entanglement, and holography”. In: *The European Physical Journal B* 87 (2014), pp. 1–18.

[58] Jacob C Bridgeman and Christopher T Chubb. “Hand-waving and interpretive dance: an introductory course on tensor networks”. In: *Journal of physics A: Mathematical and theoretical* 50.22 (2017), p. 223001.

[59] Richard L Bishop and Samuel I Goldberg. *Tensor analysis on manifolds*. Courier Corporation, 2012.

[60] Hora□iu Năstase. *Introduction to the ADS/CFT Correspondence*. Cambridge University Press, 2015.

[61] Martin Ammon and Johanna Erdmenger. *Gauge/gravity duality: Foundations and applications*. Cambridge University Press, 2015.

[62] Ofer Aharony et al. “N = 6 superconformal Chern-Simons-matter theories, M2-branes and their gravity duals”. In: *Journal of High Energy Physics* 2008.10 (2008), p. 091.

[63] Alfonso V Ramallo. “Introduction to the AdS/CFT correspondence”. In: *Lectures on Particle Physics, Astrophysics and Cosmology: Proceedings of the Third IDPASC School, Santiago de Compostela, Spain, January 21–February 2, 2013*. Springer. 2015, pp. 411–474.

[64] Jacob D Bekenstein. “Black holes and entropy”. In: *Physical Review D* 7.8 (1973), p. 2333.

[65] Stephen W Hawking. “Particle creation by black holes”. In: *Communications in mathematical physics* 43.3 (1975), pp. 199–220.

[66] Ashoke Sen. “Extremal black holes and elementary string states”. In: *Modern Physics Letters A* 10.28 (1995), pp. 2081–2093.

[67] William A Bardeen et al. “Study of the longitudinal kink modes of the string”. In: *Physical Review D* 13.8 (1976), p. 2364.

[68] Itzhak Bars and Andrew J Hanson. “Quarks at the Ends of the String”. In: *Physical Review D* 13.6 (1976), p. 1744.

[69] Andrew Strominger and Cumrun Vafa. “Microscopic origin of the Bekenstein-Hawking entropy”. In: *Physics Letters B* 379.1-4 (1996), pp. 99–104.

[70] Alexander Jahn and Jens Eisert. “Holographic tensor network models and quantum error correction: a topical review”. In: *Quantum Science and Technology* 6.3 (2021), p. 033002.

[71] Sean A Hartnoll, Andrew Lucas, and Subir Sachdev. *Holographic quantum matter*. MIT press, 2018.

[72] Jan Zaanen et al. *Holographic duality in condensed matter physics*. Cambridge University Press, 2015.

[73] Shinsei Ryu and Tadashi Takayanagi. “Holographic derivation of entanglement entropy from the anti–de sitter space/conformal field theory correspondence”. In: *Physical review letters* 96.18 (2006), p. 181602.

[74] Horatiu Nastase. *String theory methods for condensed matter physics*. Cambridge University Press, 2017.

[75] Fernando Pastawski and John Preskill. “Code properties from holographic geometries”. In: *Physical Review X* 7.2 (2017), p. 021022.

[76] Xi Dong, Daniel Harlow, and Aron C Wall. “Reconstruction of bulk operators within the entanglement wedge in gauge-gravity duality”. In: *Physical review letters* 117.2 (2016), p. 021601.

[77] Ahmed Almheiri, Xi Dong, and Daniel Harlow. “Bulk locality and quantum error correction in AdS/CFT”. In: *Journal of High Energy Physics* 2015.4 (2015), pp. 1–34.

[78] Daniel Harlow. “Jerusalem lectures on black holes and quantum information”. In: *Reviews of Modern Physics* 88.1 (2016), p. 015002.

[79] E. Noether. “Invariante Variationsprobleme”. ger. In: *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* 1918 (1918), pp. 235–257. url: `http://eudml.org/doc/59024`.

[80] Bartłomiej Czech et al. “The gravity dual of a density matrix”. In: *Classical and Quantum Gravity* 29.15 (2012), p. 155009.

[81] Raphael Bousso, Stefan Leichenauer, and Vladimir Rosenhaus. “Light-sheets and AdS/CFT”. In: *Physical Review D* 86.4 (2012), p. 046009.

[82] Veronika E Hubeny and Mukund Rangamani. “Causal holographic information”. In: *Journal of High Energy Physics* 2012.6 (2012), pp. 1–35.

[83] Masamichi Miyaji, Tadashi Takayanagi, and Kento Watanabe. “From path integrals to tensor networks for the AdS/CFT correspondence”. In: *Physical Review D* 95.6 (2017), p. 066004.

[84] Ning Bao, ChunJun Cao, and Guanyu Zhu. “Deconfinement and error thresholds in holography”. In: *Physical Review D* 106.4 (2022), p. 046009.

[85] Latham Boyle, Madeline Dickens, and Felix Flicker. “Conformal quasicrystals and holography”. In: *Physical Review X* 10.1 (2020), p. 011009.

[86] Alexander Jahn, Zoltán Zimborás, and Jens Eisert. “Central charges of aperiodic holographic tensor-network models”. In: *Physical Review A* 102.4 (2020), p. 042407.

[87] Wolfram Helwig et al. “Absolute maximal entanglement and quantum secret sharing”. In: *Physical Review A* 86.5 (2012), p. 052335.

