UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Final Dissertation

Advanced decoding techniques for quantum error correcting codes

Thesis external supervisor
Dr. Francisco Lázaro Blasco

Thesis internal supervisor
Prof. Giuseppe Vallone

Candidate
Luca Menti

Academic Year 2023/2024

Abstract

Quantum error-correcting codes offer a solution to mitigate quantum noise, which is inherent to quantum mechanics, and therefore they enable reliable quantum computation. Among these codes, quantum low-density parity-check codes are promising candidates for future quantum error correcting codes due to their ability to provide fault tolerance with constant overhead and recent advancements in designing asymptotically good quantum low-density parity-check codes.

A notable challenge in implementing quantum low-density parity-check codes for practical quantum computers is the absence of a universal decoder that delivers good decoding performance across various quantum low-density parity check codes.

Among the proposed solutions, the neural belief propagation decoder, leveraging machine and deep learning techniques, emerges as a promising approach [1]. Neural belief propagation generalizes belief propagation decoding by incorporating trainable weights optimized through supervised learning, thereby accelerating the decoding process and enhancing accuracy in quantum error correction.

In this thesis, we explore the application of machine learning and deep learning techniques in quantum error correction, focusing specifically on the toric code that is typically decoded using the minimum weight perfect matching algorithm. This algorithm has a worst-case complexity $\mathcal{O}(n^3)$ (where n is the number of qubits), and is considered to be too complex for a practical decoder. In particular, we investigate the quaternary neural belief propagation decoder proposed by Miao et al. [2] which has linear complexity in n and we introduce techniques such as residual connections, weight-sharing, higher weight overcomplete check matrices, and weight reuse to potentially improve the decoder's performance.

Our results demonstrate that some of these techniques (the use of weight-sharing and higher weight overcomplete check matrices) can lead to a significant performance improvement for small systems. In particular, for $L = 4$ and $L = 6$ we can largely outperform the minimum-weight perfect matching. However, these gains decrease as L increases, and they are negligible for $L \geqslant 10$. Furthermore, reusing the weights trained for a small toric code with larger toric codes significantly improves the performance with respect to standard belief propagation, although it fails to reach the performance of minimum-weight perfect matching.

Acknowledgments

Innanzitutto, desidero dedicare questa tesi a due donne eccezionali che hanno lasciato un segno indelebile nella mia vita: le mie amate nonne Liliana e Bruna. Da loro ho appreso i valori fondamentali di costanza, sacrificio e determinazione che sono stati la bussola nel mio cammino universitario e di vita. Il loro esempio di forza e resilienza mi ha sempre ispirato a dare il massimo e a non arrendermi mai di fronte alle avversità.

Ringrazio di cuore mia sorella Ilaria, compagna di una vita, con la quale ho condiviso gioie e dolori sin dalla tenera età. Nonostante le sfide che la vita ci ha posto davanti, siamo cresciuti insieme, sostenendoci a vicenda e rafforzando quel legame fraterno che ci rende unici.

Un grazie speciale va ai miei genitori, Stefania e Franco, per avermi insegnato l'importanza dell'ambizione e della grinta necessarie per raggiungere i traguardi che mi ero prefissato. Grazie per avermi sempre incoraggiato a inseguire i miei sogni e per avermi sostenuto in ogni mia scelta, anche quando sembrava ardua.

Ringrazio di cuore i miei zii Loris e Marco, e le mie zie Nadia e Fiorenza, per essere stati un punto di riferimento costante. Grazie per la vostra disponibilità nell'aiutarmi nei momenti più difficili e per avermi fatto sentire sempre parte di una grande famiglia unita .

Un ringraziamento speciale va anche ai miei cugini, con i quali ho condiviso tanti bei ricordi e momenti spensierati. Grazie per aver allietato la mia infanzia e per essere rimasti al mio fianco anche da grandi.

Desidero inoltre ringraziare tutti i miei amici, che rappresentano la mia seconda famiglia. Grazie per avermi supportato e sopportato in questo ultimo anno, in cui le difficoltà da superare sembravano insormontabili. Il vostro affetto e la vostra vicinanza sono stati la forza per rialzarmi. Vi sarò per sempre grato.

Infine, ma non per importanza, ringrazio il mio supervisor Dr. Francisco Lázaro Blasco per aver creduto in me fin dal primo giorno al Deutsche Zentrum für Luft- und Raumfahrt. Grazie per avermi guidato in questo percorso di tesi e di crescita lavorativa, arricchendomi di preziosi insegnamenti.
Fran ti sarò sempre riconoscente per l'opportunità che mi ha offerto e per avermi spronato a dare il meglio di me.

A tutti voi, che avete reso possibile questo traguardo, va il mio più sincero e profondo ringraziamento. Senza il vostro amore, il vostro supporto e la vostra fiducia, non sarei qui oggi a festeggiare il completamento di questa tesi.

List of Acronyms

QEC	Quantum Error Correction
QECC	Quantum Error Correcting Codes
QLDPC	Quantum Low-Density Parity-Check
LDPC	Low-Density Parity-Check
MWPM	Minimum Weight Perfect Matching
BP	Belief Propagation
BP2	Binary Belief Propagation
BP4	Quaternary Belief Propagation
OBP4	BP4 utilizing an overcomplete check matrix
NBP	Neural Belief Propagation
NBP4	Quaternary Neural Belief Propagation
ONBP4	NBP4 utilizing an overcomplete check matrix
LSTM	Long Short-Term Memory
VN	variable node
CN	check node
MS	Min-Sum
QSC	Quantum Stabilizer Codes
PCM	Parity-Check Matrix
CSS	Calderbank–Shor–Steane
GF(4)	Galois Field
Log-Likelihood Ratio	LLR
Calderbank–Shor–Steane	CSS
Neural Network	NN
Convolutional Neural Networks	CNNs
Bose–Chaudhuri–Hocquenghem	BCH

Contents

Chapter 1

Introduction

1.1 Motivation

The idea of computers that use the laws of quantum mechanics was introduced the first time by Feynman in 1982, in order to simulate quantum systems which are exponentially hard to simulate on classical computers [3]. Since then, the field of quantum computing has been subject to considerable progress, and today we know that quantum computers can provide a speed up compared to classical ones for other relevant problems. The best example is probably Shor's algorithms [4], which can provide an exponential speedup compared to the best known classical factoring algorithm. This technology presents the potential to speed up algorithms that classically are computationally hard in terms of computational complexity. Therefore, novel quantum processors could improve different applications related to cryptography, optimization, or basic science, for example, all fields that have a huge impact in modern society. However, this new technology presents different obstacles that must be overcome in order to achieve the theoretical promised results. One of the main challenges is related to quantum noise, which is intrinsic in quantum mechanics. In fact, the qubits that constitute quantum computers are prone to suffer from errors (which can have many origins such as decoherence or imperfect gates, for instance, [5]) implying that the computations being performed are inaccurate.

Quantum Error Correcting Codes (QECC) provide a solution to face this challenge and enabling reliable quantum computation consequently. Quantum Error Correction (QEC) was demonstrated to be possible theoretically when the 9-qubit Shor code was proposed in 1995 and generalized afterwards with the theory of Quantum Stabilizer Codes (QSC). From that point on, the field of QEC has advanced significantly, and different families of QECC have been proposed, such as topological codes and Quantum Low-Density Parity-Check (QLDPC) codes. In particular, the latter ones are among the most promising canditates for future QEC due to the fact that they display the potential of providing fault-tolerance with constant overhead and the recent breakthroughs in designing asymptotically good QLDPC.

The lack of a universal decoder that provides good decoding performances across a wide range of QLDPC codes is one of the most notable challenges that the implementation of QLDPC for practical quantum computers faces. Two main families of potential decoders have been explored in the literature. The first family consists of graph-theory-based decoders, such as the Minimum Weight Perfect Matching (MWPM) decoder, which is commonly used for topological codes even if it is not optimal in that it has a worst-case complexity $\mathcal{O}(n^3)$ (where n is the number of qubits). The second family is the class of message-passing decoders such as the belief propagation decoder or its low-complexity variant, the min-sum (MS) decoder. Belief Propagation (BP) works by iteratively passing messages between the so-called variables and check nodes in the code, and updating their probabilities based on the received messages. In this case, the decoding complexity is linear with respect to the block length and the number of decoding iterations. Although message passing decoders exhibit excellent performance for classical Low-Density Parity-Check (LDPC) codes thanks to the condition that the Tanner Graph has no short cycle, the performance deteriorates for QECC due to the unavoidable presence of length 4 cycles [6].

Among the several methods that have been proposed to address this problem, the neural belief propagation decoder, based on machine and deep learning techniques, is one of the most promising [1].

Neural Belief Propagation is a generalization of BP decoding in which the messages passed between variable and check nodes are multiplied by (trainable) weights that can be optimized relying on supervised learning.

Neural Belief Propagation for quantum error correction codes offers significant advantages. Primarily, it significantly reduces the computational cost of the decoding procedure. Traditional QECC techniques can be expensive in terms of computation and resources, but the application of neural networks can overcome this issue by accelerating the decoding process. Furthermore, NBP can lead to higher accuracy in decoding QECCs. Neural networks have the ability to learn and capture correlations between quantum errors, enabling them to handle even large-scale quantum systems more effectively. This ability to recognize complex error patterns contributes to improved decoding performance, potentially surpassing traditional methods in both speed and accuracy. Finally, NBP has linear decoding complexity (as BP) and this could be really advantageous.

This thesis is driven by the need to advance quantum error correction techniques beyond conventional methods. The focus is on investigating the potential of neural belief propagation in the context of quantum error correcting codes. This approach merges two powerful concepts: the adaptability and learning capabilities of neural networks and the probabilistic reasoning of belief propagation algorithms.

The primary objective is to develop a system that is capable of accurately detecting and identifying quantum errors that may arise during quantum computations. This system will utilize advanced deep learning methodologies to ensure the precise recognition of various types of errors, thereby enhancing the reliability and performance of quantum information processing. In particular, we start with the quaternary neural belief propagation decoder (NBP4) proposed in [2][1] and we introduce the following deep learning techniques to improve the performance of the decoder itself:

- *Residual Connections*: This is a deep learning framework where the weight layers are designed to learn residual functions in relation to the inputs of the layers. This allows information to bypass one or more layers, facilitating direct data transfer from earlier layers to later ones.
- *Weight-Sharing*: This fundamental concept in convolutional neural networks significantly enhances their efficiency and performance. It refers to the practice of using the same set of weights (filter) across different spatial locations in the input data, allowing the network to learn features in a more efficient manner.
- *Weights Reuse*: This technique involves training our model for a fixed system size and subsequently applying the obtained weights to systems of different sizes (whether larger or smaller) by leveraging the symmetry introduced by the application of the weight-sharing filter. This leads to significant savings in both time and computational costs (in fact training NBP is complex for very large systems), as we need to train only one system.

In addition to this, we utilized the overcomplete check matrices proposed in [2]. In particular, overcomplete check matrices are distinguished by the inclusion of supplementary redundant rows in addition to the initial full-rank check matrix. The quaternary belief propagation algorithm can be implemented on the Tanner graph corresponding to these overcomplete check matrices. In the work of Miao et al. [2], they used overcomplete check matrices with weight 6. In this thesis, we have increased the weight to 8 in an attempt to obtain improved performance.

By exploring this intersection of machine & deep learning and quantum information theory, we seek to overcome the limitations of traditional error correction methods. In particular, our challenges are

[1]In this thesis, we have used (and modified) the code proposed in [2], whose source can be found at `https://github.com/kit-cel/Quantum-Neural-BP4-demo`.

to get good performances for the NBP and to reduce the complexity of the training for large systems. The goal is to contribute to the development of more robust and scalable quantum computing systems, potentially paving the way for practical, large-scale quantum applications.

1.2 Existing Work

Although the application of neural networks for decoding algorithm is a well-established approach for several years, NBP is quite new for decoding QECC. In this chapter a state of art's overview is presented in order to underline the most important discoveries in this field.

In 2016, the field of error correction in communication systems experienced a significant breakthrough with the publication of Nachmani et al.'s seminal work [1]. This innovative research marked a pivotal moment by introducing deep learning techniques to address challenges in error correction algorithms. Conventional error correction methods, particularly the belief propagation algorithm, had consistently struggled with the complexities of high-dimensional data and intricate error patterns. These traditional approaches often relied on manually designed error models and heuristics, which have difficulties in fully encapsulating the interdependencies and correlations inherent in real-world communication channels. The approach in [1] utilized the capabilities of deep neural networks, offering a fresh perspective on error correction. Their method demonstrated remarkable improvements in both accuracy and efficiency compared to the standard BP algorithm. Crucially, these enhancements were achieved without incurring additional computational complexity, a factor that had previously limited the practical application of more sophisticated error correction techniques.

A significant progress in quantum error correction's field can be found in Poulin et al.'s NBP decoders studying [7]. In this work, published in 2019, a new approach based on the combination between neural networks and BP is presented, which led to higher accuracy and efficiency of error correction in quantum system. Indeed, the union between NBP algorithms that leverage graphical models to perform probabilistic inference and optimize error correction, and neural networks allows to capture intricate correlations and dependencies in quantum error patterns with improvements in error correction capabilities consequently.

Poulin et al. proposed the implementation of enhanced NBP for quantum error correction with the introduction of a new loss function that takes into account error degeneracy. Furthermore, they introduced weight sharing and soft weight techniques for toric codes in order to study the scalability of the approach to larger quantum systems. Finally, they explored some potential enhancements for the optimization of error correction performances.

Miao et al.'s is one of the most recent works in quantum error correction field which brought a significant advancement [2]. In this research, they initially suggested using a belief propagation decoder that functions on overcomplete check matrices to interpret QLDPC codes. Subsequently, they extended the scope of the neural NBP decoder, which was originally examined for suboptimal binary BP decoding of QLPDC codes, to encompass quaternary BP decoders. The outcome of their numerical simulations illustrate that both methodologies, as well as their integration, produce a swift and highly effective decoder for various QLDPC codes ranging from short to moderate lengths.

Recent research into NBP for quantum error correction has yielded encouraging outcomes across a spectrum of quantum error-correcting codes. These studies have introduced diverse neural network structures and training approaches for NBP, demonstrating enhanced decoding capabilities compared to alternative quantum error correction methods. Nonetheless, the field still faces numerous unresolved issues and hurdles. Key areas requiring further investigation include refining neural network architectures to better suit quantum error correction tasks and developing more efficient training protocols, particularly for large-scale quantum systems. These ongoing challenges underscore the need for continued research and innovation in the application of NBP to quantum error correction. As the field progresses, addressing these open questions could potentially lead to significant advancements in the reliability and efficiency of quantum computing systems.

1.3 Outline of this Thesis

The application of machine learning and deep learning techniques to enhance the decoding performance of quantum codes represents a concrete and promising possibility. In particular, the NBP algorithm has already demonstrated more than satisfactory results comparable to MWPM. The primary objective of this thesis is to explore the extension of neural belief propagation from the binary case to the quaternary case, as proposed in [2]. In doing so, we aim to incorporate advanced artificial intelligence techniques to enhance the performance of this framework, for achieving significant improvements in the accuracy and efficiency. The thesis is structured as follows:

- *Chapter 1 : Introduction,* this chapter offers a summary of the motivation and context for utilizing NBP in Quantum Error Correction Codes. It also outlines the goals of the thesis and highlights some pertinent research in the field.
- *Chapter 2 : Quantum Error Correction,* this chapter provides a concise overview of the fundamental concepts of quantum computing and information. The chapter begins with a brief introduction to the core principles of quantum information and computing. Following this, it presents the mathematical formalism used to describe quantum error correcting codes and the various operations that can be performed on them.
- *Chapter 3 : Classical Decoding Algorithms,* this chapter provides the basics of classical error-correcting codes, it explains how these codes can detect and correct errors in classical information. In particular it delves into the explanation of the quaternary belief propagation.
- *Chapter 4 : Neural Networks & Deep Learning Techniques,* this chapter details the extension of the NBP framework to the quaternary belief propagation decoder and describes the deep learning techniques adopted in this thesis in order to try to improve the performances.
- *Chapter 5 : Numerical Results,* this chapter details the training and test procedure applied for the neural belief propagation. Furthermore it discusses the results obtained and it compares them with the literature´s ones.
- *Chapter 6 : Conclusions,* this chapter summarizes the main findings and contributions of the thesis.

Chapter 2

Quantum Error Correction

In this chapter, we provide a concise overview of the fundamental concepts of quantum computing and information. We begin with a brief introduction to the core principles of quantum information and computing. Following this, we present the mathematical formalism used to describe quantum error correcting codes and the various operations that can be performed on them.

2.1 Classical Information

In classical information theory, the bit serves as the fundamental unit of information, represented by elements of the set $\{0, 1\} \in \mathbb{Z}_2$. The physical state of a classical system can manifest itself in two distinct forms:

1. A deterministic state, represented by either $|0\rangle\langle 0|$ or $|1\rangle\langle 1|$.
2. A probabilistic state, described by $q|0\rangle\langle 0| + (1-q)|1\rangle\langle 1|$, where $q \in \mathbb{R}$ and $0 < q \leq 1$.

In the case of a probabilistic state, the system exhibits uncertainty. This uncertainty is quantified by the probability q of finding the system in state $|0\rangle\langle 0|$, and consequently, a probability $(1-q)$ of it being in state $|1\rangle\langle 1|$. It's important to note that this uncertainty is a reflection of our knowledge about the system, rather than an inherent property of the system itself.
The sole single-bit operation in classical information theory is the bit-flip, which operates as follows:

$$\begin{aligned} 0 &\to 1 \\ 1 &\to 0 \end{aligned}$$

In the context of classical information, noise typically transitions the system from a deterministic state to a probabilistic state.

Building upon our understanding of classical bits and their states, we can now explore the concept of computation in classical information theory. Computation is the process of manipulating multiple bits through specific mappings. More formally, it consists of logical operations that act on one or more bits simultaneously. When we model computation by means of a circuit, the devices that carry out logical operations are known as logical gates. It is important to note that while we typically represent computational operations as mappings between deterministic states, real-world scenarios often involve uncertain states. Thus, in practice, computations map one probabilistic state to another.

2.2 Quantum Information

In the realm of quantum information, the state of the system is represented by a quantum state, a normalized vector in a two-dimensional Hilbert space, denoted $\mathcal{H}_2$. We express this as:

$$|\psi\rangle \in \mathcal{H}_2, \quad \text{such that} \quad \langle\psi|\psi\rangle = 1 \tag{2.1}$$

The standard basis for quantum information is known as the computational basis, comprising $\{|0\rangle, |1\rangle\}$. Any system state can be expressed as a linear combination (superposition) of these basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \text{where} \quad \alpha, \beta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

Given that the global phase is physically irrelevant, we can choose α to be real and non-negative. This, combined with the normalization condition $|\alpha|^2 + |\beta|^2 = 1$, allows us to represent the qubit state as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{2.3}$$

where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$. The parameters θ and ϕ can be interpreted as spherical coordinates, giving rise to a unit sphere in $\mathbb{R}^3$ known as the Bloch sphere (illustrated in Fig. 2.1). Each point on the Bloch sphere, characterized by the unit vector $\vec{n} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$, corresponds to a unique qubit state. It is noteworthy that antipodal points on the Bloch sphere represent orthogonal states.

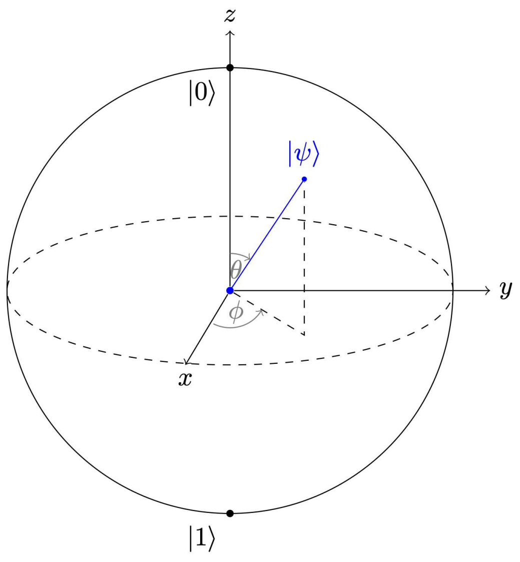


Figure 2.1: Bloch sphere representation of a qubit state.

The Bloch sphere representation can be extended to encompass mixed states as well. Any density operator $\boldsymbol{\rho}$ in a two-dimensional Hilbert space can be expressed as:

$$\boldsymbol{\rho} = \frac{1}{2}(\mathbb{I} + \vec{r} \cdot \vec{\sigma}) \tag{2.4}$$

where $\mathbb{I}$ denotes the identity matrix, $\vec{r} = (r_x, r_y, r_z) \in \mathbb{R}^3$ is a real vector, and $\vec{\sigma} = (\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y, \boldsymbol{\sigma}_z)$ is the vector of Pauli matrices.

The Pauli matrices, which are fundamental to this representation, also play a crucial role in quantum computing, as quantum gates, the building stones of quantum circuits. Quantum gates, represented as matrices, are operations that transform the state of qubits, enabling the execution of quantum algorithms and computations. These matrices represent essential operators used in quantum error correction and quantum circuit operations. As concern the Pauli's ones, they are defined as:

$$\boldsymbol{X} \equiv \boldsymbol{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{Y} \equiv i\boldsymbol{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \boldsymbol{Z} \equiv \boldsymbol{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.5}$$

Each of these matrices corresponds to a specific type of quantum gate:

- The Pauli-$\boldsymbol{X}$ gate (bit-flip)
- The Pauli-$\boldsymbol{Y}$ gate (bit-phase-flip)
- The Pauli-$\boldsymbol{Z}$ gate (phase-flip)

These gates, along with others like the Hadamard gate and the CNOT gate, form the building blocks of quantum circuits. They enable the execution of quantum algorithms by transforming the state of qubits. The effect of these Pauli operators on a generic single qubit state in superposition ($\alpha|0\rangle + \beta|1\rangle$) can be described as:

$$\boldsymbol{X}(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle \tag{2.6}$$
$$\boldsymbol{Y}(\alpha|0\rangle + \beta|1\rangle) = i(\alpha|1\rangle - \beta|0\rangle) \tag{2.7}$$
$$\boldsymbol{Z}(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle \tag{2.8}$$

These transformations correspond to specific movements on the Bloch sphere, reinforcing the connection between the abstract mathematical representation and the geometric visualization provided by the Bloch sphere.

The Pauli matrices exhibit two fundamental properties: Hermiticity and Unitarity.

Hermiticity: Each Pauli matrix is self-adjoint, which means it is identical to its own conjugate transpose. Mathematically, we express this as:

$$A = A^\dagger \tag{2.9}$$

where $A^\dagger$ denotes the Hermitian conjugate (also called the adjoint or conjugate transpose) of matrix A.

Unitarity: The Pauli matrices are also unitary, a property that implies the inverse of a matrix is equal to its Hermitian conjugate. We can represent this mathematically as:

$$A^\dagger A = AA^\dagger = \mathbb{I} \tag{2.10}$$

where $\mathbb{I}$ represents the identity matrix.

An intriguing property emerges when we multiply Pauli matrices: the product of any two distinct Pauli operators yields another Pauli operator, disregarding a phase factor of i (the imaginary unit). This relationship can be summarized as follows:

$$\boldsymbol{XY} = i\boldsymbol{Z} \qquad \boldsymbol{YX} = -i\boldsymbol{Z} \tag{2.11}$$
$$\boldsymbol{YZ} = i\boldsymbol{X} \qquad \boldsymbol{ZY} = -i\boldsymbol{X} \tag{2.12}$$
$$\boldsymbol{XZ} = i\boldsymbol{Y} \qquad \boldsymbol{ZX} = -i\boldsymbol{Y} \tag{2.13}$$

These multiplication rules highlight the cyclic nature and interrelationships among the Pauli matrices, which are fundamental to their application in quantum mechanics and quantum computing.

2.3 Stabilizer Formalism

In this section, we aim to leverage the properties just discussed to construct a more comprehensive code for n qubits, known as the stabilizer code. To establish the stabilizer formalism, we must first introduce the Pauli group.

Definition 2.3.1. *The Pauli group, $\mathcal{P}_1$, is the group consisting of the 2×2 identity matrix, $\mathbb{I}$, and the Pauli matrices together with the product of these matrices with the factor -1, which are*

$$\mathcal{P}_1 \equiv \{\pm\mathbb{I}, \pm\boldsymbol{X}, \pm\boldsymbol{Y}, \pm\boldsymbol{Z}\}$$

where $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ are defined in Equation 2.5.

It is worth noting that the Pauli group has an order of eight, $|\mathcal{P}_1| = 8$, indicating that it contains eight elements. These elements are interconnected through the commutation relations of the Pauli matrices:

$$\begin{aligned}[\boldsymbol{X}, \boldsymbol{Y}] &= 2i\boldsymbol{Z} \\ [\boldsymbol{X}, \boldsymbol{Z}] &= -2i\boldsymbol{Y} \\ [\boldsymbol{Y}, \boldsymbol{Z}] &= 2i\boldsymbol{X}\end{aligned}$$

When we consider the n-fold tensor product of the Pauli group, we obtain another group, denoted as $\mathcal{P}_n$:

$$\mathcal{P}_n \equiv \{\pm\mathbb{I}, \pm\boldsymbol{X}, \pm\boldsymbol{Y}, \pm\boldsymbol{Z}\}^{\otimes n} \equiv \{\pm\boldsymbol{G}_{\vec{\alpha}}\}$$

For concise notation, we define $\boldsymbol{G}_{\vec{\alpha}} = \boldsymbol{\sigma}_{\alpha_1} \otimes \cdots \otimes \boldsymbol{\sigma}_{\alpha_n}$ where $\boldsymbol{\sigma}_{\alpha_1} = \mathbb{I}$, $\boldsymbol{\sigma}_{\alpha_2} = \boldsymbol{X}$, $\boldsymbol{\sigma}_{\alpha_3} = \boldsymbol{Y}$, and $\boldsymbol{\sigma}_{\alpha_4} = \boldsymbol{Z}$.

Some notable properties of the group P_n include:

- Its order is $|\mathcal{P}_n| = 2 \cdot 4^n = 2^{2n+1}$
- Any element $\boldsymbol{G}_{\vec{\alpha}} \in \mathcal{P}_n$ satisfies $\boldsymbol{G}_{\vec{\alpha}}^2 = \mathbb{I}$ and $\boldsymbol{G}_{\vec{\alpha}}^\dagger \boldsymbol{G}_{\vec{\alpha}} = \mathbb{I}$
- For any two distinct elements $\boldsymbol{G}_{\vec{\alpha}}, \boldsymbol{G}_{\vec{\beta}} \in \mathcal{P}_n$, they either commute ($[\boldsymbol{G}_{\vec{\alpha}}, \boldsymbol{G}_{\vec{\beta}}] = 0$) or anticommute ($\{\boldsymbol{G}_{\vec{\alpha}}, \boldsymbol{G}_{\vec{\beta}}\} = 0$). In the case of commutation, they also share a common eigenbasis.

The weight w of a Pauli operator $\mathcal{P}_n$ is the number of non-identity components in the tensor product.

Having examined the Pauli group and its generalization to n qubits, we can now proceed to define the stabilizer code.

Definition 2.3.2. *Let $\mathcal{S}$ be an abelian[1] subgroup of $\mathcal{P}_n$. A stabilizer code, $\mathcal{C}$, is defined as:*

$$\mathcal{C} \equiv \{|\psi\rangle \mid S\,|\psi\rangle = |\psi\rangle \quad \forall S \in \mathcal{S}\}$$

We refer to $\mathcal{S}$ as the stabilizer (group) of the code, and the elements $S \in \mathcal{S}$ as stabilizer operators of the code. The stabilizer group completely characterizes the code.

It's important to note that stabilizer operators do not generally exhibit linear independence. The concept of linear independence is traditionally associated with vector spaces rather than groups. In our context, when we discuss linear independence, we're actually referring to a mapping of elements from $\mathcal{P}_n$ onto the vector space $(\mathbb{Z}_2)^{2n}$. This mapping is defined as:

$$\varphi : \left(\frac{\mathcal{P}_n}{\mathbb{Z}_2}; \cdot\right) \longrightarrow (\mathbb{Z}_2)^{2n}$$

[1]A group is abelian if all its elements commute, i.e., $[S_1, S_2] = 0\ \forall S_i \in \mathcal{S}$.

Once this mapping is established, we can apply the notion of linear independence within $(\mathbb{Z}_2)^{2n}$. This concept has a straightforward interpretation for elements of $\mathcal{P}_n$: two elements are considered linearly independent if their product does not belong to $\mathcal{P}_n$. It is worth noting that any product of these elements remains within the group.

To simplify our approach, we aim to identify the smallest set of elements capable of generating the entire stabilizer group. We refer to these as the generators of the stabilizer group. More precisely, the generators are a set of operators $\{S_j\}_{j=1}^m$, $S_j \in \mathcal{S}$, which are both mutually commuting and linearly independent. This formulation leads to an important result: the stabilizer code contains $k = n - m$ logical qubits. In other words, the code space $\mathcal{C}$ has a dimension of 2^k.

Example 2.3.1. *Consider the Shor code represented by nine qubits. The stabilizer group is generated by eight operators, denoted as $\mathcal{S} = \langle \{S_k\}_{k=1}^8 \rangle$, which can be expressed as follows:*

$$\begin{aligned} S_1 &= \boldsymbol{Z}_1\boldsymbol{Z}_2, \quad S_2 = \boldsymbol{Z}_2\boldsymbol{Z}_3, \quad S_3 = \boldsymbol{Z}_4\boldsymbol{Z}_5, \\ S_4 &= \boldsymbol{Z}_5\boldsymbol{Z}_6, \quad S_5 = \boldsymbol{Z}_7\boldsymbol{Z}_8, \quad S_6 = \boldsymbol{Z}_8\boldsymbol{Z}_9, \\ S_7 = \boldsymbol{X}_1\boldsymbol{X}_2\boldsymbol{X}_3\boldsymbol{X}_4\boldsymbol{X}_5\boldsymbol{X}_6, &\quad S_8 = \boldsymbol{X}_4\boldsymbol{X}_5\boldsymbol{X}_6\boldsymbol{X}_7\boldsymbol{X}_8\boldsymbol{X}_9. \end{aligned}$$

We can derive additional stabilizer operators by multiplying any two generators of the stabilizer group:

$$\begin{aligned} S_1S_2|\phi\rangle = \boldsymbol{Z}_1\boldsymbol{Z}_3|\phi\rangle = \boldsymbol{Z}_1\boldsymbol{Z}_3(\alpha|\bar{0}\rangle + \beta|\bar{1}\rangle) = \boldsymbol{Z}_1\boldsymbol{Z}_3(\alpha|+++\rangle + \beta|---\rangle) = \\ = \boldsymbol{Z}_1(\alpha|-++\rangle + \beta|+--\rangle) = (\alpha|+++\rangle + \beta|---\rangle) = |\phi\rangle, \end{aligned}$$

where $|\bar{0}\rangle$, and $|\bar{1}\rangle$ are defined as the logical qubits of the Shor code:

$$|\bar{0}\rangle = \frac{1}{\sqrt{2^3}}\left(|000\rangle + |111\rangle\right)\left(|000\rangle + |111\rangle\right)\left(|000\rangle + |111\rangle\right)$$

$$|\bar{1}\rangle = \frac{1}{\sqrt{2^3}}\left(|000\rangle - |111\rangle\right)\left(|000\rangle - |111\rangle\right)\left(|000\rangle - |111\rangle\right).$$

while for $|\pm\rangle$ it is easy to see that the structure of the logical qubits consists of three chunks of three qubits each. If we focus on a single chunk, we can interpret it as a logical $\pm$ of the classical repetition code, i.e.:

$$|\pm\rangle = \frac{1}{\sqrt{2}}\left(|000\rangle \pm |111\rangle\right).$$

Therefore, we conclude that $S_1S_2|\phi\rangle = |\phi\rangle$, which indicates that $S_1S_2 \in \mathcal{S}$.

Let us examine a state $|\psi\rangle$ within the code space $\mathcal{C}$, and an operator T that commutes with all stabilizer operators, such that $[T, S_k] = 0 \ \forall \ S_k \in \mathcal{S}$. We can demonstrate that the state $T|\psi\rangle$ also resides in the code space through the following sequence:

$$\begin{aligned} T|\psi\rangle &= TS_k|\psi\rangle = S_kT|\psi\rangle \\ &\Rightarrow S_kT|\psi\rangle = T|\psi\rangle \\ &\Rightarrow T|\psi\rangle \in \mathcal{C} \end{aligned}$$

Furthermore, we designate T as a logical operator because it maps a state in the code space, $|\psi\rangle \in \mathcal{C}$, to another state[2] within the same code space, $|\psi'\rangle \in \mathcal{C}$. It is important to note that this property does not hold when T anticommutes with the stabilizer operators.

[2]Stabilizer operators are not logical operators because they map a state in the code, $|\psi\rangle$, to itself, not to another state.

Example 2.3.2. *Consider the Shor code on nine qubits. Its logical operators are defined as follows:*

$$\bar{\boldsymbol{X}} = \boldsymbol{X}_1 \cdot \boldsymbol{X}_9 = \prod_{j=1}^{9} \boldsymbol{X}_j \quad \textit{and} \quad \bar{\boldsymbol{Z}} = \boldsymbol{Z}_1 \cdot \boldsymbol{Z}_9 = \prod_{j=1}^{9} \boldsymbol{Z}_j, \tag{2.14}$$

where $[\bar{\boldsymbol{X}}, \bar{\boldsymbol{Z}}] = 2\bar{\boldsymbol{Y}}$. *We are interested in the effect of these operators on the logical qubits, which is given by:*

$$\begin{aligned} \bar{\boldsymbol{X}}|\bar{0}\rangle &= |\bar{1}\rangle, \quad \bar{\boldsymbol{Z}}|\bar{0}\rangle &= |\bar{0}\rangle, \\ \bar{\boldsymbol{X}}|\bar{1}\rangle &= |\bar{0}\rangle, \quad \bar{\boldsymbol{Z}}|\bar{1}\rangle &= -|\bar{1}\rangle, \end{aligned}$$

where $|\bar{0}\rangle$ *and* $|\bar{1}\rangle$ *are the logical qubits of Shor code. Thus, the operators* $\bar{\boldsymbol{X}}$ *and* $\bar{\boldsymbol{Z}}$ *are the logical Pauli operators* $\boldsymbol{X}$ *and* $\boldsymbol{Z}$.

Consider a state $|\psi\rangle \in \mathcal{C}$. We can demonstrate that any operator of the form $\bar{\boldsymbol{X}}S_k$ also functions as a logical operator, as it maps $|\psi\rangle \in \mathcal{C}$ to another state $|\psi'\rangle \equiv \bar{\boldsymbol{X}}|\psi\rangle \in \mathcal{C}$. This can be illustrated as follows:

$$\bar{\boldsymbol{X}}S_k|\psi\rangle = \bar{\boldsymbol{X}}|\psi\rangle = |\psi'\rangle \in \mathcal{C}$$

This property leads to an important observation: logical operators are not uniquely defined. In fact, there exists a class of equivalent logical operators that produce the same effect on the code space.

Example 2.3.3. *Consider the Shor code on nine qubits. More logical operators apart from* $\bar{\boldsymbol{X}}$ *and* $\bar{\boldsymbol{Z}}$ *(Eq. 2.14) would be:*

$$\bar{\boldsymbol{X}}S_8 = \boldsymbol{X}_1\boldsymbol{X}_2\boldsymbol{X}_3,$$
$$\bar{\boldsymbol{X}}S_2 = \boldsymbol{X}_1\boldsymbol{Y}_2\boldsymbol{Y}_3\boldsymbol{X}_4\boldsymbol{X}_5\boldsymbol{X}_6\boldsymbol{X}_7\boldsymbol{X}_8\boldsymbol{X}_9$$

It can be demonstrated that the minimum weight of all logical operators corresponds to the code's distance. This is intuitive because a logical operator transforms one logical state into another. Consequently, the minimum length of a logical operator represents the fewest number of qubit operations required to map between two distinct logical states. In fact, this is the definition of the code's distance.

Let us consider a general error of the form $\mathcal{E} = e_x\boldsymbol{X} + e_y\boldsymbol{Y} + e_z\boldsymbol{Z}$, we can correct errors $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ individually. For an error $\mathcal{E}_\alpha \in \mathcal{P}_n$, it either commutes or anticommutes with the stabilizer operators. We can analyze these cases as follows:

- If the error commutes with a stabilizer operator S:

$$S\mathcal{E}_\alpha|\psi\rangle = \mathcal{E}_\alpha S|\psi\rangle = \mathcal{E}_\alpha|\psi\rangle$$

- If the error anticommutes with a stabilizer operator S:

$$S\mathcal{E}_\alpha|\psi\rangle = -\mathcal{E}_\alpha S|\psi\rangle = -\mathcal{E}_\alpha|\psi\rangle$$

In other words, for $|\psi\rangle \in \mathcal{C}$, the state $\mathcal{E}_\alpha|\psi\rangle$ is an eigenvector of the stabilizer operators. The corresponding eigenvalue is $+1$ if $[\mathcal{E}_\alpha, S] = 0$ (commutation) and -1 if $\{\mathcal{E}_\alpha, S\} = 0$ (anticommutation). This property enables the stabilizer to function as a parity check, effectively detecting errors by measuring these eigenvalues.

2.4 Sympletic and Quaternary Representations

To construct a valid stabilizer code, it is essential to determine the stabilizer group $\mathcal{S}$, which is an Abelian subgroup of $\mathcal{P}_n$. This task can be converted into a classical coding problem by mapping Pauli errors to binary strings as follows:

$$\mathcal{P}_n \mapsto \big(x_1 \cdots x_n \mid z_1 \cdots z_n\big) =: \big(p_X \mid p_Z\big)$$

where $\mathcal{P}_i = \boldsymbol{X}^{x_i}\boldsymbol{Z}^{z_i}$. Pauli errors $\mathcal{A}$ and $\mathcal{B}$ in $\mathcal{P}_n$ commute if the symplectic product of their corresponding binary strings $(a_X \mid a_Z)$ and $(b_X \mid b_Z)$ equals zero, i.e.,

$$\sum_{i=1}^{n} a_{X,i} \cdot b_{Z,i} + \sum_{i=1}^{n} a_{Z,i} \cdot b_{X,i} = 0 \tag{2.15}$$

where the addition is performed in the binary field (modulo 2). Consequently, a stabilizer code can be derived from a $[2n, k]$ classical binary code, represented by its full-rank parity-check matrix (PCM) $\boldsymbol{H}= \big(\boldsymbol{H}_X \mid \boldsymbol{H}_Z\big)$ with dimensions $m = 2n - k$ by $2n$, satisfying the symplectic condition:

$$\boldsymbol{H}_X\boldsymbol{H}_Z^T + \boldsymbol{H}_Z\boldsymbol{H}_X^T = 0 \tag{2.16}$$

Calderbank–Shor–Steane (CSS) codes represent a specialized class of stabilizer codes characterized by a parity check matrix of the form:

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}'_X & 0 \\ 0 & \boldsymbol{H}'_Z \end{pmatrix} \tag{2.17}$$

In this structure, 2.16 is satisfied when $\boldsymbol{H}'_X\boldsymbol{H}'^T_Z = 0$, which simplifies the code construction process.

To simultaneously decode the four types of Pauli errors, it is advantageous to consider the quaternary form of $\boldsymbol{H}$, denoted as $\boldsymbol{S} \in \mathrm{GF}(4)^{m\times n}$, where the Galois Field GF(4) consists of the elements $\{0, 1, \omega, \overline{\omega}\}$ which are linked to the Pauli operators by the following map:

$$\mathbb{I} \to 0 \quad \boldsymbol{X} \to 1 \quad \boldsymbol{Z} \to \omega \quad \boldsymbol{Y} \to \overline{\omega}$$

To properly adopt this representation the sum and product in GF(4) are reported in Tables 2.1 and 2.2.

$+$	0	1	ω	$\overline{\omega}$
0	0	1	ω	$\overline{\omega}$
1	1	0	$\overline{\omega}$	ω
ω	ω	$\overline{\omega}$	0	1
$\overline{\omega}$	$\overline{\omega}$	ω	1	0

Table 2.1: Addition Table in GF(4).

$\times$	0	1	ω	$\overline{\omega}$
0	0	0	0	0
1	0	1	ω	$\overline{\omega}$
ω	0	ω	$\overline{\omega}$	1
$\overline{\omega}$	0	$\overline{\omega}$	1	ω

Table 2.2: Multiplication Table in GF(4).

Based on this isomorphism, multiplying Pauli operators is equivalent to adding them over GF(4). Additionally, since Pauli operators are mapped to a quaternary symbol, it is feasible to express the PCM as a quaternary matrix.

In light of these considerations, the matrix $\boldsymbol{H}$ (Eq. 2.17) can be transformed into $\boldsymbol{S}$ by mapping binary vectors to vectors over GF(4) as follows:

$$\big(x_1 \cdots x_n \mid z_1 \cdots z_n\big) \mapsto \big(p_1 \cdots p_n\big) =: p \quad with \quad p_i = x_i\omega + z_i\overline{\omega}.$$

We can then verify that for Pauli errors $\mathcal{A}$ and $\mathcal{B}$, mapped to vectors a and b over GF(4), equation 2.15 is equivalent to:

$$\langle a, b\rangle = \sum_{i=1}^{n} \langle a_i, b_i\rangle = 0, \tag{2.18}$$

where $\langle\cdot,\cdot\rangle$ denotes the trace Hermitian inner product over GF(4), which is defined as $\mathrm{tr}(a \cdot \bar{b})$, with $\bar{b}$ being the conjugate of b and $a \cdot \bar{b}$ representing the Hermitian inner product of a and b. The trace operation yields the following results: $\mathrm{tr}(\omega) = \mathrm{tr}(\bar{\omega}) = 1$ and $\mathrm{tr}(0) = \mathrm{tr}(1) = 0$. According to [9] it is also important to underline that two operators in the set of stabilizers commute if the inner product of their images, the vectors over GF(4), is either 1 or 0.

The matrix $\boldsymbol{S}$ is called *quaternary parity check matrix*. Each row of $\boldsymbol{S}$ (called *check*) represents a parity check, corresponding to one of the m stabilizers that generate the stabilizer group $\mathcal{S}$.
An $[[n, k, d]]$ stabilizer code is defined as a 2^k-dimensional subspace within the n-qubit Hilbert space $(\mathbb{C}^2)^{\otimes n}$. This subspace is characterized as the joint $+1$ eigenspace of $\mathcal{S}$.
We denote the normalizer of $\mathcal{S}$ as $\mathcal{N}(\mathcal{S})$. Furthermore, $\boldsymbol{S}^{\perp}$ represents the matrix containing vectors corresponding to the $2n - m$ generators of $\mathcal{N}(\mathcal{S})$. The minimum distance d of a code is defined as the smallest error weight in the set $\mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$.

A code is classified as degenerate if $\mathcal{S}$ contains a stabilizer with weight less than d. Moreover, a code is considered highly degenerate when d significantly exceeds the minimum weight of the stabilizers.

2.5 Quantum Error Correction

Quantum error correction is essential for the advancement of practical quantum computers. Quantum systems are inherently vulnerable to errors that arise from environmental noise. In the absence of effective QEC, even minor errors can accumulate quickly, leading to ineffective computations. To leverage the principles of quantum mechanics, it is crucial that qubits, the basic units of quantum information, are implemented with precision. Specifically, qubits must be realized using single particles or a small number of particles [9].

To manage errors in quantum computers, quantum communication plays a crucial role in that we can module the errors through a quantum channel. Thus, it is important to understand the basics of quantum communication. Quantum communication systems generally comprise three main components: a sender, a receiver, and a communication channel. The sender is responsible for preparing the quantum state to be transmitted. The receiver, on the other hand, measures the state to retrieve the transmitted information. The communication channel, which can be a fiber optic cable or a free-space channel, serves as the physical medium for transmitting the quantum state between the sender and receiver.

Quantum codes introduce two significant challenges:

- *Tanner Graphs and Small Loops*: Tanner graphs in quantum codes inherently contain small loops, which can undermine the performance of belief propagation algorithms.
- *Sparsity and Degeneracy*: Sparse quantum codes are, by definition, highly degenerate, posing additional challenges.

Another critical problem in quantum communication is the fragility of quantum states. Any interaction with the environment, such as noise or interference, can introduce errors and lead to loss of information. To mitigate this issue, quantum error correction techniques are used to protect quantum information from errors [10].

The fundamental principle underlying QEC is to safeguard encoded quantum information against errors that arise from two primary sources: uncontrolled interactions with the environment and imperfect implementations of quantum logical operations [11]. To achieve this protection, stabilizer codes are typically employed.

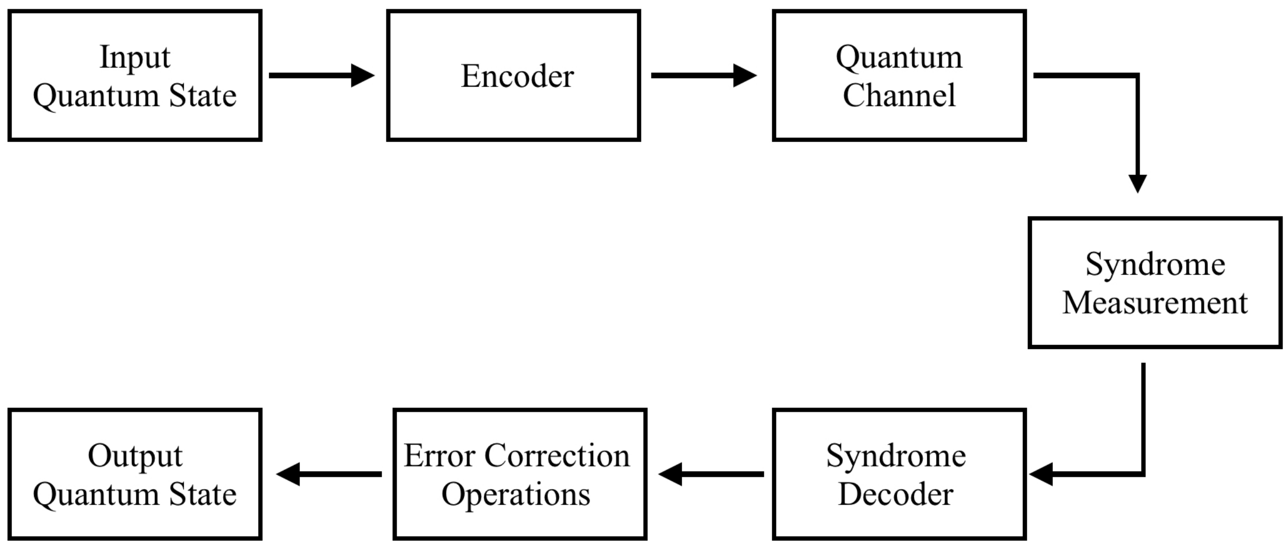


Figure 2.2: Block diagramm of QEC.