[88] Wolfram Helwig. “Absolutely maximally entangled qudit graph states”. In: *arXiv preprint arXiv:1306.2879* (2013).

[89] Robert J Harris et al. “Calderbank-Shor-Steane holographic quantum error-correcting codes”. In: *Physical Review A* 98.5 (2018), p. 052301.

[90] Andrew Steane. “Multiple-particle interference and quantum error correction”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452.1954 (1996), pp. 2551–2577.

[91] Matthew Steinberg and Javier Prior. “Conformal properties of hyperinvariant tensor networks”. In: *Scientific Reports* 12.1 (2022), p. 532.

[92] Matthew Steinberg et al. “Quantum Error Correction with Hyperinvariant Tensor-Network Codes”. In: *In Preparation* (2024).

[93] David E Muller. “Application of Boolean algebra to switching circuit design and to error detection”. In: *Transactions of the IRE professional group on electronic computers* 3 (1954), pp. 6–12.

[94] Scott Aaronson and Daniel Gottesman. “Improved simulation of stabilizer circuits”. In: *Physical Review A* 70.5 (2004), p. 052328.

[95] Robert J Harris et al. “Decoding holographic codes with an integer optimization decoder”. In: *Physical Review A* 102.6 (2020), p. 062417.

[96] Andrew J Ferris and David Poulin. “Tensor networks and quantum error correction”. In: *Physical review letters* 113.3 (2014), p. 030501.

[97] Christopher T Chubb. “General tensor network decoding of 2d pauli codes (2021)”. In: *arXiv preprint arXiv:2101.04125* ().

[98] Terry Farrelly et al. “Tensor-network codes”. In: *Physical Review Letters* 127.4 (2021), p. 040507.

[99] David K Tuckett et al. “Tailoring surface codes for highly biased noise”. In: *Physical Review X* 9.4 (2019), p. 041031.

[100] David K Tuckett, Stephen D Bartlett, and Steven T Flammia. “Ultrahigh error threshold for surface codes with biased noise”. In: *Physical review letters* 120.5 (2018), p. 050505.

[101] Arpit Dua et al. “Clifford-deformed surface codes”. In: *PRX Quantum* 5.1 (2024), p. 010347.

[102] Sergey Bravyi, Martin Suchara, and Alexander Vargo. “Efficient algorithms for maximum likelihood decoding in the surface code”. In: *Physical Review A* 90.3 (2014), p. 032326.

[103] Thomas M Stace, Sean D Barrett, and Andrew C Doherty. “Thresholds for topological codes in the presence of loss”. In: *Physical review letters* 102.20 (2009), p. 200501.

[104] Héctor Bombin et al. “Strong resilience of topological codes to depolarization”. In: *Physical Review X* 2.2 (2012), p. 021004.

[105] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2`. 2024.

Acknowledgement

Fifteen years ago, I naively watched Michio Kaku's popular science program on quantum computing. The seeds of aspiration planted in my youth seemed to subconsciously guide me to where I truly wanted to be, doing what I wished to do.

Over the two years at TU Delft, I've undergone many changes—from knowing just a few basic quantum logic gates before starting school to now achieving surprising results. I have so many people and so many serendipitous encounters to be thankful for.

Before I joined the Feld group, I was going through a long period of confusion, not knowing how I could expand my capabilities or contribute in the fields I was passionate about. By chance, I saw some information about the QML group while walking through the corridors of QuTech, and decided to just give it a try. I must say, from my first conversation with Sebastian, I realized this would be a great group with a positive atmosphere, and I found a topic that intrigued me. Naturally, I ended up joining.

The truth is, I made the right choice. In these past eight months, with everyone's support, I've learned a lot of really cool stuff and found effective ways to utilize my skills to solve problems.

I want to thank **Sebastian Feld** for giving me tremendous support in the projects and directions I wanted to pursue, for allowing me the greatest freedom in my research, and for the friendly, family-like atmosphere in the group. All of these have been incredibly important to me. :)

And then a huge, huge thank you to **Matthew Steinberg**. You've given me so much support—from selecting topics and discussions at the beginning, carefully teaching me some of the concepts of this subject, to suggesting really cool research directions, involving me in numerous collaborations and discussions, and meticulously helping me revise my thesis, and so much more. There's almost too much to thank you for. Also, your research area is really cool; maybe we can gradually prove that these cool studies are not just crazy. Plus, I wish you all the best after your PhD graduation. :)

I also want to express my gratitude to **Barbara Terhal**. It was your QEC course in 2023 that sparked my passion for this broad field. Even though I was still feeling lost and down at that time, looking back now, it was a significant turning point during my time at TU Delft. :)

Regarding cool code and a fun atmosphere, I must extend my thanks to **Aritra Sarkar**, **Roberto Turrado Camblor**, and **Pablo Le Henaff**. You've supported me in developing coding thinking and aesthetics, and I've learned a lot from you. :)

I'd also like to thank everyone in the group— **Medina, Nikiforos, Boran, Sibasish, Joris.** etc. You're all doing really cool stuff, and I appreciate the encouragement and support you've given me. Working with you has been a great learning experience.

Finally, regarding computational resources, I would like to thank **Delft Blue** [105] for providing the HPC resources.

Junyu Fan
Delft, November 2024

A

Data and Code Availability

The code and data used in this work will be made available in the open-source community, along with some other preliminary results and performance overhead analysis: `https://github.com/FJY08/HQEC`