The quantum error correction process can be represented by the block diagram shown in Fig. 2.2, made by the following components:

1. *Input Quantum State:* This represents the initial quantum state or qubits that need to be protected from errors. It could be a logical qubit or a collection of physical qubits.
2. *Encoder:* The encoder transforms the input quantum state into an encoded state by applying a set of quantum gates and operations. Various quantum error correction codes, such as stabilizer codes or topological codes, can be used for encoding. Examples of encoding techniques include the application of Hadamard gates, controlled gates, or other unitary operations.
3. *Quantum Channel:* It describes the presence of noise and errors that can occur during the transmission or storage of the encoded state. It may include various types of errors, such as bit-flip errors, phase-flip errors, or more complex errors.
4. *Syndrome Measurements:* Syndrome measurements are performed on the encoded state to identify the presence and location of errors. These measurements are typically performed by applying a set of specific measurements or operations to certain qubits in the encoded state. The measurement outcomes provide information about the syndrome, which is used to identify the error type and location.
5. *Syndrome Decoder:* The syndrome decoder processes the measurement outcomes obtained from the syndrome measurements to determine the appropriate error correction operations. It employs decoding algorithms, such as maximum likelihood decoding or minimum weight decoding, to infer the most likely error configuration based on the syndrome information.
6. *Error Correction Operations:* Based on the decoding results, error correction operations are applied to the encoded state to reverse the effects of errors. These operations aim to restore the original quantum information by applying appropriate quantum gates or recovery operations.
7. *Output Quantum State:* The output quantum state represents the corrected version of the encoded state. It should ideally match the original input quantum state, thereby preserving the integrity of the quantum information.

It should be emphasized that the block diagram outlined above presents a broad conceptualization of a quantum error correction system. The actual architecture and execution can differ significantly, depending on the selected error correction methodology, the nature of the error model under consideration, and the particular experimental configuration employed. The main purpose of any error correction scheme is to accurately identify the most probable error $\mathcal{E}$ that has affected the system, given a specific set of syndrome measurements z [12]. This process of error inference and correction is essential in maintaining the integrity of quantum information in the face of environmental noise and system imperfections.

One of the primary obstacles in QEC is the development of efficient decoding algorithms that can swiftly identify and rectify errors. A widely-used method for this purpose is minimum-weight perfect matching decoding, which entails determining the minimum-weight match between the error syndrome and potential error locations.

QEC is a rapidly advancing field, presenting numerous challenges and opportunities for further investigation. A significant hurdle is the creation of QEC techniques capable of managing larger qubit systems and more intricate error models. Currently, there are promising opportunities to leverage QEC to devise powerful new quantum algorithms and technologies.

2.5.1 Quantum Error Correction with Stabilizer Codes

The process of error correction utilizing stabilizer codes is depicted in Fig. 2.3.

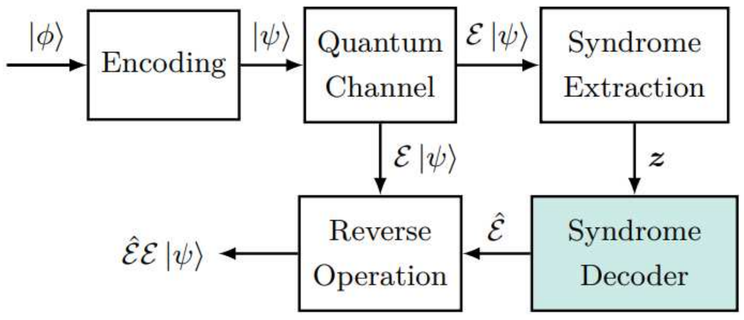


Figure 2.3: Block diagram of QEC using QSCs.

To protect a logical quantum state $|\phi\rangle$ consisting of k qubits, it is encoded into a state $|\psi\rangle$ comprising n qubits through the application of a stabilizer code. Noise that affects quantum memory can be represented as a depolarizing channel characterized by a depolarizing probability ϵ. Within this channel, the errors $\boldsymbol{X}$, $\boldsymbol{Z}$, and $\boldsymbol{Y}$ occur with equal likelihood, each with a probability of $\frac{\epsilon}{3}$ (more details are provided in Sect. 2.7).

In the event of a potentially corrupted state, the syndrome vector z is extracted and forwarded to a decoder, which seeks to identify the most likely error corresponding to the syndrome z. This study focuses on enhancing the syndrome decoder, specifically a quaternary belief propagation (BP4) decoder. The result of the decoding process is utilized as a reverse operation that is applied to the corrupted quantum state, with the aim of recovering the state $|\psi\rangle$.

Clarify Let $\hat{e}$ denote the estimate of the error e provided by the decoder, and let $\hat{\mathcal{E}}$ represent the associated Pauli error. The objective of the decoder is to find an $\hat{e}$ that produces the same syndrome z and satisfies the condition $\mathcal{E}\hat{\mathcal{E}} \in \mathcal{S}$. This condition can be verified using the equation

$$\langle (e + \hat{e}), \boldsymbol{S}_i^{\perp} \rangle = 0 \tag{2.19}$$

for each row $i \in \{1, 2, \ldots, 2n - m\}$ of $\boldsymbol{S}^{\perp}$.

From 2.19, we can observe that when correcting errors using a Quantum Stabilizer Code (QSC), four potential outcomes may occur:

- *Type I success:* The estimated error $\hat{e}$ is exactly the error that occurred, meaning $e + \hat{e} = 0$.
- *Type II success (with degeneracy):* The estimated error $\hat{e}$ is not exactly the same as e, but Eq. 2.19 is satisfied.

- *Type I failure (flagged):* The BP decoder is unable to find an $\hat{e}$ that matches the syndrome z.
- *Type II failure (unflagged):* The syndrome matches, but Eq. 2.19 is not satisfied. This results in an undetectable erroneous quantum state.

One approach to improve the decoding of QSC is to design a decoder that maximizes the probability of Type I success, adhering to classical decoding principles. However, this strategy often proves inadequate to achieve satisfactory decoding performance in QSCs. To develop a near-optimal decoder for quantum codes, it is essential to maximize the sum of the probabilities of Type I and Type II successes in order to correctly identify the exact error (Type I), while also accounting for degenerate cases, where the estimated error may not be exact but still corrects the state (Type II). This dual requirement highlights a significant challenge, as no effective decoding algorithm currently exists that systematically utilizes degeneracy.

This limitation has led researchers to investigate the potential of neural network (NN) decoders. The advantage of NN decoders lies in their ability to learn complex patterns and relationships within data, which allows them to uncover and leverage degeneracy in ways that traditional algorithms cannot.

2.6 QLDPC Codes

Quantum low-density parity-check codes are among the most promising types of quantum error correction codes due to their practical benefits. To understand QLDPC codes, we first need to define classical low-density parity-check codes.

Low-density parity-check codes are a family of linear block codes characterized by the fact that both row and column weights of $\boldsymbol{H}$ are bounded by a relatively small constant, independent of the block length. This results in a sparse parity check matrix $\boldsymbol{H}$. These codes are typically represented using a bipartite or Tanner graph [13] (more details about Tanner Graph are explained in 3.2).

It is well established that the performance of LDPC codes under belief propagation decoding deteriorates when short cycles are present in their Tanner graph [14]. A cycle is defined as a path in the graph that returns to its starting point, as highlighted in Fig. 2.4, where the Tanner graph of the (7,4) Hamming code is shown [15]. Consequently, when designing an LDPC code, a primary objective is to maximize the girth, which represents the length of the shortest cycle in the graph.

To satisfy the constraint $\boldsymbol{H}'_X \boldsymbol{H}'^T_Z = 0$, the parity check matrix $\boldsymbol{H}$ of the dual-containing Calderbank-Shor-Steane code must have rows that overlap with each other at an even number of positions[3] . This requirement inherently leads to a Tanner graph with numerous small loops (of length 4). To improve the code properties, one can consider relaxing the dual-containing CSS constraint.

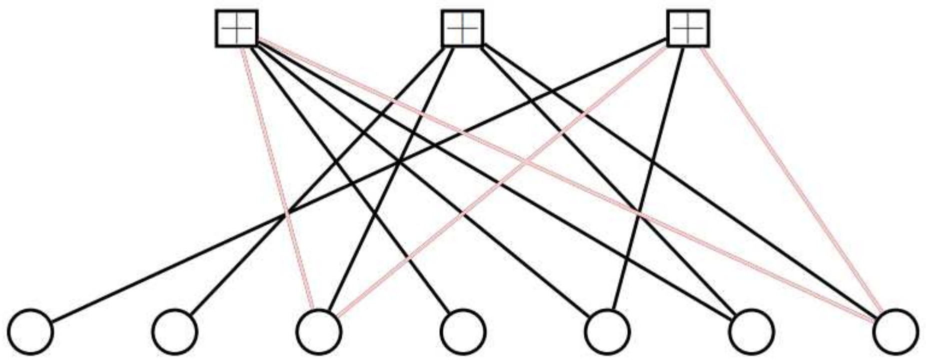


Figure 2.4: Tanner graph of the (7, 4) Hamming code with highlighted cycle.

QLDPC codes are a quantum analogue of classical LDPC codes. They are defined by a sparse parity-check matrix, where each row corresponds to a stabilizer operator that checks the parity of a subset

[3]It is important to underling that this is true for any non-trivial stabilizer code, not just dual-containing CSS codes.

of qubits. The sparsity of the matrix is crucial for efficient error correction, as it reduces the number of measurements needed to detect and correct errors.

The design of high-performance QLDPC codes is much more complicated than in the classical setting. This complexity arises from several factors, the most notable being that the parity-check matrix of quantum LDPC codes must satisfy certain orthogonality constraints originating from the fact that the stabilizes must mutually commute with each other [6]. The performance of QLDPC codes depends significantly on the choice of the parity-check matrix. One method to construct a QLDPC code is to randomly generate a sparse parity-check matrix with specific properties, such as regularity, where each row and column have a fixed number of nonzero entries.

One advantage of QLDPC codes is their potential for efficient error correction. Unlike some other quantum error-correcting codes, QLDPC codes do not require complex decoding algorithms and can be decoded using simple belief propagation algorithms. This makes QLDPC codes a promising candidate for practical implementations of quantum error correction.

QLDPC codes also hold potential for fault-tolerant quantum computation. Fault-tolerant quantum computation is a quantum computation that remains resilient to errors and can be performed with high accuracy. QLDPC codes have been shown to have a relatively high threshold for fault-tolerant quantum computation, meaning that they can tolerate a certain amount of noise before the computation fails.

2.7 Error Model

In this section, we provide a concise explanation of the effects of measurement and how a quantum channel can be modeled to apply classical decoding techniques in a quantum context.

The measurement of quantum information imposes constraints on the use of various decoding algorithms. As discussed in previous sections, a quantum bit , unlike a classical bit, can exist in a superposition of states, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and when the qubit is measured, it collapses to the state $|0\rangle$ with probability $|\alpha|^2$ and to $|1\rangle$ with probability $|\beta|^2$. To address this, one typically measures the effect of the error in the form of a syndrome and applies a correction to the corrupted quantum states.

Another significant aspect of the quantum channel, which imposes strict constraints on the decoding scheme, is known as *decoherence* and is described as the destructive interaction between the environment and the qubit [16]. In a quantum channel, this phenomenon, also known as quantum noise, is the primary obstacle to the practical implementation of a quantum communication system [17].

Finally, it is important to underline that the error model is highly dependent on the hardware. Typically, a standard model known as the depolarizing channel is used, which will also be employed in this study.

2.7.1 Depolarizing Channel

A commonly used model is the depolarizing channel with error probability ϵ. In this model, a quantum state of n qubits is subject to a random Pauli error:

$$\mathcal{E} = \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n \tag{2.20}$$

where each $\mathcal{E}_i \in \{\mathbb{I}, \boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Y}\}$ independently affects a single qubit, potentially causing a bit-flip, a phase-flip, or both, with equal probabilities:

$$P(\mathcal{E}_i = \boldsymbol{X}) = P(\mathcal{E}_i = \boldsymbol{Z}) = P(\mathcal{E}_i = \boldsymbol{Y}) = \frac{\epsilon}{3} \tag{2.21}$$

The depolarizing channel, characterized by the probability of error ϵ, assumes that the errors $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ occur with the same probability $\frac{\epsilon}{3}$. Consequently, this channel is isomorphic to a *4-ary symmetric channel* [6].

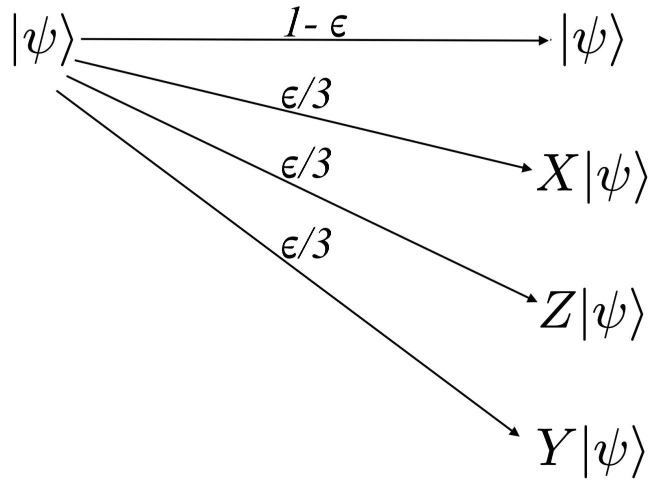


Figure 2.5: The likelihood of a state transition in a depolarizing channel with a given probability ϵ.

2.8 Quantum Degeneracy

In traditional coding theory, distinct error vectors applied to a single codeword result in unique received words. However, quantum communication systems exhibit a phenomenon known as *degeneracy*, where two different errors acting on the same quantum state $|\psi\rangle$ can produce an identical (corrupted) state $|\phi\rangle$. Specifically, for a quantum state $|\phi\rangle$ within the +1 eigenspace of a stabilizer code (i.e., a codeword), two operators $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ will generate the same quantum state if they differ only by a stabilizer:

$$\mathcal{E}_\alpha = \mathcal{E}_\beta S_i \implies \mathcal{E}_\alpha|\phi\rangle = \mathcal{E}_\beta S_i|\phi\rangle = \mathcal{E}_\beta|\phi\rangle. \tag{2.22}$$

Degeneracy has significant implications for quantum error correction. Consider a scenario in which a quantum state $|\psi\rangle$ has been transformed to $|\phi\rangle = \mathcal{E}_\alpha|\psi\rangle$. As discussed previously, to restore the system to state $|\psi\rangle$, it suffices to apply the same operator $\mathcal{E}_\alpha$ again:

$$\mathcal{E}_\alpha|\phi\rangle = \mathcal{E}_\alpha\mathcal{E}_\alpha|\psi\rangle = |\psi\rangle.$$

Interestingly, due to degeneracy, the quantum system can also be returned to its original state $|\psi\rangle$ by applying any operator $\mathcal{E}_\beta = S_i\mathcal{E}_\alpha$, where S_i is any stabilizer. In fact, for a given stabilizer code, all possible error patterns can be categorized into equivalence classes or cosets. A coset comprises a set of error operators that differ only by a stabilizer. Consequently, any error operator within the same coset, when applied to $|\psi\rangle$, results in the same state. Thus, given $\mathcal{E}_j|\psi\rangle$ and $\mathcal{E}_i$ as an error operator of the coset $\mathcal{G}$:

$$\forall \mathcal{E}_i \in \mathcal{G} : \mathcal{E}_i|\psi\rangle = \mathcal{E}_j|\psi\rangle. \tag{2.23}$$

The degeneracy of the toric code adds complexity to the problem of optimal decoding compared to classical codes. In classical codes, the goal is to find the most likely error that occurred during transmission. However, in the toric code, due to its degeneracy, there are multiple errors that lead to the same syndrome. Therefore, the objective shifts from finding the most likely error to finding the most likely coset [18].

2.9 Toric Code

In this section, we employ the stabilizer formalism, previously discussed, to explain the concept of the toric code, $\mathcal{C}_{\text{toric}}$. Alexei Kitaev introduced this quantum error correction scheme in 1998. Its significance stems from its potential as the primary code for initial quantum computing systems. The toric code derives its name from its implementation on a lattice with periodic boundary conditions, effectively forming a torus.

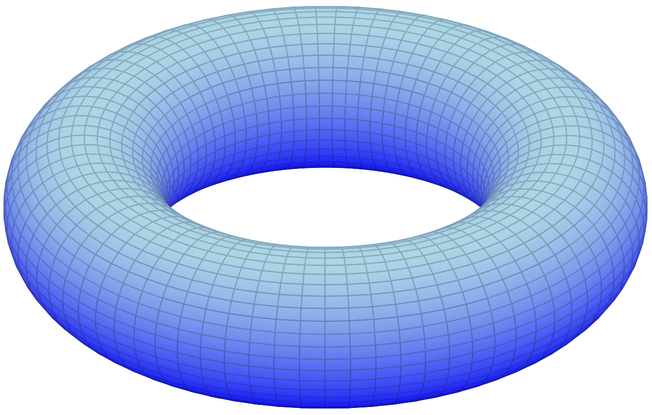


Figure 2.6: The fundamental structure of the toric code is based on a lattice with cyclical boundaries, effectively mapping the lattice onto the surface of a torus.

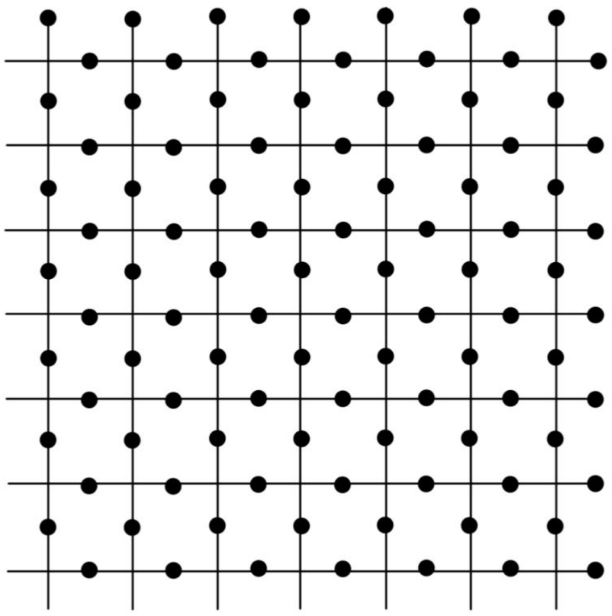


Figure 2.7: A representation of the toric grid used in the toric code, illustrating the arrangement of qubits and the connections between them.

As a stabilizer code, it is characterized by two types of stabilizer operators:

1. Vertex operators ($\boldsymbol{A}_X$):
$$\boldsymbol{A}_X \equiv \boldsymbol{X}_u \otimes \boldsymbol{X}_d \otimes \boldsymbol{X}_l \otimes \boldsymbol{X}_r$$

2. Plaquette operators ($\boldsymbol{B}_Z$):
$$\boldsymbol{B}_Z \equiv \boldsymbol{Z}_u \otimes \boldsymbol{Z}_d \otimes \boldsymbol{Z}_l \otimes \boldsymbol{Z}_r$$

Here, u, d, l, and r represent the "up", "down", "left", and "right" qubits, respectively, surrounding a vertex or plaquette. The stabilizer operators $\boldsymbol{A}_X$ and $\boldsymbol{B}_Z$ are defined for each vertex (cross) and face (plaquette) of the lattice, respectively (illustrated in Fig. 2.8).

A key feature of the toric code is the locality of its stabilizers, which contrasts with the non-local nature of stabilizers in other quantum error correction schemes, such as the Shor code [8]. This locality can be a big advantage since in some quantum computing platforms, such as those based on superconducting qubits, it is difficult to implement long-range interactions between far-away qubits.

For a lattice of dimensions $L \times L$, the toric code employs:

- $n = 2L^2$ physical qubits
- L^2 plaquette operators of weight 4
- L^2 cross operators of weight 4

This structure provides a scalable framework for quantum error correction, with the code distance increasing linearly with the lattice size.

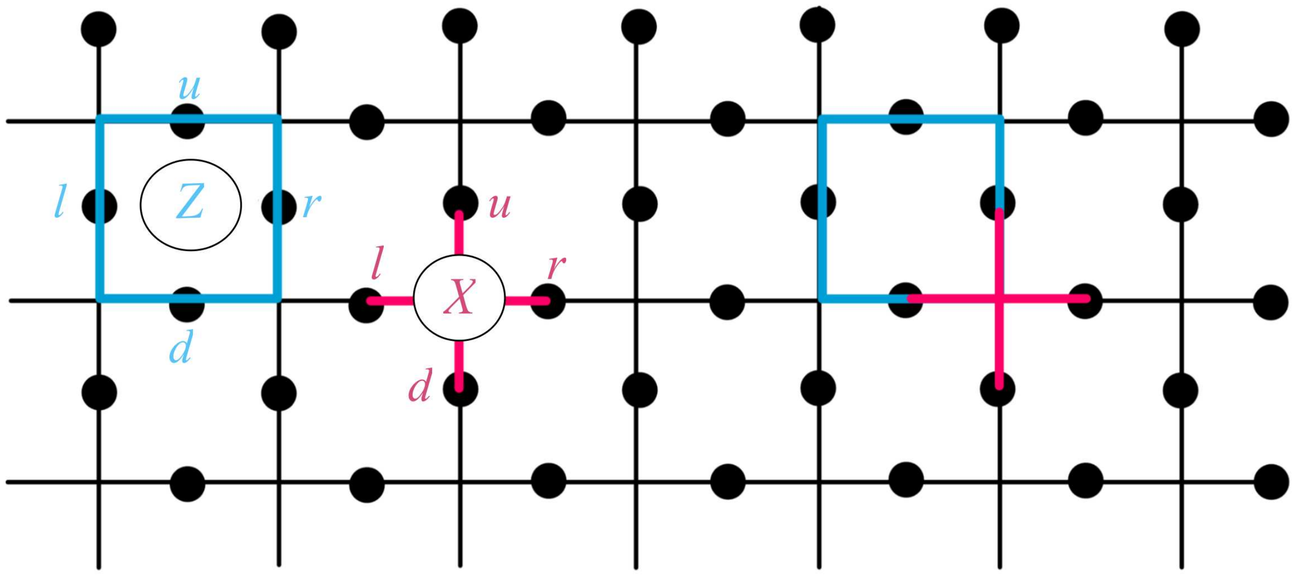


Figure 2.8: In the toric code, both Z operators (plaquettes) and X operators (cross) function as stabilizers. When a face operator intersects with a vertex operator, their interaction consistently involves two qubits.

We can easily verify that the stabilizers of the toric code commute. The cases $[\boldsymbol{A}_X, \boldsymbol{A}_{X'}] = 0$ and $[\boldsymbol{B}_Z, \boldsymbol{B}_{Z'}] = 0$ are straightforward because $\boldsymbol{Z}^2 = \boldsymbol{X}^2 = \mathbb{I}$. The case $[\boldsymbol{A}_X, \boldsymbol{B}_Z] = 0$ is also trivial if $\boldsymbol{A}_X$ and $\boldsymbol{B}_Z$ do not overlap. When they do overlap, the operators $\boldsymbol{A}_X$ and $\boldsymbol{B}_Z$ coincide on two qubits, causing the phase produced by $\boldsymbol{XZ} = -\boldsymbol{ZX}$ to cancel out (see Fig. 2.8).

The multiplication of two plaquettes can be easily understood through illustrations (see Fig. 2.9). Consider two overlapping plaquettes, $\boldsymbol{B}_Z$ and $\boldsymbol{B}_{Z'}$, where the right qubit of the first plaquette is the left qubit of the second plaquette. On this qubit, two $\boldsymbol{Z}$ operators are applied, one from each plaquette. However, since $\boldsymbol{Z}^2 = I$, the identity is effectively applied on this qubit. This results in a plaquette composed of six $\boldsymbol{Z}$ operators (see Fig. 2.9), which is also a stabilizer. Note that the multiplication of plaquettes will always produce open strings. We will primarily discuss the plaquette operators, but the discussion for cross operators can be conducted analogously.

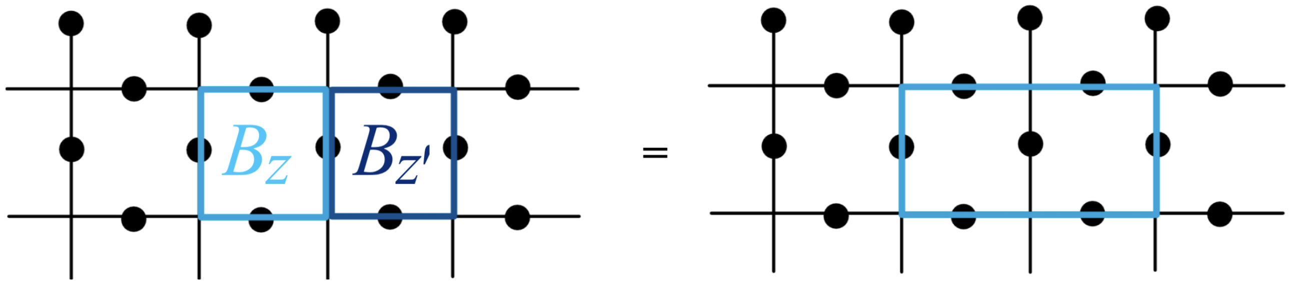


Figure 2.9: The combination of two elementary surface elements, $\boldsymbol{B}_Z$ and $\boldsymbol{B}_{Z'}$, through multiplication results in an expanded surface element. This larger element also maintains the stabilizing properties inherent to the toric code structure.

In the section on the stabilizer formalism, we discussed that the stabilizer generators must be linearly independent. We can easily verify that the cross and plaquette operators, $\boldsymbol{A}_X$ and $\boldsymbol{B}_Z$, are not linearly independent since

$$\prod_{\text{all crosses}} \boldsymbol{A}_X = \mathbb{I} \quad \text{and} \quad \prod_{\text{all plaquettes}} \boldsymbol{B}_Z = \mathbb{I}.$$

To obtain a set of independent stabilizers for the toric code, we simply need to remove one plaquette and one cross. Consequently, the toric code has

$$m = 2L^2 - 2$$

independent stabilizers. The number of encoded qubits (logical qubits) in a stabilizer code is given by

$$k = n - m.$$

For the toric code, this results in $k = 2$ logical qubits.

To complete the characterization of the code space $\mathcal{C}_{\text{toric}}$, we need to identify the logical operators. Recall that these operators must commute with the stabilizers without being stabilizers themselves. Logical operators can be either a product of $\boldsymbol{Z}$ or a product of $\boldsymbol{X}$. We have observed that multiplying plaquettes results in loops of different sizes, which are also stabilizers, but they never form open strings.

Consider a product of $\boldsymbol{Z}$ along an entire horizontal string of the lattice, i.e., $\overline{\boldsymbol{Z}}_1 \equiv \boldsymbol{Z}_1 \otimes \cdots \otimes \boldsymbol{Z}_L$. It commutes with $\boldsymbol{B}_Z$ and $\boldsymbol{A}_X$ for the same reasons that have implied $[\boldsymbol{A}_X, \boldsymbol{A}_{X'}] = [\boldsymbol{B}_Z, \boldsymbol{B}_{Z'}] = [\boldsymbol{A}_X, \boldsymbol{B}_Z] = 0$ (see Fig. 2.10). This is also true for a product of $\boldsymbol{Z}$ along an entire vertical string, $\overline{\boldsymbol{Z}}_2$, and for a product of $\boldsymbol{X}$ along an entire horizontal and vertical string, $\overline{\boldsymbol{X}}_1$ and $\overline{\boldsymbol{X}}_2$ (see Fig. 2.10). Note further that $\{\overline{\boldsymbol{Z}}_i, \overline{\boldsymbol{X}}_i\} = 0$ and $[\overline{\boldsymbol{Z}}_i, \overline{\boldsymbol{X}}_j] = 0$ for $i \neq j$ and $i, j = 1, 2$.

Defining $\{|\bar{0}_1\rangle, |\bar{1}_1\rangle\}$ as the eigenvectors of $\overline{\boldsymbol{Z}}_1$, we can easily verify that the logical operators satisfy:

$$\overline{\boldsymbol{X}}_1|\bar{0}_1\rangle = |\bar{1}_1\rangle, \quad \overline{\boldsymbol{Z}}_1|\bar{0}_1\rangle = |\bar{0}_1\rangle,$$

$$\overline{\boldsymbol{X}}_1|\bar{1}_1\rangle = |\bar{0}_1\rangle, \quad \overline{\boldsymbol{X}}_1|\bar{1}_1\rangle = -|\bar{1}_1\rangle.$$

We find analogous equalities for $\overline{\boldsymbol{X}}_2$ and $\overline{\boldsymbol{Z}}_2$ as shown in Fig. 2.10.

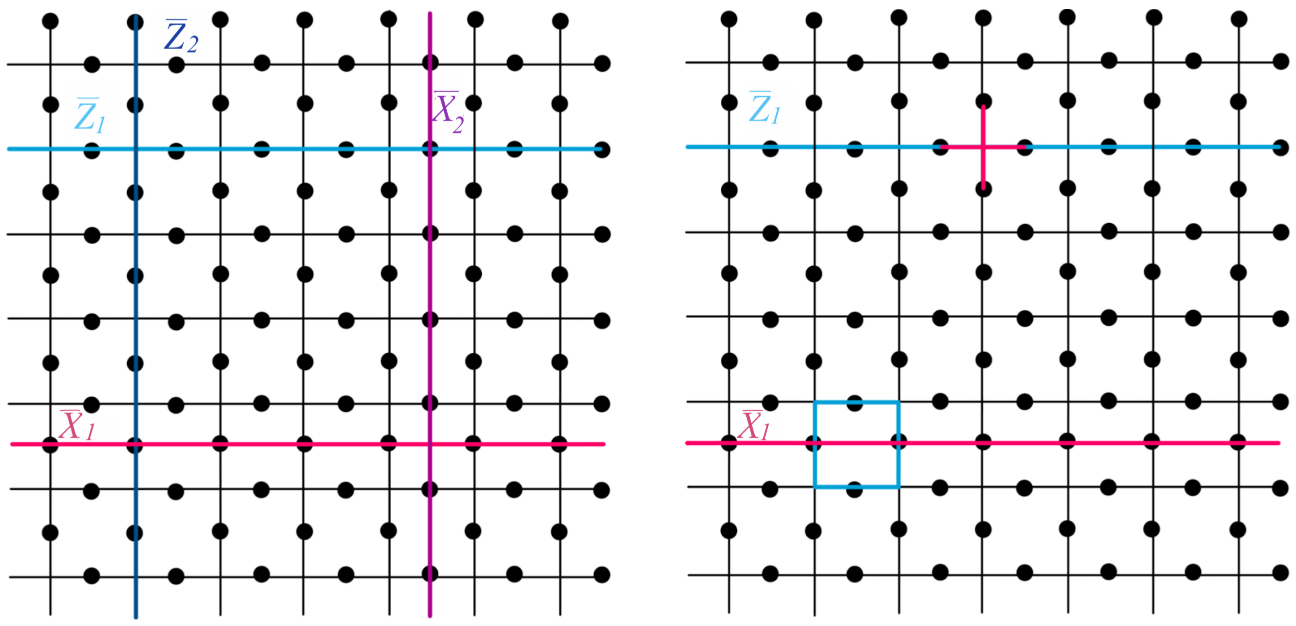


Figure 2.10: Fundamental operators in the toric code framework. The logical operators $\overline{\boldsymbol{Z}}_1$ and $\overline{\boldsymbol{Z}}_2$ (as well as $\overline{\boldsymbol{X}}_1$ and $\overline{\boldsymbol{X}}_2$) intersect with either a vertex or plaquette operator on precisely two qubits.

Recall from Sect. 2.3 that logical operators do not have a unique representation. In fact, we can obtain a new logical operator by multiplying any operator $\overline{\boldsymbol{X}}_1$, $\overline{\boldsymbol{X}}_2$, $\overline{\boldsymbol{Z}}_1$, $\overline{\boldsymbol{Z}}_2$ by any stabilizer operator, i.e., by any plaquette or cross. In the toric code, this property translates to the fact that $\overline{\boldsymbol{Z}}_1$, which is a straight line, can be stretched in various ways and still represent the same logical operator (see Fig. 2.11). Note that the same is true for $\overline{\boldsymbol{X}}_1$, $\overline{\boldsymbol{X}}_2$, $\overline{\boldsymbol{Z}}_1$, and $\overline{\boldsymbol{Z}}_2$. Therefore, the logical subspace is the subspace spanned by all strings with "the same topology".

The distance of a stabilizer code is defined as the weight of the minimal representation of logical operators, and for the toric code we have $d_{\text{Toric}} = L$.

In summary, we can characterize the toric code as a $[2L^2, 2, L]$ code. It is important to note that the toric code has a topological nature because its encoded information is defined by objects that exist solely due to the topology of the space, i.e., the torus.

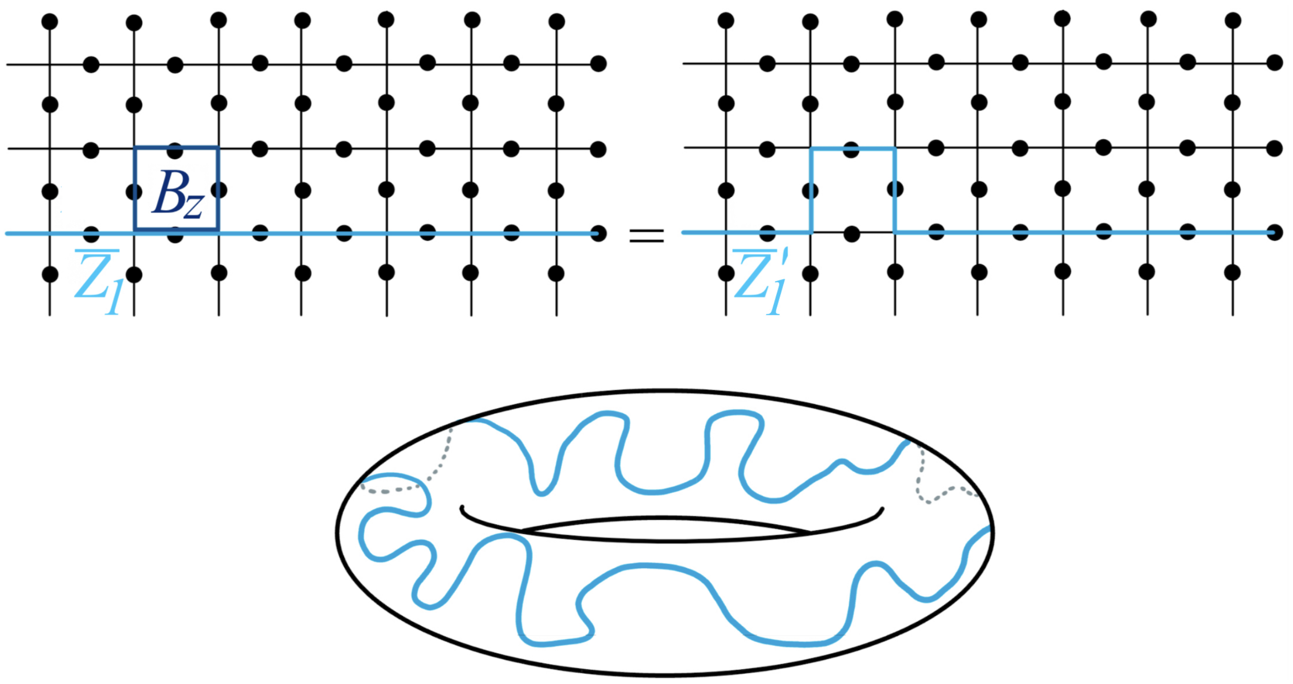


Figure 2.11: The product of a face operator $\boldsymbol{B}_Z$ and a logical operator, such as $\overline{\boldsymbol{Z}}_1$, yields an alternative representation of the same logical operator. Consequently, any closed loop encircling the torus constitutes a valid logical operator.

2.9.1 Overcomplete Check Matrices

Overcomplete check matrices are distinguished by the inclusion of supplementary redundant rows in addition to the initial full-rank check matrix. The quaternary belief propagation algorithm can be implemented on the Tanner graph corresponding to these overcomplete check matrices. This technique is designated as BP4 utilizing an overcomplete check matrix (OBP4) decoding. In this subsection, we explain two possible ways to construct overcomplete check matrices and then investigate the reason for the improvement in performance of OBP4 [2].

To generate additional parity checks, we consider $\boldsymbol{H}_X$ and $\boldsymbol{H}_Z$ as distinct binary matrices. The process of creating overcomplete check matrices is equivalent to identifying numerous low-weight stabilizers, and there are different ways to do this.

The first approach involves a comprehensive search, which is generally computationally intensive. However, probabilistic techniques can be leveraged to enhance the likelihood of discovering low-weight stabilizers during the exhaustive search.

The alternative approach uses the inherent features of topological codes. For example, as illustrated in Fig. 2.12, merging two adjacent stabilizers in a toric code produces a new stabilizer of weight 6. This phenomenon occurs because each toric code stabilizer is supported by four qubits, with neighboring stabilizers sharing one qubit. Consequently, for every toric code, we can construct $2n$ redundant $\boldsymbol{X}$ stabilizers and $2n$ redundant $\boldsymbol{Z}$stabilizers, each of weight 6. When combined with the n weight-4 stabilizers, we obtain an overcomplete check matrix of dimensions $3n \times n$ for toric codes, which is utilized in this study.

The values of redundant syndromes can be determined without additional syndrome extraction operations. Let $\boldsymbol{H}$ represent either $\boldsymbol{H}_X$ or $\boldsymbol{H}_Z$. The parity-check matrix $\boldsymbol{H}_{oc}$ with redundant rows is obtained by $\boldsymbol{H}_{oc} = \boldsymbol{M}\boldsymbol{H}$, where $\boldsymbol{M}$ is a binary matrix of size $m_{oc} \times m$. The original syndrome is calculated as $\boldsymbol{H}e^T = z$. Subsequently, the new syndrome associated with $\boldsymbol{H}_{oc}$ is given by:

$$z_{oc} = \boldsymbol{H}_{oc}e^T = \boldsymbol{M}\boldsymbol{H}e^T = \boldsymbol{M}z \tag{2.24}$$

This represents a linear transformation of z defined by $\boldsymbol{M}$.

The concept of performing belief propagation decoding using an overcomplete parity-check matrix has been explored in prior works [2] for classical linear codes. One of the primary motivations was to enable more parallel node updates, thereby mitigating the impact of short cycles. Interestingly, this method significantly enhances the decoding performance of QLDPC codes, regardless of whether the number of iterations is limited or unlimited. This contrasts with classical linear codes, where using overcomplete check matrices primarily accelerates convergence and does not yield noticeable improvements with a high number of iterations.

For QLDPC codes, this technique is similar to matrix augmentation, where some rows of the check matrix are duplicated. The benefit is that the messages associated with duplicated check nodes are amplified, aiding in breaking the symmetry during decoding and leading to a more accurate error estimate [2].

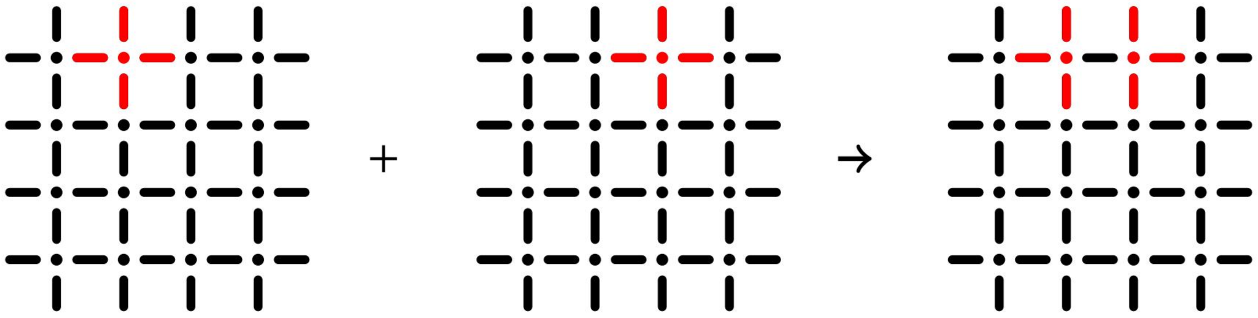


Figure 2.12: Example of the construction of a weight-6 stabilizer for the toric code with $L = 4$. The red edges are the qubits which support an $\boldsymbol{X}$ stabilizer.

2.10 Minimum Weight Perfect Matching

The Edmonds algorithm, also known as the perfect minimum weight matching algorithm, functions by establishing pairs from all measurements of the syndrome [8]. This decoding procedure can be formally expressed as:

$$\text{MWPM} =_{\text{matching } \mathcal{Q}} \sum_{(i,j)\in\mathcal{Q}} d(i,j) \tag{2.25}$$

where $d(i, j)$ represents the distance between syndromes i and j.

The (worst-case) computational complexity of this decoding method is given by:

$$\text{Time Complexity} = \mathcal{O}(n^3)$$

where n denotes the number of syndrome measurements. This cubic time complexity makes this algorithm impractical for large topological codes.

Chapter 3

Classical Decoding Algorithms

3.1 Graph-Based Decoding of LDPC Codes

Graph-based decoding is a widely utilized technique for interpreting low-density parity-check codes. An LDPC code is defined by a parity-check matrix that possesses specific characteristics: each column contains a small, fixed number $j > 3$ of ones, and each row contains a small, fixed number $k > j$ of ones [19]. This method employs the Tanner graph representation of the code to create a factor graph, which aids in the iterative decoding process.

In the graph-based decoding algorithm, messages are exchanged between variable nodes and check nodes within the factor graph. These messages convey information regarding the probability that a bit is 0 or 1, based on the values of the other bits connected to it at the respective check or variable nodes.

Research has demonstrated that graph-based decoding is effective in decoding LDPC codes, particularly for those with large block sizes. Moreover, the efficacy of graph-based decoding can be enhanced through the application of advanced techniques such as neural belief propagation.

3.2 Tanner Graph

The belief propagation decoding process for QLDPC codes is carried out on the Tanner graph associated with the code. This Tanner graph is a bipartite structure consisting of two distinct types of vertices. The first type comprises variable nodes (VNs), each representing a code bit (or qubit) and corresponding to a column in the check matrix $\boldsymbol{S}$. The second type consists of check nodes (CNs), each linked to a specific check and corresponding to a row in the check matrix.

A variable node v_i is connected to a check node c_j if the corresponding entry $S_{j,i} \neq 0$, where $S_{j,i} \in \mathrm{GF}(4)\backslash\{0\}$ denotes the coefficient of the edge between them. For instance, consider the [[7, 1, 3]] quantum Bose–Chaudhuri–Hocquenghem (BCH) code, where both $\boldsymbol{H}_{X'}$ and $\boldsymbol{H}_{Z'}$ represent the parity-check matrix of a [7, 4, 3] BCH code:

$$\boldsymbol{H}_{\mathrm{BCH}} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The check matrix $\boldsymbol{S} \in \mathrm{GF}(4)^{6\times 7}$ can be represented as follows:

$$\boldsymbol{S} = \begin{pmatrix} \omega H_{\mathrm{BCH}} \\ \overline{\omega} H_{\mathrm{BCH}} \end{pmatrix} \tag{3.1}$$

The Tanner Graph of BCH code is shown in Fig. 3.1.

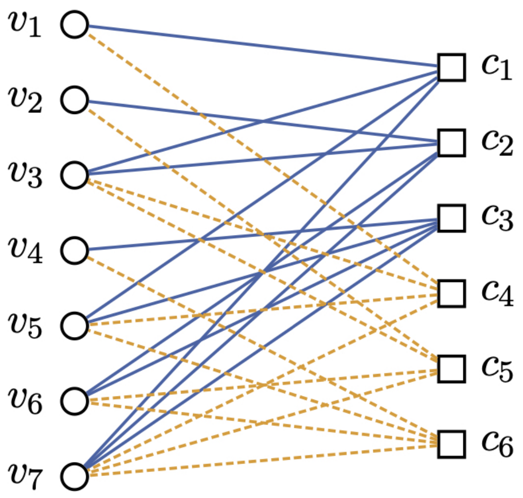


Figure 3.1: The Tanner graph of the [[7,1,3]] quantum BCH code where VNs are represented by circles and CNs by squares. Blue solid edges indicate coefficients ω, while yellow dashed edges represent coefficients $\overline{\omega}$.

In accordance with the principles of belief propagation algorithms, messages are naturally transmitted between the check nodes and the variable nodes. Specifically, messages flow from CNs to VNs and from VNs back to CNs, facilitating the iterative decoding process [20].

The messages are iteratively refined until a valid codeword is produced or a predefined maximum number of iterations is reached. This decoding process can be interpreted as a type of belief propagation, where each node adjusts its belief based on the beliefs of its neighboring nodes within the Tanner graph.

The configuration of the Tanner graph is dictated by the parity-check matrix associated with the code. The rows of this matrix represent the parity-check equations, while the columns correspond to the individual code bits. A nonzero entry in the matrix signifies that the associated variable is involved in the respective parity-check equation.

Tanner graphs serve as a valuable framework for evaluating the performance of a code and for the development of new codes. The minimum distance of a code can be assessed by analyzing the cycles present in the Tanner graph. A cycle of length h in the graph corresponds to a collection of h equations that share a nontrivial common solution, indicating that the code can accommodate at least h errors in a codeword.

3.3 LDPC Decoding: Quaternary Belief Propagation

The fundamental concept of belief propagation involves the exchange of messages. This algorithm iteratively refines messages exchanged between nodes in the Tanner graph through local computations. These messages encapsulate beliefs or probabilities linked to each variable, providing insights into the system's state. Updates occur in a parallel and distributed fashion, enabling efficient processing of large-scale graphical models.

In the realm of quantum error correction, the objective is to safeguard the information encoded within a quantum state from potential errors during computation or communication. Directly measuring a qubit's state would destroy its fragile quantum information, rendering error correction impossible. Instead, we measure error syndrome, which involves a series of measurements on multiple qubits. This approach reveals the type and location of errors without disrupting the encoded quantum state.

In this thesis, the syndrome decoder depicted in Fig. 2.3 is the quaternary BP4 decoder. Unlike binary decoding, which is less effective and exhibits a higher error floor, the BP4 decoder considers

the correlation between $\boldsymbol{X}$ and $\boldsymbol{Z}$ errors. The primary drawback of the traditional BP4 decoder is its high complexity, resulting from the transmission of vector messages rather than scalar messages, as is done in BP2 (Binary Belief Propagation). This issue is addressed by the newly introduced refined BP4 decoder, which utilizes scalar messages. In particular, we employ the log-domain refined BP4 decoder discussed in [21, 22]. This decoder takes advantage of the binary nature of the syndrome in a stabilizer code, which denotes whether the error commutes with the stabilizer, thus facilitating scalar message passing.
We will now provide a brief overview of the algorithm.

For each variable node v_i, where $i \in \{1, 2, \ldots, n\}$, we start by setting up the log-likelihood ratio (LLR) vector $\boldsymbol{\Gamma}_{i\to j}$ as $\boldsymbol{\Lambda}_i = (\Lambda_i^{(1)}\ \Lambda_i^{(\omega)}\ \Lambda_i^{(\overline{\omega})}) \in \mathbb{R}^3$ with

$$\Lambda_i^{(\zeta)} = ln\left(\frac{P(e_i = 0)}{P(e_i = \zeta)}\right) = ln\left(\frac{1-\epsilon_0}{\frac{\epsilon_0}{3}}\right)$$

where $\zeta \in \mathrm{GF}(4) \setminus \{0\}$ and ϵ_0 is the estimated physical error probability of the channel. To exchange scalar messages, a *belief-quantization operator* $\lambda_\eta : \mathbb{R}^3 \to \mathbb{R}$ is defined as

$$\lambda_\eta(\boldsymbol{\Lambda}_i) = ln\left(\frac{P(\langle e_i, \eta\rangle = 0)}{P(\langle e_i, \eta\rangle = 1)}\right) = ln\left(\frac{1 + e^{-\Lambda_i^{(\eta)}}}{\sum_{\zeta\neq 0, \zeta\neq\eta} e^{-\Lambda_i^{(\zeta)}}}\right)$$

The operator λ_η transforms the LLR vector into a scalar LLR associated with the binary random variable $\langle e_i, \eta\rangle$, where η iterates over the nonzero elements of $\boldsymbol{S}$. The initial scalar messages for the variable nodes are computed as

$$\lambda_{i\to j} := \lambda_{S_{j,i}}(\boldsymbol{\Gamma}_{i\to j}) \tag{3.2}$$

and are passed to the neighboring CNs where $i \to j$ denotes the message from VN v_i to CN c_j.

The outgoing messages from check node c_j $j \in \{1, 2, \ldots, m\}$ are determined by

$$\Delta_{i\leftarrow j} = (-1)^{z_j} \cdot \underset{i'\in\mathcal{N}(j)\setminus\{i\}}{\boxplus} \lambda_{i'\to j} \tag{3.3}$$

where $\mathcal{N}(j)$ represents the indices of the neighboring VNs of c_j, and the $\boxplus$ operation is defined as

$$\overset{I}{\underset{i=1}{\boxplus}} x_i := 2\tanh^{-1}\left(\prod_{i=1}^{I} \tanh\frac{x_i}{2}\right).$$

During the variable node update, we begin by computing the log-likelihood ratio vector $\boldsymbol{\Gamma}_{i\to j} = (\Gamma_{i\to j}^{(1)}\ \Gamma_{i\to j}^{(\omega)}\ \Gamma_{i\to j}^{(\overline{\omega})})$ with

$$\Gamma_{i\to j}^{(\zeta)} = \Lambda_i^{(\zeta)} + \sum_{j'\in\mathcal{M}(i)\setminus\{j\}, \langle\zeta, S_{j',i}\rangle = 1} \Delta_{i\leftarrow j'} \tag{3.4}$$

for all $\zeta \in \mathrm{GF}(4) \setminus \{0\}$, where $\mathcal{M}(i)$ indicates the indices of the neighboring check nodes of the variable node v_i. Subsequently, the outgoing messages $\lambda_{i\to j} = \lambda_{S_{j,i}}(\boldsymbol{\Gamma}_{i\to j})$ are computed and sent to the neighboring CNs.

To assess the error, a hard decision is made at the variable nodes by computing $\boldsymbol{\Gamma}_i$ for $i \in \{1, 2, \ldots, n\}$ using the formula

$$\Gamma_i^{(\zeta)} = \Lambda_i^{(\zeta)} + \sum_{j\in\mathcal{M}(i), \langle\zeta, S_{j,i}\rangle = 1} \Delta_{i\leftarrow j} \tag{3.5}$$

for all $\zeta \in \mathrm{GF}(4) \setminus \{0\}$. If all $\Gamma_i^{(\zeta)} > 0$, then $\hat{e}_i = 0$, otherwise $\hat{e}_i = \arg\min_\zeta \Gamma_i^{(\zeta)}$.

The iterative procedure continues until either the maximum number of iterations R is achieved or the syndrome is satisfied. In Fig. 3.4 the node processor schema is shown for one iteration.

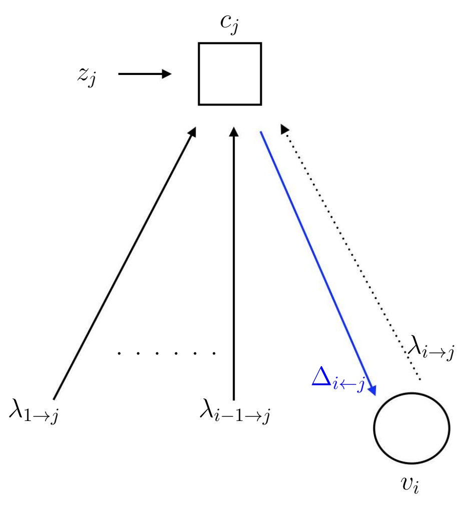


Figure 3.2: CN decoder

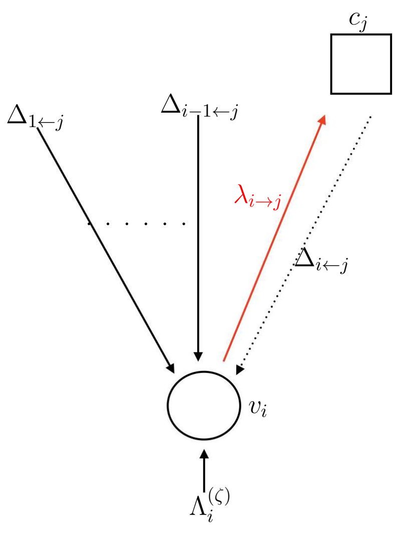


Figure 3.3: VN decoder

Figure 3.4: Node processors

Chapter 4

Neural Networks & Deep Learning Techniques

Recent advancements in machine learning, particularly through the use of neural networks, have opened new possibilities for improving quantum error correction methods. Neural networks possess the capability to learn complex patterns and relationships within data, making them well-suited for tasks such as syndrome decoding and error prediction in quantum systems. For instance, studies have demonstrated that artificial neural networks can effectively decode syndrome measurement data from quantum error correcting codes, providing a promising alternative to traditional decoding methods [23] [24]. In particular, neural belief propagation has proven to be effective in enhancing the decoding performance of classical linear codes without increasing latency [23]. The application of NBP to improve the decoding of quantum stabilizer codes has been explored in [7] for the BP2 decoder, which processes X and Z errors independently. In this chapter, we detail the extension of the NBP framework to the BP4 decoder by incorporating a modified loss function designed to leverage degeneracy, as described in [2]. The proposed NBP model can be utilized with both the original check matrix and the overcomplete check matrix, resulting in a Quaternary Neural Belief Propagation (NBP4) decoder and a NBP4 utilizing an overcomplete check matrix (ONBP4) decoder, respectively.
Moreover, in this chapter we also describe some deep learning techniques adopted in this thesis to try to improve the performances of Miao et al.'s decoder [2].

4.1 Quaternary Neural Belief Propagation

Standard belief propagation can be interpreted as a deep neural network when the Tanner graph associated to the BP algorithm is unfolded [1]. In particular, each BP iteration is represented as two successive layers within a neural network, a method known as the "unfolding" or "layer-by-layer" approach, as illustrated in Fig. 4.1.

The expanded BP architecture enables the use of backpropagation-based training methods to learn the network's parameters. This structure supports end-to-end training, allowing for the simultaneous optimization of the entire architecture's parameters. Moreover, the expanded BP can be adapted into neural belief propagation. Following the training phase, messages from variable to check and from check to variable are appropriately weighted to account for small cycles in the Tanner graph.

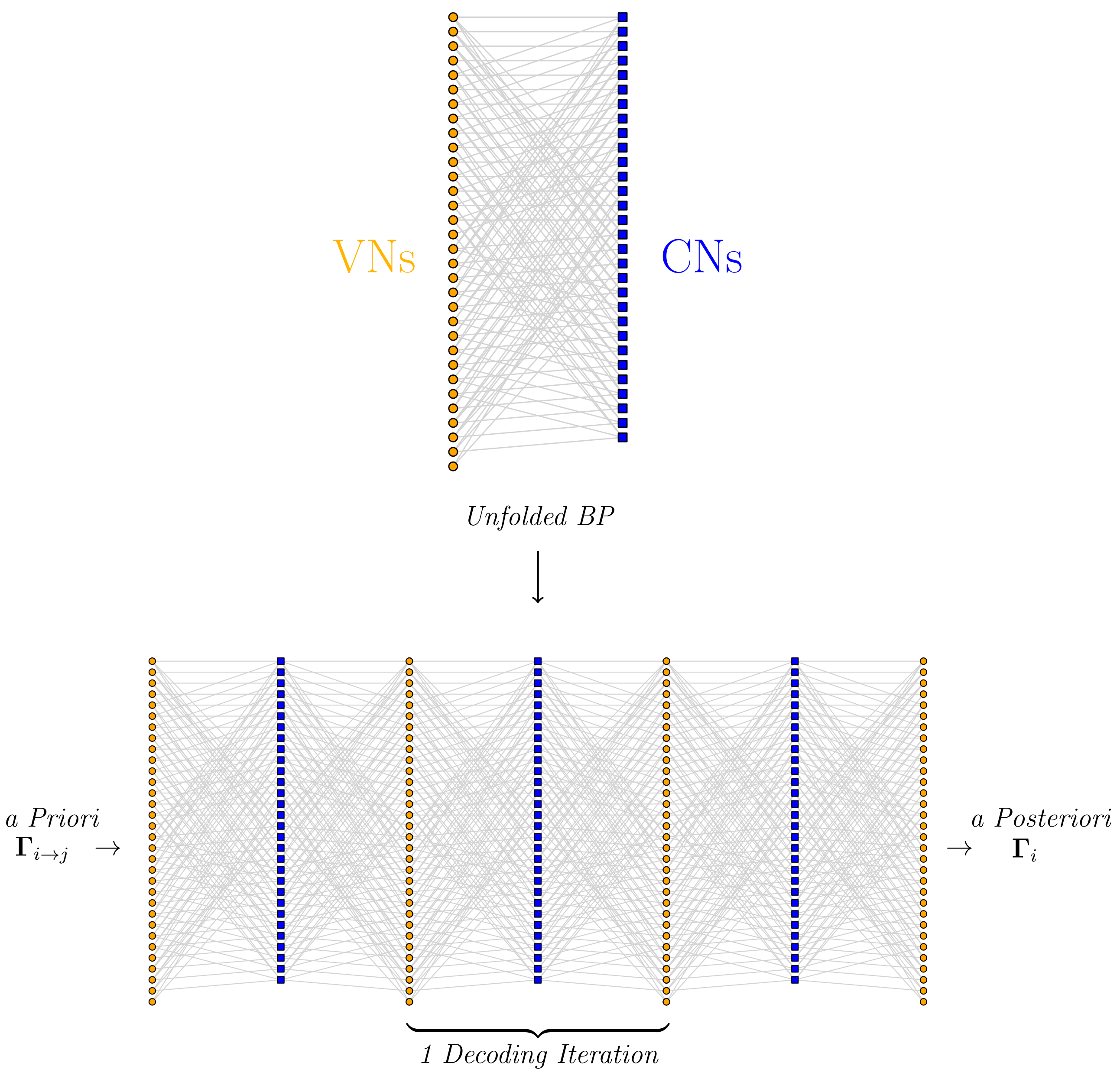


Figure 4.1: The Tanner Graph of Toric Code with $L = 4$, $n = 32$ and $m = 30$ associated to the BP is unfolded by a "layer-by-layer" approach where each iteration is represented as two successive layers within a neural network. Messages are passed between VNs and CNs in one direction until the maximum iteration number is reached.

As illustrated in Fig. 4.2, the input layer corresponds to the initial data or observations provided to the NBP model, such as the prior log-likelihood ratios. The hidden layers serve as the message-passing layers, facilitating the exchange of messages between VNs and CNs. The neural network's output is the final estimation of the variable nodes' values, specifically the estimated error value at each qubit position. Training the neural network involves minimizing a loss function that quantifies the discrepancy between the inferred and actual errors.

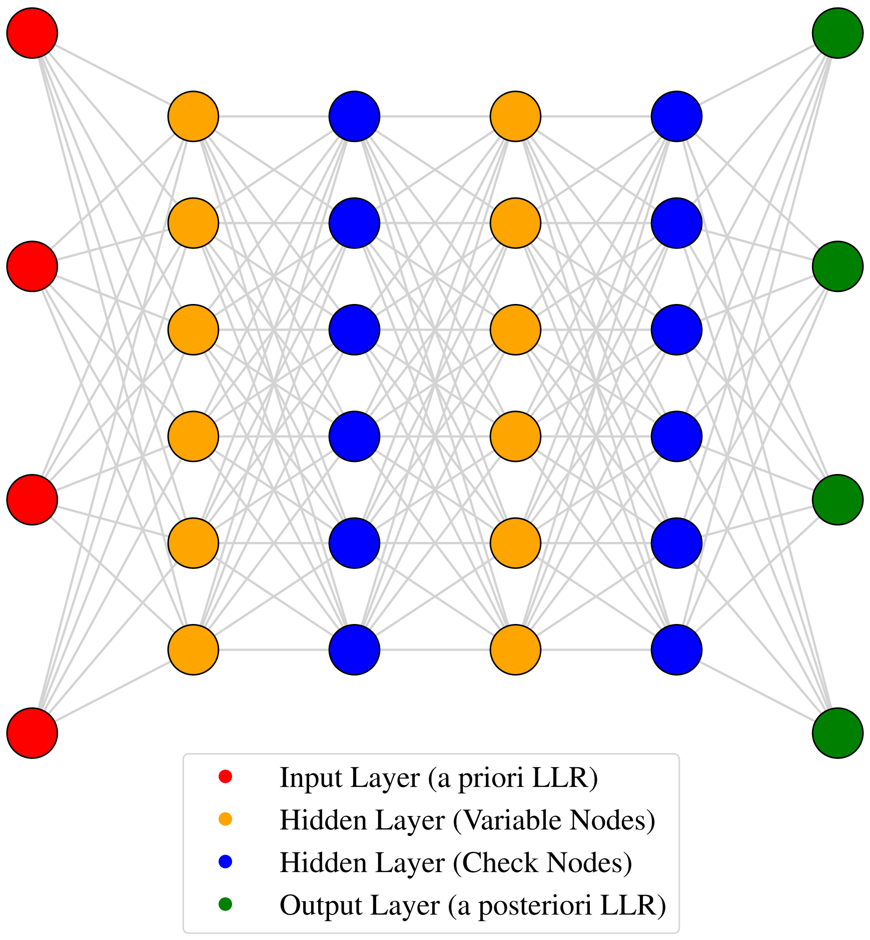
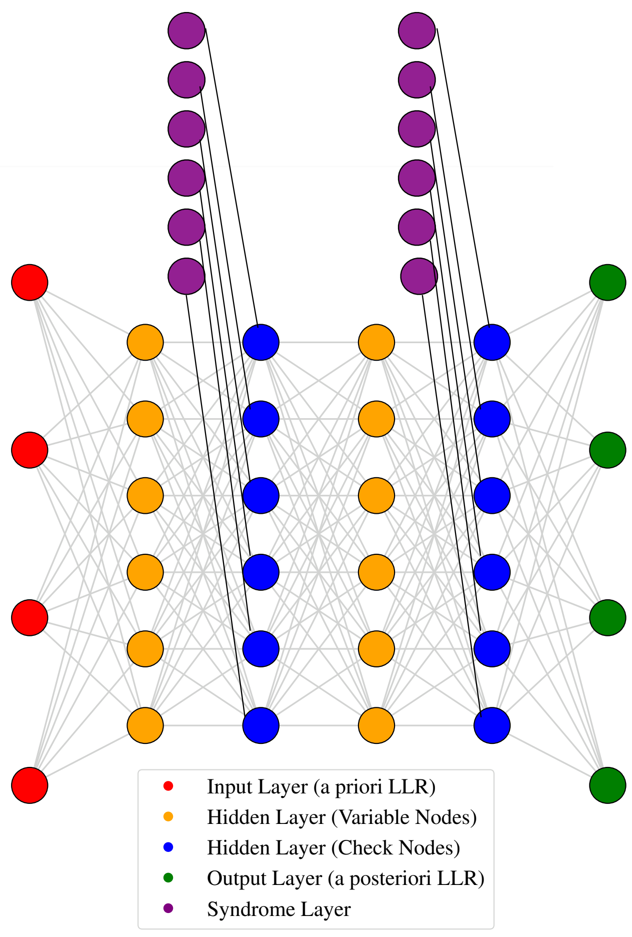

● Input Layer (a priori LLR)
● Hidden Layer (Variable Nodes)
● Hidden Layer (Check Nodes)
● Output Layer (a posteriori LLR)

Figure 4.2: An example of a feedforward neural network architecture. Neurons, edges and layers correspond to the quantities related to BP algorithm.

Fig. 4.3 shows the structure of a deep neural network architecture that incorporates the syndrome neuron [7]. The neural network receives the error syndrome derived from measurements of the quantum system as input and produces the likelihood of errors occurring at each qubit within the code as output. Training of the neural network is conducted through backpropagation and stochastic gradient descent, aiming to reduce the discrepancy between the predicted error probabilities and the actual observed values.

By incorporating neural networks into the message-passing and belief-update steps, neural belief propagation harnesses the expressive capabilities of deep learning to capture complex dependencies and nonlinear relationships, resulting in enhanced performance compared to the traditional belief propagation algorithm. The messages exchanged between variable nodes and check nodes are typically adjusted by trainable weights acquired during the neural network's training phase. This adjustment enables the neural network to learn the significance or relevance of various incoming messages in the message-passing process, allowing it to mitigate the effects of unavoidable short cycles in the Tanner graph, particularly in the context of quantum error correction.



● Input Layer (a priori LLR)
● Hidden Layer (Variable Nodes)
● Hidden Layer (Check Nodes)
● Output Layer (a posteriori LLR)
● Syndrome Layer

Figure 4.3: The feedforward neural network architecture associated to NBP with the syndrome layers.

With the addiction of trainable weights equations 3.3 and 3.4 become:

$$\Delta_{i\leftarrow j} = (-1)^{z_j} \cdot \underset{i'\in\mathcal{N}(j)\setminus\{i\}}{\boxplus} w^{(l)}_{\mathrm{v},i',j} \cdot \lambda_{i'\to j} \tag{4.1}$$

$$\Gamma^{(\zeta)}_{i\to j} = w^{(l)}_{\mathrm{v},i} \cdot \Lambda^{(\zeta)}_i + \sum_{j'\in\mathcal{M}(i)\setminus\{j\},\langle\zeta,S_{j'},i\rangle=1} w^{(l)}_{\mathrm{c},i,j'} \cdot \Delta_{i\leftarrow j'} \tag{4.2}$$

where l represents the index of the decoding iteration, and $w^{(l)}_{\mathrm{v},i',j}$, $w^{(l)}_{\mathrm{c},i,j'}$, and $w^{(l)}_{\mathrm{v},i}$ denote the trainable real-valued weights assigned to the variable node messages, check node messages, and channel log-likelihood ratio values, respectively.

As concern the loss, we consider the one proposed in [2] which takes degeneracy into account. Using $\mathbf{\Gamma}_i$ computed by 3.4 in the hard-decision pass, we can calculate

$$P\left(\langle\hat{e}_i,\eta\rangle = 1 \mid \boldsymbol{z}\right) = \left(1 + \mathrm{e}^{-\lambda_\eta(\mathbf{\Gamma}_i)}\right)^{-1} \tag{4.3}$$

for $\eta \in \mathrm{GF}(4)\setminus\{0\}$, representing the estimated probability of the i-th error commuting with $\boldsymbol{X}$, $\boldsymbol{Z}$, and $\boldsymbol{Y}$, respectively.
Consequently, the proposed loss function for each error pattern can be expressed as

$$\mathcal{L}(\mathbf{\Gamma};\mathbf{e}) = \sum_{j=1}^{2n-m} f\left(\sum_{i=1}^{n} P(\langle e_i + \hat{e}_i, S^{\perp}_{j,i}\rangle = 1 \mid \boldsymbol{z})\right) \tag{4.4}$$

where $f(x) = |sin(\pi x/2)|$, which approaches zero as x nears any even number (i.e., a "soft modulo 2" function) [7]. The loss value is accumulated over all rows $\boldsymbol{S}_j^{\perp}$ of $\boldsymbol{S}^{\perp}$. For each row j, we sum the values $P(\langle e_i + \hat{e}_i, S_{j,i}^{\perp}\rangle = 1 \mid \boldsymbol{z})$ for all elements $S_{j,i}^{\perp}$ in $\boldsymbol{S}_j^{\perp}$, representing the probability that $S_{j,i}^{\perp}$ is unsatisfied after estimating e_i as $\hat{e}_i$. In particular we can apply the following mathematical operation $P(\langle \hat{e}_i, S_{j,i}^{\perp}\rangle = 1 + \langle e_i, S_{j,i}^{\perp}\rangle \mid \boldsymbol{z})$ that leads to two possibilities:

- if $\langle e_i, S_{j,i}^{\perp}\rangle = 0 \longrightarrow P(\langle \hat{e}_i, S_{j,i}^{\perp}\rangle = 1 \mid \boldsymbol{z})$ is calculated through Eq. 4.3, with $\eta = S_{j,i}^{\perp}$.
- if $\langle e_i, S_{j,i}^{\perp}\rangle = 1 \longrightarrow P(\langle \hat{e}_i, S_{j,i}^{\perp}\rangle = 0 \mid \boldsymbol{z}) = 1 - P(\langle \hat{e}_i, S_{j,i}^{\perp}\rangle = 1 \mid \boldsymbol{z})$ is calculated.

The loss $\mathcal{L}$ is minimized if Eq. 2.19 holds.

In the case where a large number of redundant check nodes is utilized, it becomes essential to accurately normalize the messages, as the interdependence between messages intensifies during decoding. Consequently, using a relatively large ϵ_0 may not offer sufficient flexibility for message normalization. To address this, we introduce another parameter, w_r, which acts as a normalization factor for the check node messages, similar to $w_{\mathrm{c},i,j}^{(l)}$ in equation 4.2, but independent of the number of iterations l and the indices of the connected CN j and VN i :

$$\Gamma_{i\to j}^{(\zeta)} = \Lambda_i^{(\zeta)} + \sum_{j'\in\mathcal{M}(i)\setminus\{j\},\langle\zeta,S_{j'},i\rangle=1} w_r \cdot \Delta_{i\leftarrow j'} \tag{4.5}$$

This assumption can be used as in the OBP4 decoder as for the initial weight of $w_{\mathrm{c},i,j'}^{(l)}$ in the NOBP4's training. When optimization of the decoder configuration is performed, we set $w_r = 1$ if adjusting ϵ_0 alone provides satisfactory decoding performance, as this achieves the fastest convergence speed. If this approach does not suffice, the optimal pair (ϵ_0, w_r) presented in the Tab. 5.1 are used.

4.2 Residual Connections

A residual neural network, commonly known as a residual network or ResNet, is a deep learning framework where the weight layers are designed to learn residual functions in relation to the inputs of the layers. This architecture was introduced in 2015 specifically for image classification tasks and achieved victory in that year's ImageNet Large Scale Visual Recognition Challenge [25].

In terms of terminology, a "residual connection" or a "skip connection" denotes the distinctive architectural pattern represented as

$$x \mapsto f(x) + x$$

where f signifies any arbitrary neural network module. Although residual connections were used prior to the advent of ResNet, such as in Long Short-Term Memory (LSTM) networks and highway networks, the introduction of ResNet significantly popularized this concept, leading to its adoption in various unrelated neural network architectures.

In the context of QECC, they were introduced by Poulin et al. [7] with the following adhoc adaptation:

$$\Delta_{i\leftarrow j}^{(p+1)} = NBP\left(\Delta_{i\leftarrow j}^{(p)}\right) + r_p \cdot \Delta_{i\leftarrow j}^{(p)} \tag{4.6}$$

where $\Delta_{i\leftarrow j}^{(p+1)}$ denotes the input to the $(p+1)$-th layer, while $NBP\left(\Delta_{i\leftarrow j}^{(p)}\right)$ represents the current output of the NBP decoder, with r_p as the trainable parameters that scale the previous message.

Residual connections allow information to bypass one or more layers, facilitating direct data transfer from earlier layers to later ones. This approach enhances gradient flow, which improves training and mitigates the vanishing gradient problem, especially in quantum error correction contexts.

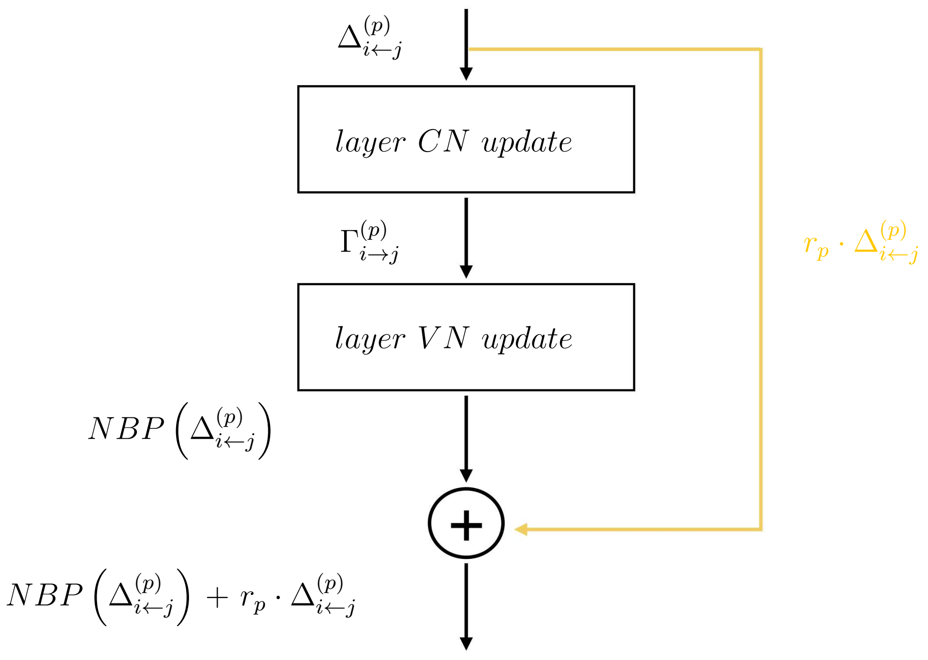

$\Delta_{i\leftarrow j}^{(p)}$
layer CN update
$\Gamma_{i\rightarrow j}^{(p)}$
layer VN update
$r_p \cdot \Delta_{i\leftarrow j}^{(p)}$
$NBP\left(\Delta_{i\leftarrow j}^{(p)}\right)$
$NBP\left(\Delta_{i\leftarrow j}^{(p)}\right) + r_p \cdot \Delta_{i\leftarrow j}^{(p)}$

Figure 4.4: Residual Block in the NBP. The message $\Delta_{i\leftarrow j}^{(p)}$ from check to variable nodes is passed through the CN's layer and then through VN's layer in order to perform a NBP decoding. The scaled previous message is added to NBP output and passed through the $(p+1)$-layer.

In neural belief propagation, residual connections can be added between layers responsible for horizontal and vertical message passing, promoting efficient information sharing. By enabling the addition of a layer's input to its output, these connections create shortcuts for gradients, allowing them to bypass layers and flow more freely.

This mechanism helps counteract the vanishing gradient problem, which can hinder learning in deep networks by causing minimal updates to parameters during backpropagation. As a result, the network can learn more effectively and converge faster.

4.3 Weight-Sharing

Weight-sharing is a fundamental concept in convolutional neural networks (CNNs) that significantly enhances their efficiency and performance. It refers to the practice of using the same set of weights, known as filters or kernels, across different spatial locations in the input data, such as images. This means that as a filter slides over the input image, it applies the same weights to various regions, allowing for easier training and enabling the network to consistently learn features across the entire image [26]. In light of these considerations, we have decided to introduce weight-sharing in the decoding of the toric code.

In the context of QECC, weight-sharing involves creating a filter that is applied to the various cells of the toric code, following the periodic boundary conditions imposed by the toric code itself. The application of this filter across the grid ensures that the different CNs share the same weights, rather than each one having its own weight (the same for VNs and LLRs).

The addition of weight-sharing transforms our feedforward neural network into a convolutional neural network (more details in Sect. 4.3.1). In particular, our decoder with weight-sharing offer several advantages typical of the CNNs [27]:

- *Fewer weights for the same number of output neurons*: weight-sharing reduces the number of parameters required, making the network more efficient.

- *Higher efficiency in encoding information*: by leveraging spatial structure, CNNs are better at encoding relevant features.
- *Reduced likelihood of overfitting*: shared weights help to prevent the model from memorizing the training data.
- *Lower memory requirements*: fewer parameters mean less memory is needed, making the network more resource-efficient.
- *Deeper networks*: with fewer parameters, it is feasible to implement much deeper networks that can learn increasingly complex features.

As illustrated in Figure 4.5, we employ the same set of weights (trainable parameters) for multiple edges. This approach reduces the number of trainable parameters, which in turn leads to easier convergence during the training process. The figure 4.5 depicts the duplication of 16 checks for plaquette operators, which correspond to $\boldsymbol{Z}$-operators, to other segments of the lattice. The same idea is also implemented for $\boldsymbol{X}$-operators. Consequently, a total of 32 weights are trained during the neural network training process.

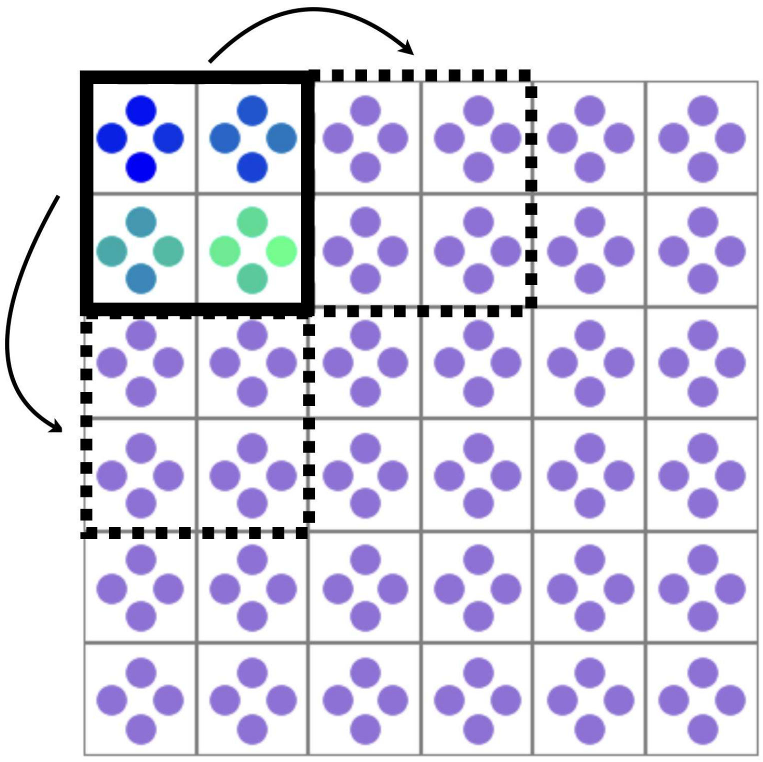


Figure 4.5: Weight-sharing in a toric code with L=6. A 2x2 kernel is moved across the lattice to share the weights.

Figures 4.6 and 4.7 illustrate the distribution of CN weights before and after training and the implementation of weight-sharing for operators $\boldsymbol{X}$ and $\boldsymbol{Z}$ (the same consideration has been also taken for the weights of VNs and LLRs). Initially, all trainable parameters are set to 1, and then these weights are replicated across different segments of the toric code. The topological structure enables weight sharing not only within a single lattice size but also across toric codes of varying sizes.

In addition to replicating weights within the lattice of the toric code, weight sharing can be applied to reuse weights across different system sizes. Specifically, we can train our model for a fixed length L and subsequently apply the weights obtained to systems of different sizes (whether larger or smaller) without losing in performance by leveraging the symmetry introduced by the application of the weight-sharing filter. In particular, the computations executed by each neuron are the same, allowing for the sharing of weights among them. In other words, the weights are tied across all edges in the Tanner graph that share the same geometry, such as all vertical or all horizontal edges. This concept is of significant importance as it allows us to train small systems and apply the results obtained to systems that may be exponentially larger, which would otherwise require substantial computational power and memory in that training NBP for large systems is very complex. Consequently, this leads to significant savings in both time and computational costs. In our study, neural network training was conducted for toric code with $L = 8$, and the weights were reused for other code sizes, specifically for toric codes with $L = 4$, $L = 6$, $L = 8$, $L = 10$, $L = 12$ and $L = 14$.

In summary, weight-sharing offers numerous advantages in the context of neural belief propagation for quantum error correcting codes. Firstly, it decreases the number of parameters in the model, resulting in faster training times and improved generalization performance. Secondly, it serves to regularize the model by promoting similarity among the weights across various edges in the Tanner graph. This can enhance the model's capability to generalize effectively to new quantum error-correcting codes. Finally, it allows us to reuse weights across several systems without the need of training all of them, leading to significant saving in computational costs and time.

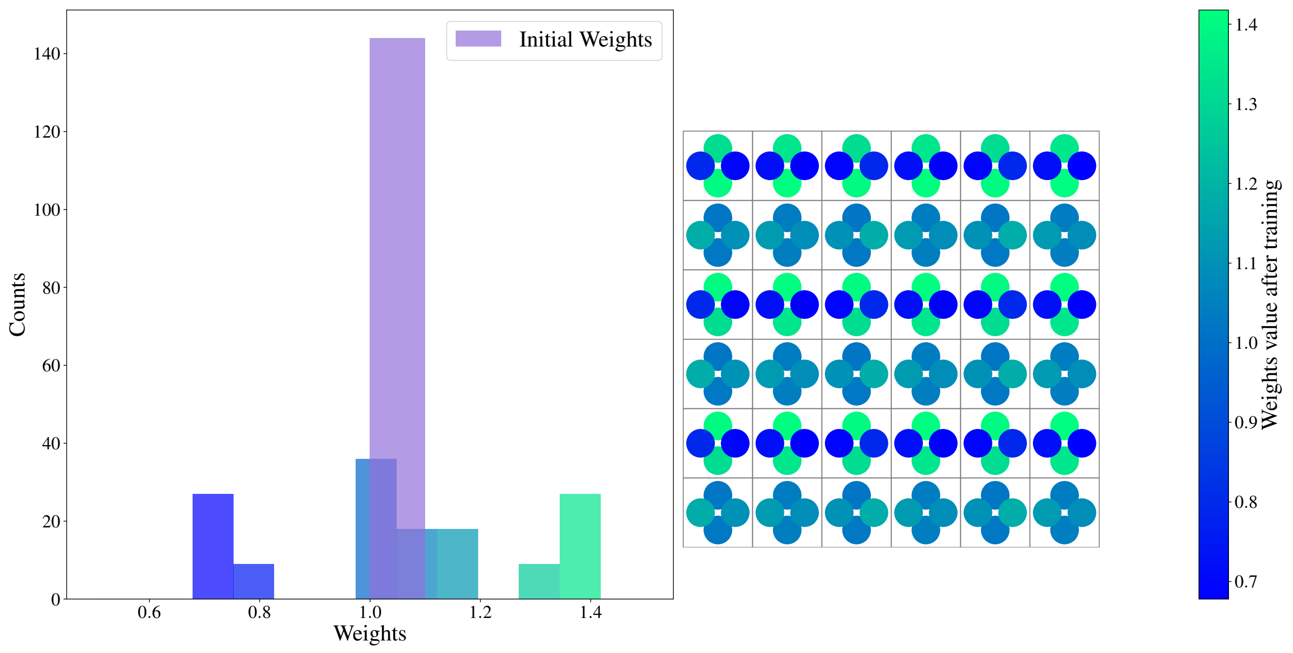


Figure 4.6: Left) Distribution of the $w_{c,i,j'}$ related to $\boldsymbol{Z}$ operators after the training in a toric code with L=6, n=72 and m=70. Right) Implementation of weight sharing for plaquette operators.

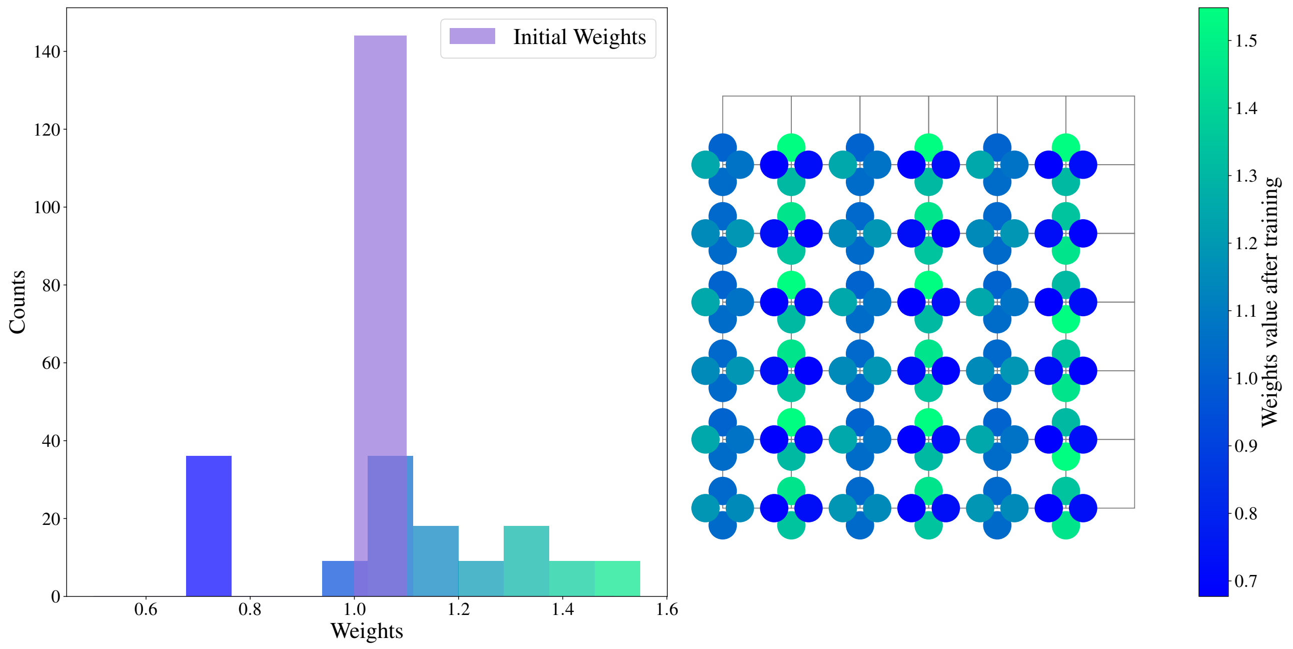


Figure 4.7: Left) Distribution of the $w_{c,i,j'}$ related to $\boldsymbol{X}$ operators after the training in a toric code with L=6, n=72 and m=70. Right) Implementation of weight sharing for vertex operators.

4.3.1 Convolutional Neural Networks for Toric Code

As mentioned above, the application of weight sharing transforms our feedforward neural network in a convolutional neural network. In this section, we present a brief parallel between the typical definitions of CNNs and their ad hoc usage in our decoder for the toric code.

The general framework of CNNs, shown in Fig. 4.8 is the following [27]:

- **Input Layer**: the network takes an input tensor with dimensions typically represented by (height,widht,channels).
- **Convolutional Layer**: this layer applies convolutional filters to the input image to create feature maps. Each filter learns to recognize specific features like edges, textures, etc. The main components include:
  - *Filters/Kernels*: small matrices (e.g., 3x3, 5x5) that slide (convolve) over the input image.
  - *Stride*: which refers to the number of steps the filter is shifted, it is represented by the movement of our kernel across the grid.
  - *Padding*: which involves adding zeros to the edges of the input tensors so that the filter fits an integer number of times.
- **Pooling Layer**: this layer, applied after the convolutional layer, reduces the spatial dimensions of the feature maps (subsampling), lowering the number of parameters and computation needed in the network. Typical pooling functions are:
  - *Average pooling* : it calculates the average value for patches of a feature map and uses it to create a downsampled (pooled) feature map.
  - *Max pooling* : it takes the max value for patches of a feature map.
- **Fully Connected Layers**: after several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Every neuron in this layer is connected to every neuron in the previous layer. The output of these layers can be fed into a softmax function to provide probabilities for classification tasks.

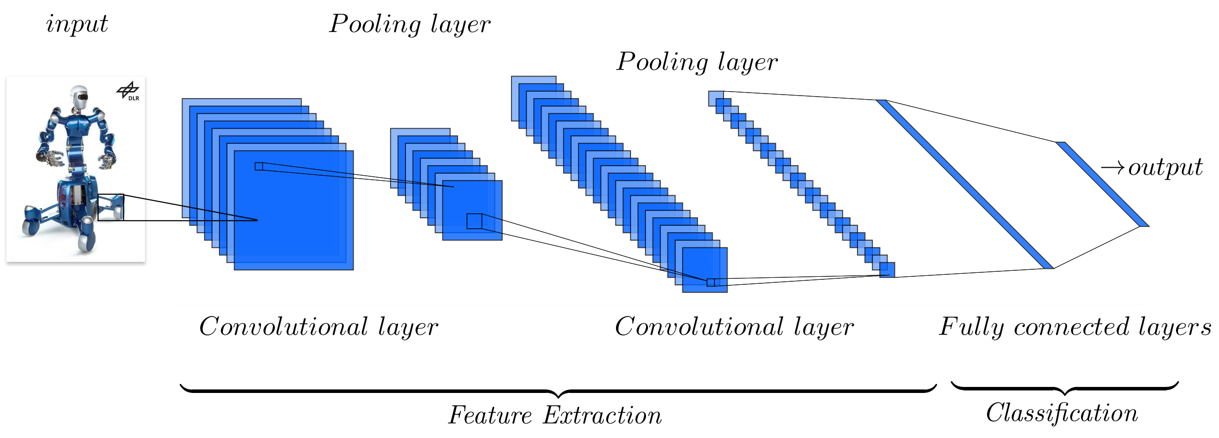


Figure 4.8: General framework of a Convolutional Neural Network.

The main characteristic of CNNs is the presence of the convolutional layer, which utilizes the convolution operation. Convolution is defined as:

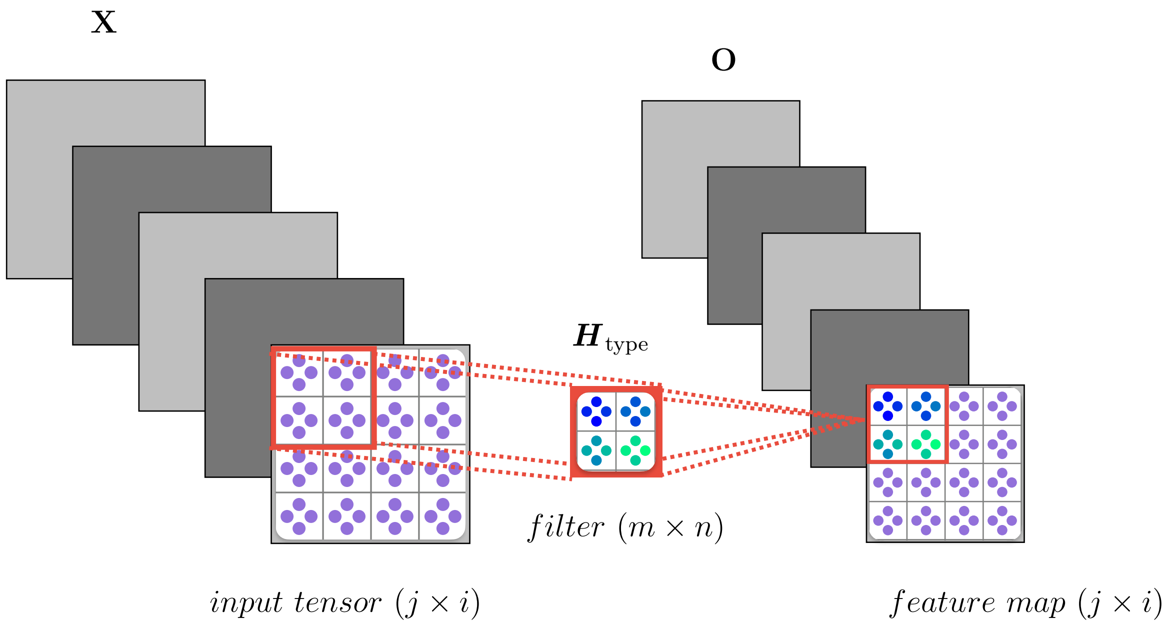
$$o_{i,j} = f\left(\sum_{q=1}^{D}(\mathbf{K}_q * \mathbf{X}_q)(i,j) + b\right) \tag{4.7}$$

where $\mathbf{K}$ is the filter containing the learnable weights $w_{i,j}$, $\mathbf{X}$ is the input tensor, $o_{i,j}$ is the output of the layer (feature map) associated with the weights $w_{i,j}$, f is the activation function, b is the bias and D the number of filters [27].

Adapting this operation to our toric code, we first define the filter $\boldsymbol{H}_{\text{type}}$, represented by a matrix of dimensions $m \times n$ containing values from 0 to m. These values represent the indices that define which check nodes, as well as log-likelihood ratios and variable nodes, share the same weight. This filter is then applied to the input weight tensor of our toric code, which allows us to obtain the activation map.

Subsequently, the activation function f is applied. In particular, we consider f as the identity function when the index in the pivot of $\boldsymbol{H}_{\text{type}}$ is different from 0. In this case, the corresponding input weight for each index is retained, and the filter is applied to all elements of the tensor located at the corresponding index position. Conversely, if the pivot´s value is 0, the weight is set to 0. This approach ensures that the output tensor presents identical weights for all indices that are equal in $\boldsymbol{H}_{\text{type}}$, thereby enabling weight sharing. This output tensor serves as our feature map o, which is fed as input to the next layer. It is important to note that we consider a bias b to be zero. This represents the adaptation of the classic operation between two layers of a standard CNN to our weight-sharing decoder for the toric code.

In order to obtain the convolutional layer we also need the stride and the padding. The first is set to 2 (in fact the 2x2 kernel is moved across the lattice kernel as shown in Fig. 4.5), while the padding is simply replaced by the periodic boundary conditions of the toric code. This condition imposes a symmetric filter in both width and height, resulting in the filter being repeated an integer number of times.



$\mathbf{X}$
$\mathbf{O}$
$\boldsymbol{H}_{\text{type}}$
$filter\ (m \times n)$
$input\ tensor\ (j \times i)$
$feature\ map\ (j \times i)$

Figure 4.9: Schematic representation of the convolutional layer used in our decoder.

Regarding the pooling operation, we considered an average pooling between the weights corresponding to the same value in the pivots of $\boldsymbol{H}_{\text{type}}$. On the other hands, the fully connected layer is represented by the NBP architecture discussed in Sect. 4.1. The schema of our ad hoc CNN is shown in Fig. 4.10.

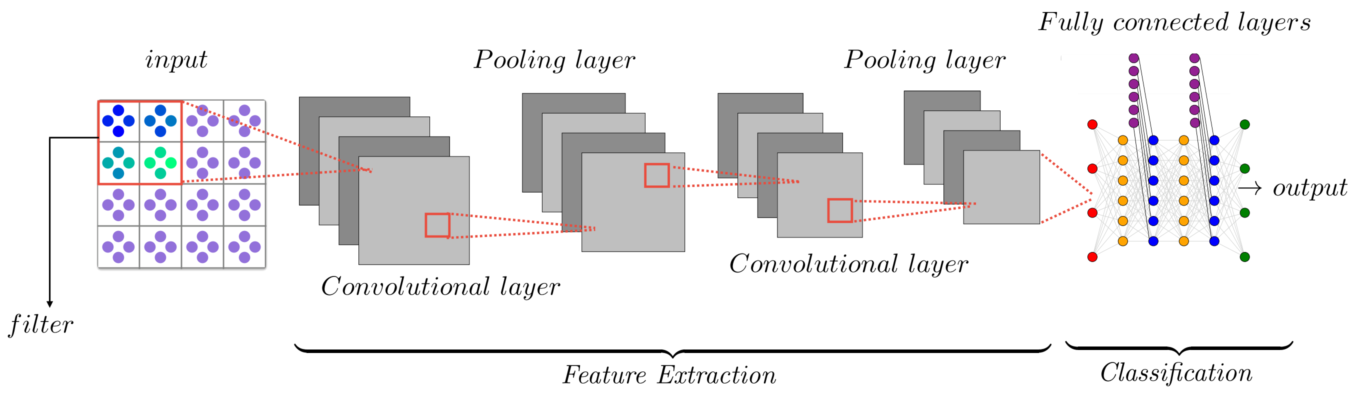


Figure 4.10: Schematic representation of the CNN used in this thesis.

Chapter 5

Numerical Results

In this chapter, we present the outcomes derived from the implementation and evaluation of the NBP4 decoder with the deep learning techniques described in the previous chapters. The results obtained have been compared with existing findings in the literature computed by Miao et al. [2] to assess the effectiveness and improvements offered by deep learning to the NBP decoder in detecting and correcting quantum errors.

The findings illustrate how the integration of deep learning methodologies enhances performance. In cases where improvements are not observed, the results still demonstrate comparability with those documented in the literature, achieved with a reduction in computational complexity and memory resource requirements.

This comparative analysis not only highlights the potential of the NBP4 decoder in the realm of quantum error correction but also emphasizes the efficiency gains that can be realized through the use of deep learning techniques. The results suggest that the NBP4 decoder is a valuable option to effectively address quantum errors while maintaining manageable resource consumption.

5.1 Training

In this section, we present the training procedure for the NBP decoder. The training phase was conducted using the PyTorch library in Python 3.11.7 on a CUDA-enabled device, while the testing part was implemented in C++ 13.2.0. To accelerate the computations, an Ubuntu virtual machine equipped with 2 NVIDIA RTX 6000 Ada EDU GPUs and 48GB of RAM was utilized.

At the beginning of the training process, the initial values for $w^{(l)}_{\mathrm{v},i',j}$, $w^{(l)}_{\mathrm{v},i}$ are both set to 1 and ϵ_0 to 0.1. If an overcomplete check matrix is used, we use the initial weight w_r for redundant checks in conjunction with ϵ_0. In this scenario, the initial value of $w^{(l)}_{\mathrm{c},i,j}$ will be set to w_r. In contrast, when redundant checks are not employed, the initial value of $w^{(l)}_{\mathrm{c},i,j'}$ is set to 1. The optimal values used for ϵ_0 and w_r are taken from [2] and shown in Tab. 5.1.

L	n	k	d	w_c	m_{oc}	ϵ_0	w_r
4	32	2	4	4	96	0.45	-
6	72	2	6	4	216	0.35	0.1
8	128	2	8	4	384	0.37	0.1
10	200	2	10	4	600	0.45	0.15

Table 5.1: Parameters used for the training.

Following this, the neural network decoder training is executed using the proposed loss function 4.4 through a standard stochastic gradient descent. During the training phase, mini-batches containing 120 samples are utilized. Each mini-batch consists of six sets of 20 randomly generated error patterns derived from depolarizing channels, each associated with six distinct physical error probabilities ϵ. For training the NBP4 decoder, the set of ϵ values is $\{0.02, 0.03, ..., 0.07\}$, while for the ONBP4 decoder, we employ the set $\{0.06, 0.07, ..., 0.11\}$. In addition, to ensure sufficiently accurate results, 300 logical errors are collected for each data point. Throughout this thesis, we use a flooding schedule where all VNs/CNs are updated in parallel in each decoding iteration.

The loss function 4.4 is computed after each decoding iteration l, which yields a loss value $\mathcal{L}^{(l)}$. The loss per error pattern is determined as the minimum value across all 25 iterations, represented as $\min\{\mathcal{L}^{(l)} : l \in \{1, 2, \dots, L\}\}$. In cases where multiple iterations produce the same minimum value, we select the earliest iteration. The plots showing the behaviors of $\mathcal{L}$ for the different decoders are reported in Appendix 7. To prevent exploding gradients during the initial decoding iterations, where the magnitude of the messages is small, the gradients are clipped to 10^{-3}. Furthermore, we gradually reduce the learning rate from 1 to 0.1 using a linear scheduler. For the NBP4 decoder without overcomplete check matrices, we conduct training over 2000 batches. In contrast, when using ONBP4 decoder, we limit the training to 200 batches. We maintain the same batch counts for the implementations of the decoders with residual connections (RNBP4 and RONBP4) and weight-sharing (WSNBP4 and WSONBP4).

5.2 Quaternary Neural Belief Propagation

In this section, we introduce the results for the NBP4 and ONBP4 decoders (considering overcomplete check matrix with weight $\mathbf{w} = 6$), without the application of additional deep learning techniques at this stage. The results are taken from Miao et al. paper [2], where they have been compared with those obtained from the neural BP2 decoder and the Minimum Weight Perfect Matching decoder. The results are illustrated in Fig. 5.1.

The findings indicate that, following training, both the binary and quaternary neural belief propagation methods (without the use of overcomplete check matrices) show significant improvements compared to standard belief propagation. This demonstrates the effectiveness of the training process. In both cases, the performance is lower bounded by the MWPM decoder, which is regarded as near-optimal for the XZ channel.

Furthermore, the introduction of overcomplete check matrices enables us to surpass this performance bound for all values of L except for $L = 10$, requiring only 25 iterations. However, for larger block lengths, it becomes necessary to increase the number of iterations to achieve performance levels comparable to those of the MWPM decoder.

Finally, from the results shown in Fig. 5.1, it is intersting to observe that the performance of belief propagation deteriorates as L increases in both binary and quaternary cases. This can be attributed to several factors. In BP, information is exchanged between neighboring qubits through message passing. As the size of the system increases, the distance between qubits also increases, resulting in longer propagation paths for messages and a higher probability of errors occurring. Longer message paths make it more challenging for BP to accurately estimate error probabilities, leading to reduced performance.

In larger systems, the likelihood of errors occurring and spreading throughout the code is higher. The presence of multiple errors in different locations can result in error accumulation, complicating BP's ability to differentiate between various error configurations. Accumulated errors can introduce additional complexities and increase the difficulty of accurately inferring and correcting errors.

Another factor affecting BP performance is the computational complexity, which grows with the size of the system. As the number of qubits and syndrome measurements increases, the computational resources required for BP also rise. This increased complexity makes it harder to execute BP efficiently, especially for large-scale quantum systems.

So in contrast to standard BP, neural belief propagation improves performance as the system size L increases in the context of the toric code and can overcome MWPM bound with the employment of overcomplete check matrices, showing that NBP itself has certain advantages that enable it to handle larger toric code systems more effectively.

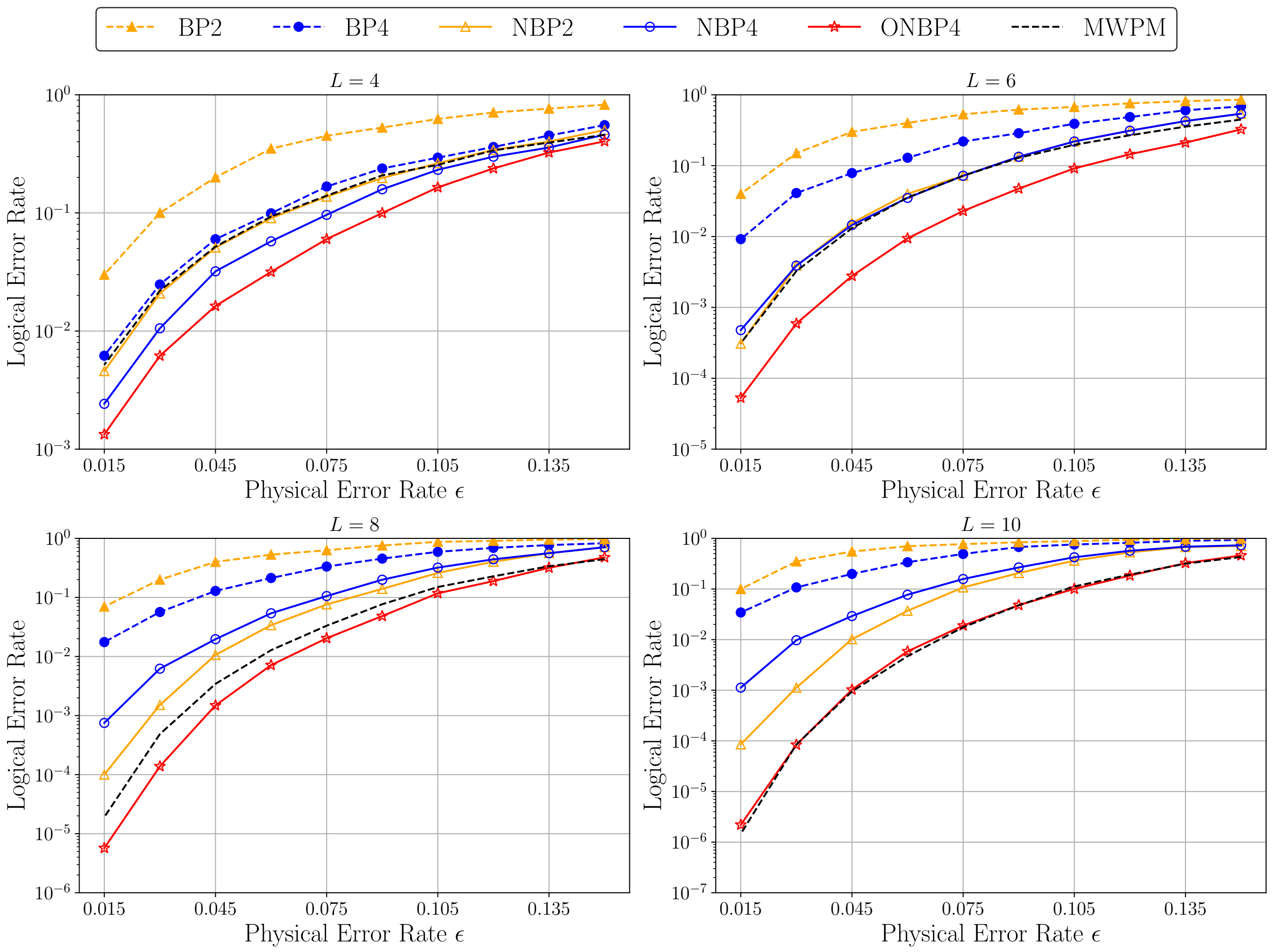


Figure 5.1: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ utilizing the trained NBP decoders.

5.3 Residual Connections

The first deep learning technique applied to improve the performance is residual connections. The results of residual quaternary neural belief propagation and RNBP4 using an overcomplete check matrix decoders are shown in Fig. 5.2

From the results, we observe that the application of residual connections does not lead to a significant improvement in the decoder's performance, unlike what is observed with the NBP2 decoder presented in [7] and [28]. The lack of benefit from residual connections may be attributed to the architecture of the neural network. In fact, for ONBP4, the performance is already quite remarkable, indicating that the network is well-optimized for the task at hand. Consequently, the training process remains effective even in the absence of residual connections.

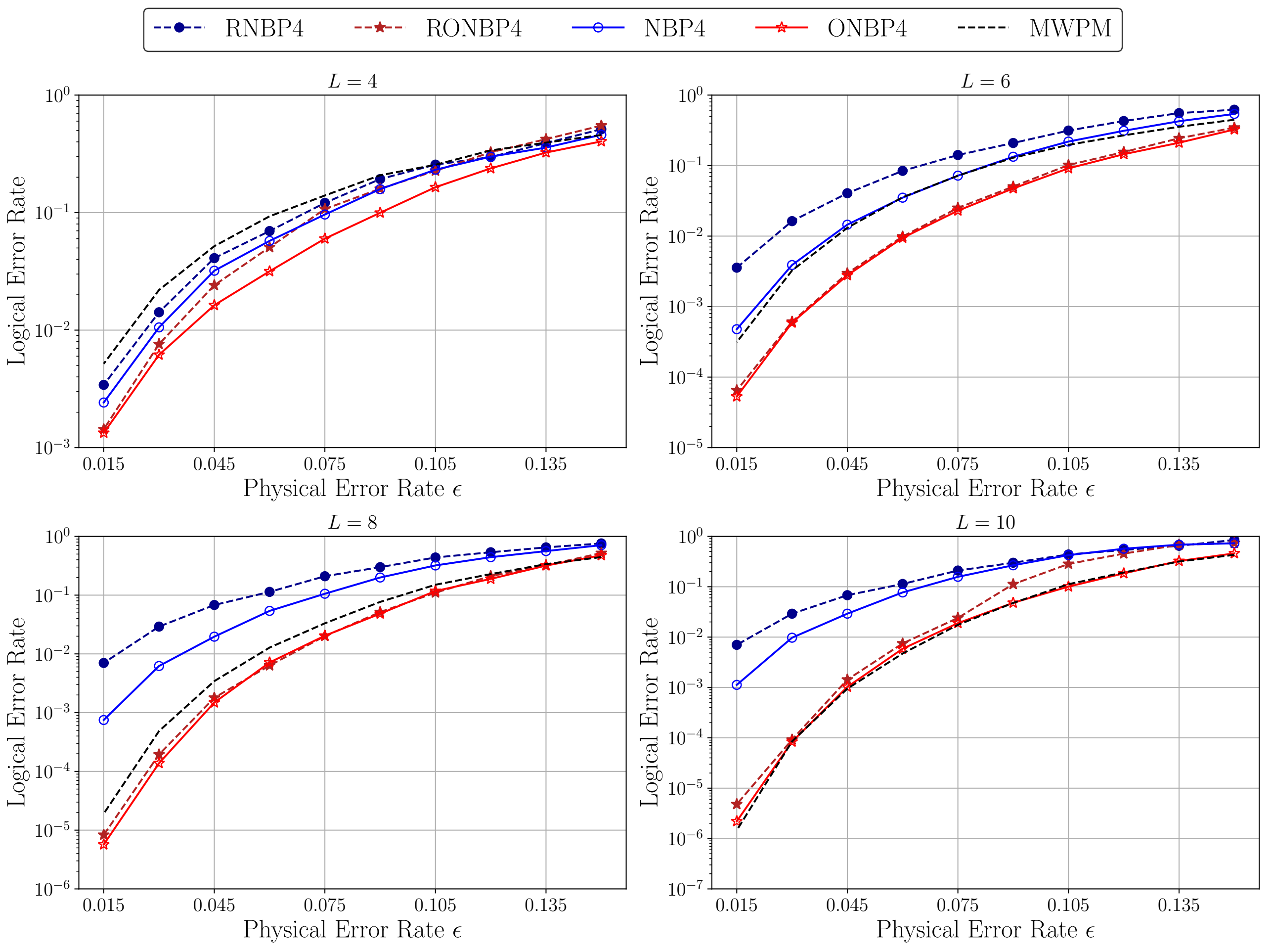


Figure 5.2: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ utilizing the residual connections.

5.4 Weight-Sharing

The second advanced deep learning technique used to enhance the performance of the decoder is weight-sharing that leads us to weight-sharing quaternary neural belief propagation (WSNBP4) decoder and WSNBP4 utilizing overcomplete check matrix (WSONBP4) decoder.

To obtain the best filter size and error interval for the architecture, preliminary training and testing sessions for WSONBP4[1] were conducted using kernels of sizes 1x1, 2x2, and 4x4, along with intervals of ϵ equal to $\{0.02, 0.03, \ldots, 0.07\}$, $\{0.06, 0.07, \ldots, 0.11\}$, and $\{0.09, 0.10, \ldots, 0.14\}$. In particular, firstly we have seeked for the best filter size keeping ϵ fixed to $\{0.06, 0.07, \ldots, 0.11\}$ in that it was the best interval provided in [2]. Once the optimal plaquette was found, we searched for the best ϵ interval. Investigation of these parameters revealed that the optimal filter size is 2x2, while the best interval for ϵ is $\{0.09, 0.10, \ldots, 0.14\}$. The results of this investigation are presented in Fig. 7.3 in Appendix 7.

The results concerning weight-sharing decoder with optimize parameters, obtained from the previous investigation, are presented in Fig. 5.3.

The results demonstrate that implementing weight-sharing with the overcomplete check matrix leads to slight performance improvements compared to ONBP4, for all values of L, with the exception of $L = 10$, where the performance still falls short of the MWPM lower bound. In particular, the case of $L = 6$ shows the greatest benefit from the weight-sharing approach,

[1]We have chosen to focus our investigation on the overcomplete check matrix case, as it has shown to represent the most effective decoder in previous results, even without the implementation of weight-sharing.

indicating that this method is especially advantageous in this scenario.

On the other hand, when the overcomplete check matrix is not utilized, the performance for $L = 6$ does not exhibit the same level of improvement. While some enhancements are observed for other values of L, they are less pronounced, and crucially, the MWPM threshold is surpassed only for the $L = 4$ case, even with the weight-sharing implementation.

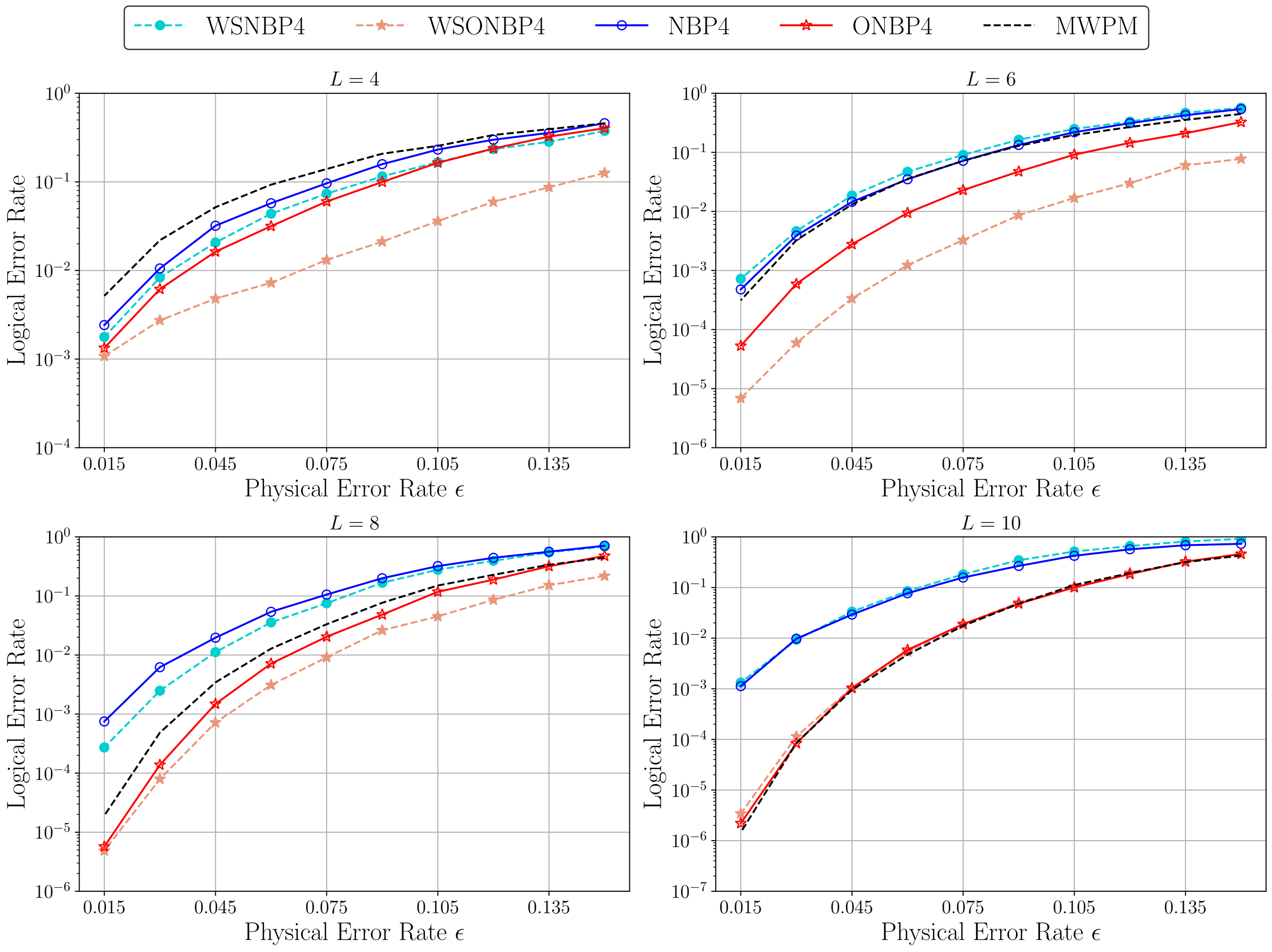


Figure 5.3: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ utilizing weight-sharing.

Furthermore, to evaluate the effectiveness of weight-sharing and overcomplete check matrices more comprehensively, we increased the weight of the latter from 6 to 8. Specifically, we followed the procedure outlined in Sect. 2.9.1, considering combinations of three consecutive elementary cells both vertically and horizontally. This approach allowed us to construct matrices that incorporate all previously used weight-4 and weight-6 stabilizers, along with n weight-8 stabilizers. The updated values for training and validation are presented in Tab. 5.2, while the results are illustrated in Fig. 5.4.

L	n	k	d	w_c	m_{oc}	ϵ_0	w_r
4	32	2	4	4	128	0.45	-
6	72	2	6	4	288	0.35	0.1
8	128	2	8	4	512	0.37	0.1
10	200	2	10	4	800	0.45	0.15

Table 5.2: Parameters used for the training with weight-8 overcomplete check matrix.

From the graph in Figure 5.4, we observe that the introduction of an overcomplete check matrix with weight 8 results in a substantial improvement for $L = 4$ and $L = 6$. In particular, the decoders outperform the MWPM decoder by two orders of magnitude and the results from Miao et al. [2] by one order of magnitude. For $L = 8$, there is an improvement, but it is less pronounced compared to the previous cases (one order of magnitude with respect to MWPM). Meanwhile, for $L = 10$, the performance reaches that of the MWPM decoder.

This analysis demonstrates that weight-sharing and the use of an overcomplete check matrix with $\mathbf{w} = 8$ are effective in enhancing decoder performance compared to other methods. Specifically, these techniques provide the most significant benefits for systems with smaller sizes, achieving improvements up to two orders of magnitude. However, as the size of the system increases, the performance gains become less pronounced.

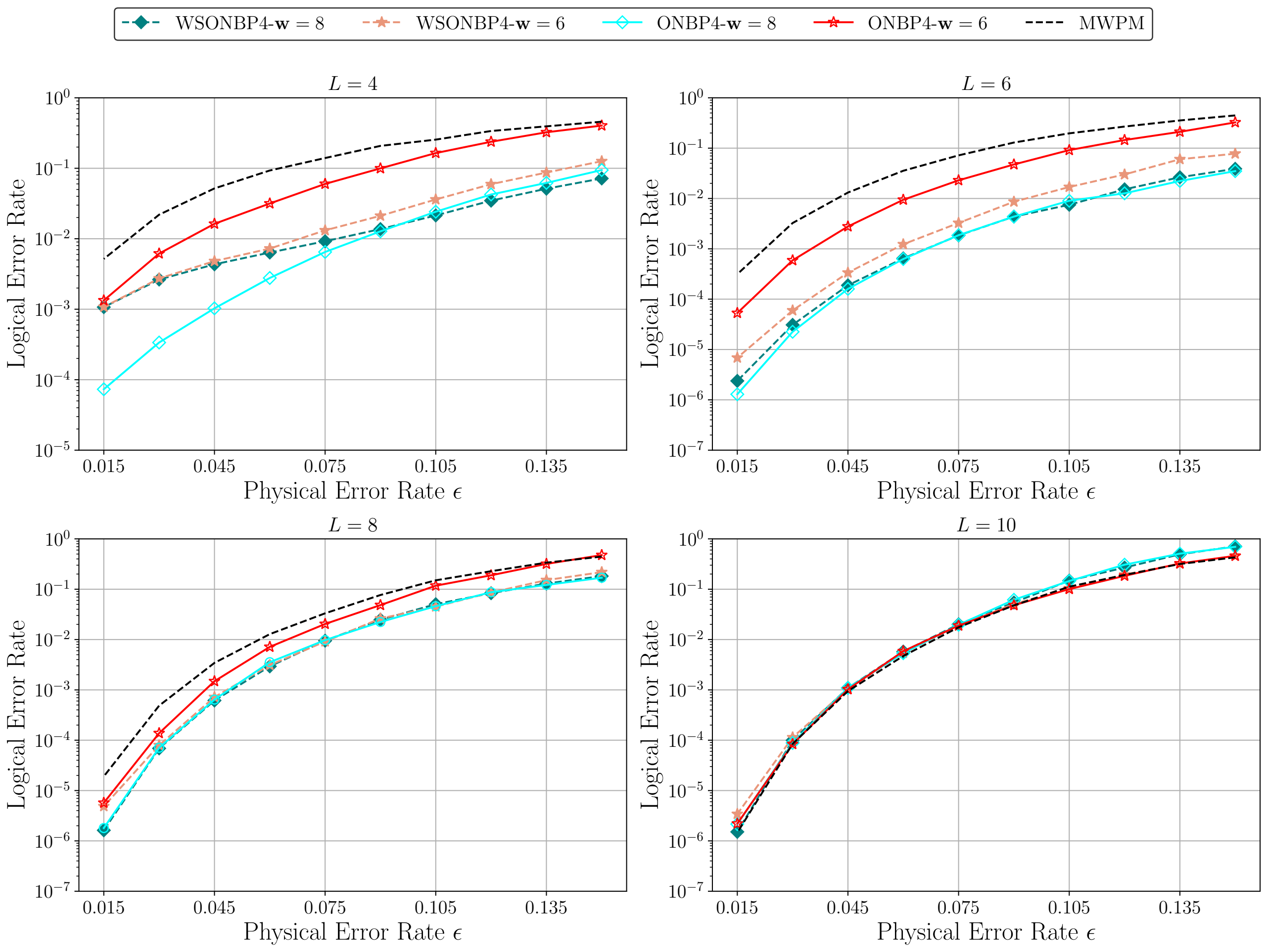


Figure 5.4: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ utilizing weight-sharing and overcomplete check matrix with weight $\mathbf{w}$=6 and $\mathbf{w}$=8.

5.4.1 Weights Reuse Strategy

The implementation of weight-sharing allows us to consider the reuse of weights obtained from training of one code to the others. We can train our model for a fixed length L and subsequently apply the weights obtained to systems of different sizes without losing in performance by leveraging the symmetry introduced by the application of the weight-sharing filter as explained in Sect. 4.3.

In this thesis, we have chosen to reuse the weights obtained from the WSNBP4 and WSONBP4 decoders trained on the toric code with $L = 8$ (the abbreviation used to denote weight-reuse is RW8). The results without the use of overcomplete check matrices are presented in Fig. 5.5, while those with the use of overcomplete check matrices are shown in Fig. 5.6.

Regarding the NBP4 decoder, from Fig. 5.5 it is evident that weight reuse matches the performance in the system as the size increases, while for smaller systems, it presents an improvement in $L = 4$ and a deterioration in $L = 6$. This demonstrates that reusing weights for larger systems can be advantageous; however, the performance does not reach that of MWPM (both with and without weight reuse). Consequently, it is necessary to focus on the case of the decoder with the use of overcomplete check matrices, ONBP4.

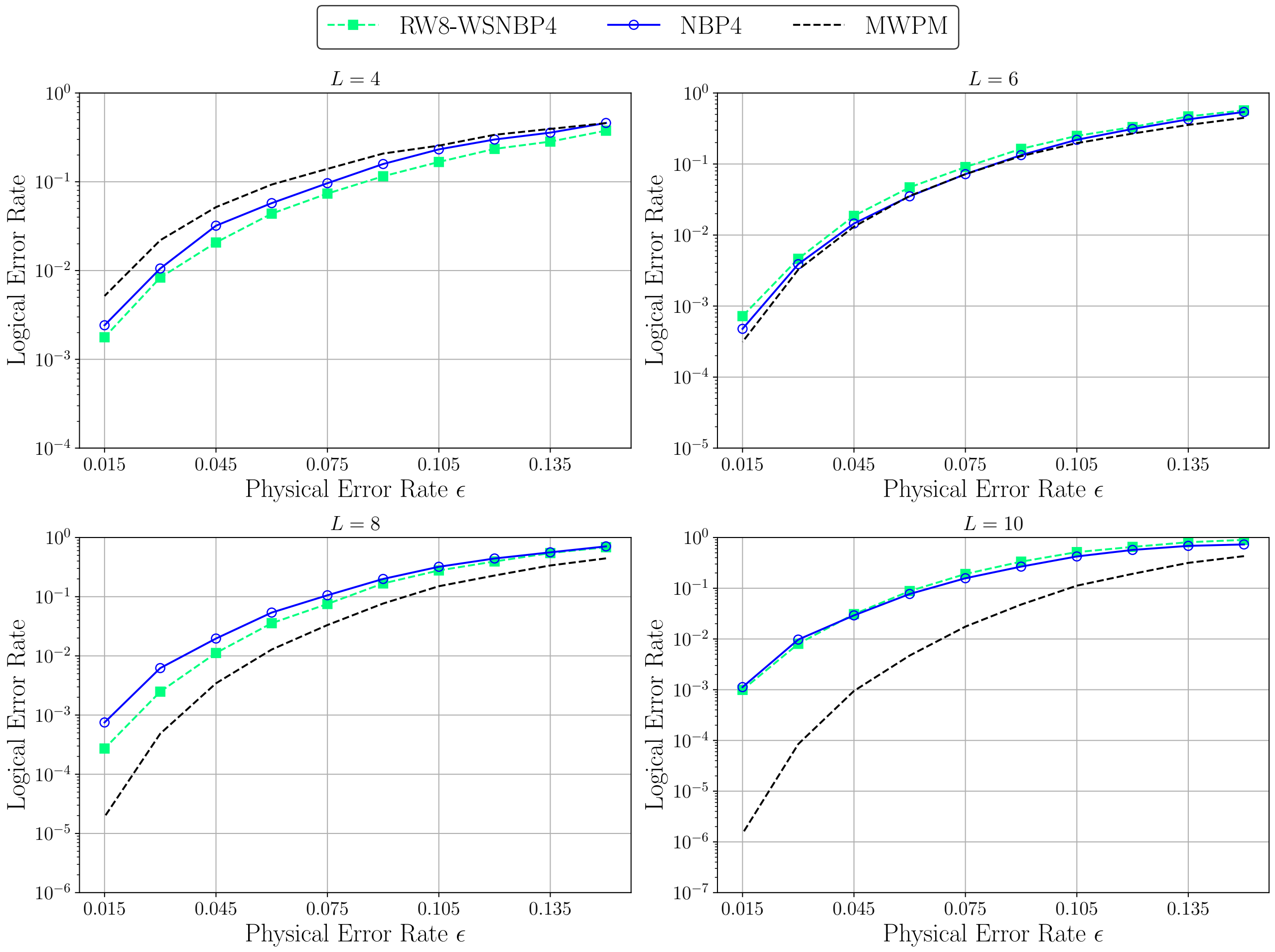


Figure 5.5: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ reusing the weights of L=8 from WSNBP4 decoder.

In the case of the ONBP4 decoder, the results presented in Fig. 5.6 show that reusing the weights obtained for a toric code of size $L = 8$ allows surpassing the error threshold given by the MWPM decoder, except for the case of $L = 10$. In particular, using overcomplete check matrices with larger weights improves the performance of weight reuse, obtaining optimal results especially for $L = 6$ where the gain is two orders of magnitude with respect to MWPM. However, as the system size increases, the performance decreases slightly, preventing it from surpassing that of the ONBP4 decoder and the MWPM decoder. As a result, to achieve the performance of the MWPM decoder as the system size increases, it is necessary to increase the number of decoding iterations (in our case from 25 to 50, as shown in Fig. 7.4 in Appendix 7).

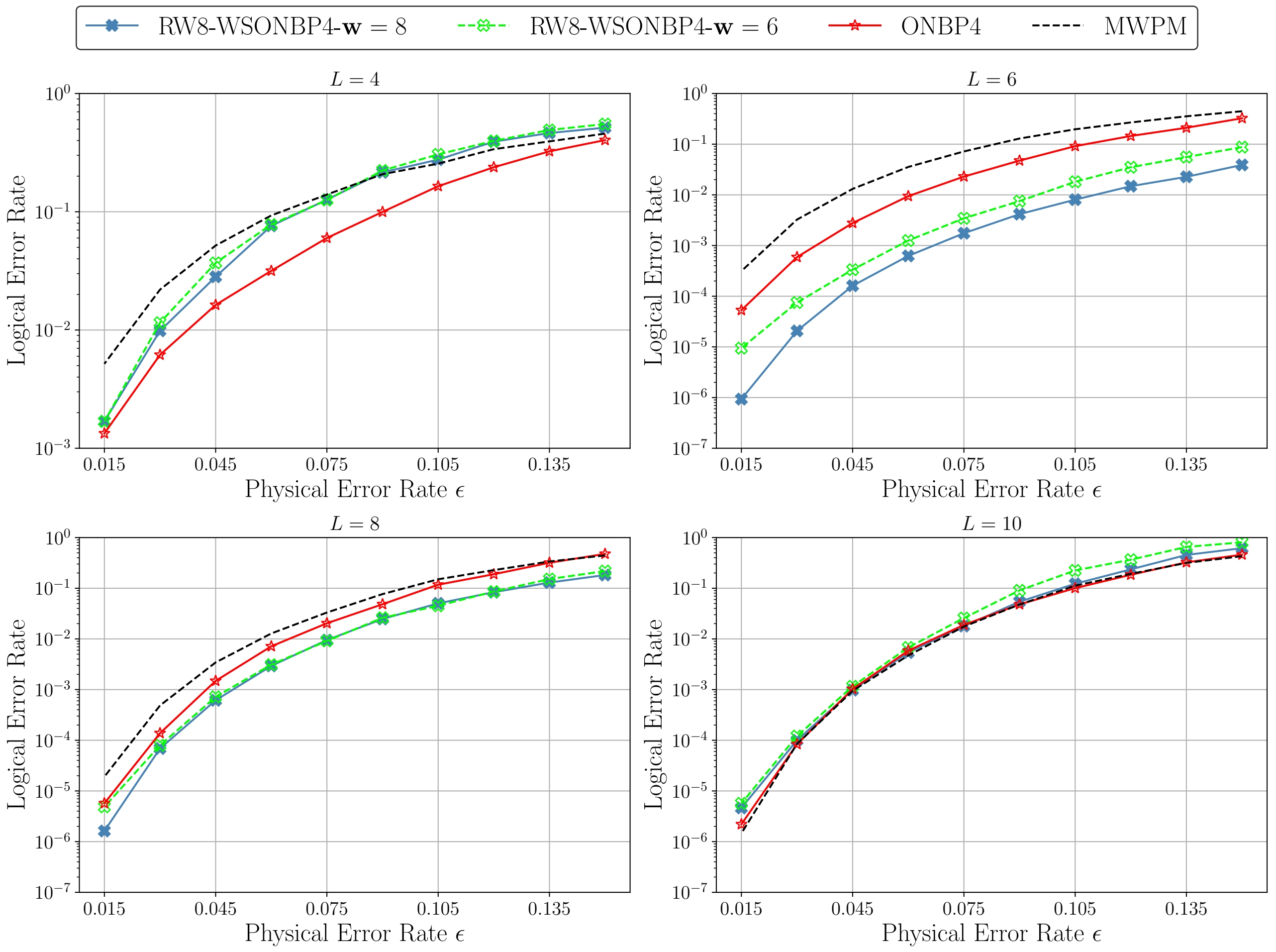


Figure 5.6: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ reusing the weights of L=8 from WSONBP4 decoder.

Furthermore, when considering even larger systems with $L = 12$ and $L = 14$, reusing the weights from $L = 8$ proves to be even less effective, as shown in Fig. 5.7. In this case, even increasing the number of iterations does not allow us to achieve the performance levels of the MWPM (as demonstrated by the graph 7.5 in Appendix 7).

This analysis leads to the conclusion that the reuse of weights does not result in performance gains as L increase. So it is essential to train the system individually as the system size increases.

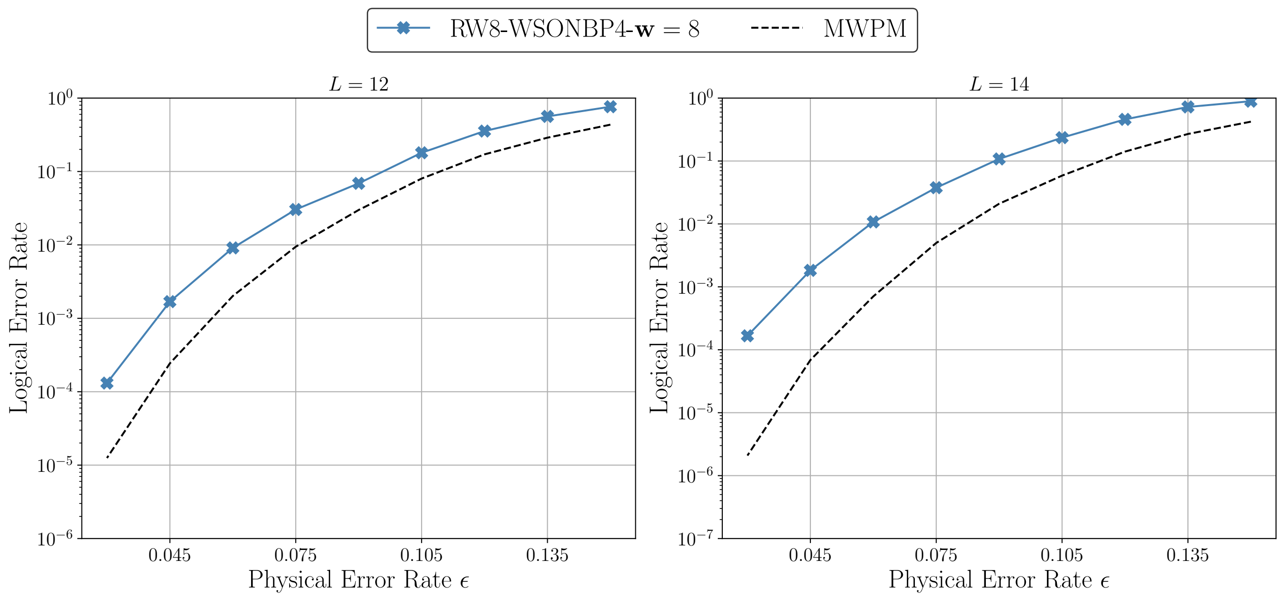


Figure 5.7: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{12, 14\}$ reusing the weights of L=8 from WSONBP4 decoder.

Chapter 6

Conclusions

6.1 Conclusions

In this thesis, we have explored the application of machine learning and deep learning techniques in quantum error correction, focusing specifically on the toric code. We investigated the implementation of the NBP4 decoder proposed by Miao et al. [2] and introduced techniques such as residual connections, weight-sharing (which introduces a convolutional structure in the neural network), higher weight overcomplete check matrices, and weight reuse to potentially improve the decoder's performance.

Our results indicate that the use of residual connections does not lead to an improvement in the performance of the NBP4 decoder presented in [2]. This contrasts with the binary case, where the introduction of residual connections yields benefits, as reported in [7] and [28].

Regarding the application of weight-sharing and higher weight overcomplete check matrices, our results demonstrate an improvement of two orders of magnitude for $L = 4, 6$ compared to the results obtained with MWPM and an improvement of one order of magnitude compared to those presented by Miao et al. [2]. For $L = 8$, an improvement is observed, though it is less significant, while for $L = 10$, we achieve a performance comparable to that reported in the literature [2].

Weight-sharing also allowed us to introduce the technique of weight reuse. Training for large toric codes can be computationally expensive and time-consuming, so reusing weights from a moderately sized system for much larger systems could present a viable alternative. Our results show that, while weight reuse largely improves performance with respect to standard BP, it does not appraoch the performance of miminum-weight perfect matching. More concretely, the performance under weight reuse appears to saturate, so that the performance does not improve even if the toric code size L increases. However, we have also observed that increasing the number of decoding iterations can improve performance in this case, see Appendix 7.

In conclusion, this thesis has further investigated the potential of neural belief propagation applied to the toric code in quantum error correction. We have implemented advanced artificial intelligence techniques to improve the performance of the NBP4 decoder proposed in [2]. Our results demonstrate that the integration of weight-sharing and higher-weight overcomplete check matrices leads to a significant improvement in results for small systems compared to those in the literature. However, as the size of the toric code grows, the improvement diminishes.

These findings contribute to ongoing research on quantum error correction and provide insights into the scalability and effectiveness of machine learning techniques in this field. In order to enhance quantum error correction algorithms, future work could focus on developing other strategies to maintain performance improvements for larger system sizes, on exploring the use of other optimizers or schedulers for the NBP architecture, and on the implementation of a more realistic circuit level noise model as channel. Moreover, the results obtained in this work represent a solid foundation for the publication of a future scientific article, which could further investigate and expand upon the findings presented here.

Chapter 7

Appendix

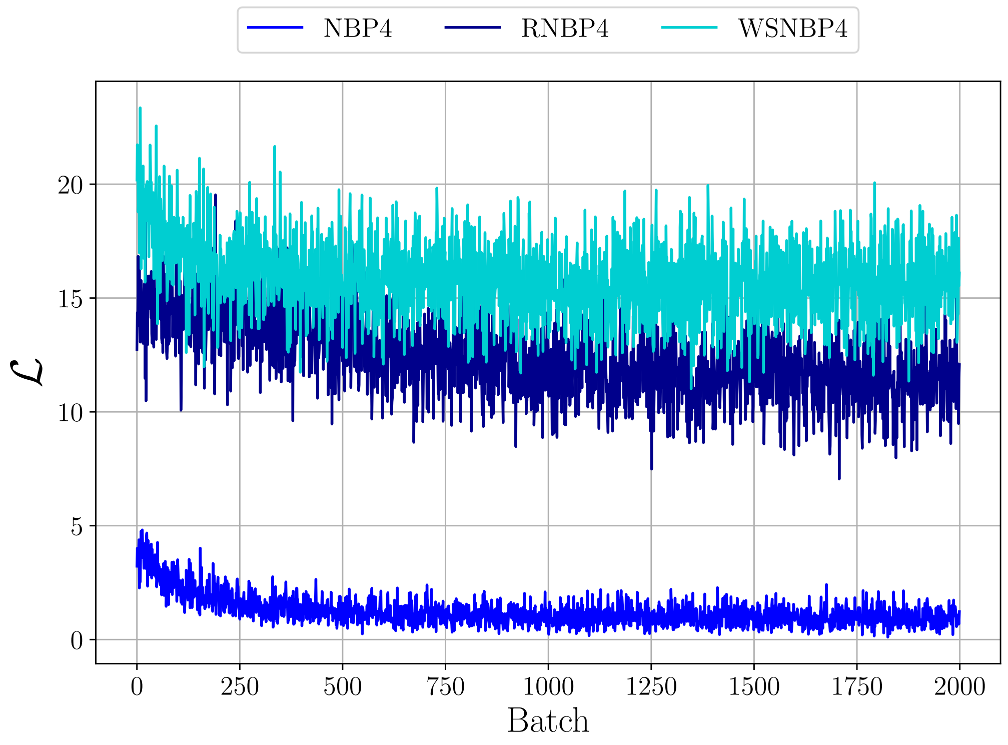


Figure 7.1: The training loss curves for the NBP4 decoder and its deep learning variations are analyzed for the toric code with $L = 10$. The WSNBP4 decoder converges to the highest loss value, indicating that its training is the least effective among the decoders considered. In contrast, the NBP4 decoder appears to be the most efficient. The same considerations are applied also for the other values of L.

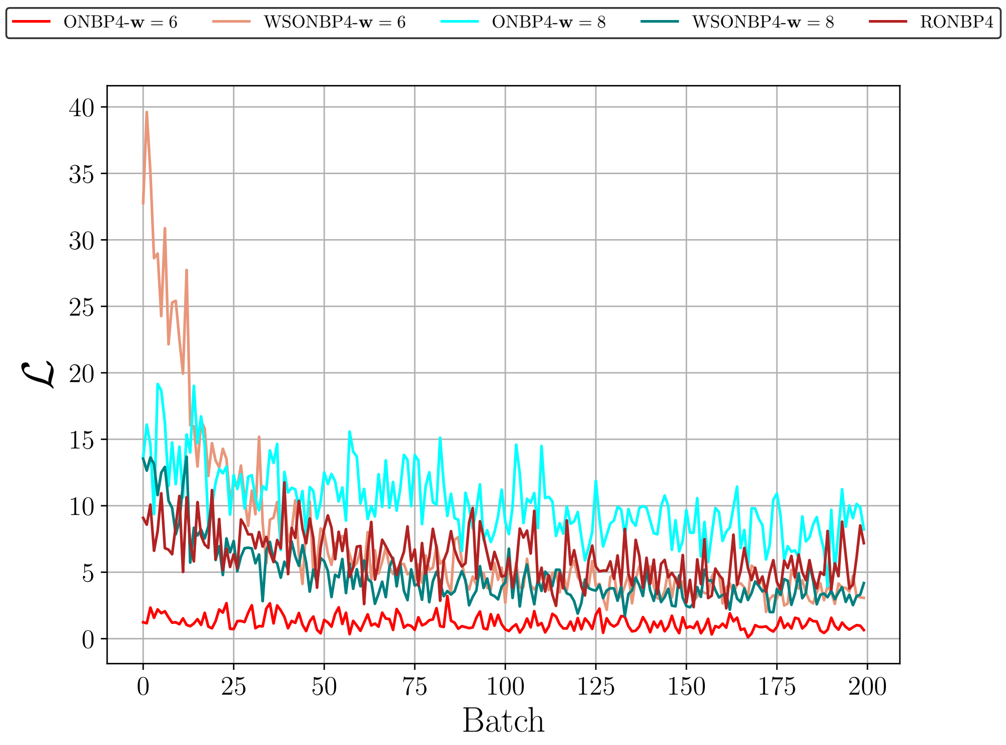


Figure 7.2: The training loss curves for the ONBP4 decoder and its deep learning variations are analyzed for the toric code with $L = 10$. The ONBP4-$\mathbf{w} = 8$ decoder converges to the highest loss value, indicating that its training is the least effective among the decoders considered. In contrast, the ONBP4-$\mathbf{w} = 6$ decoder appears to be the most efficient training (with risk of overfitting altough).Same considerations are applied also for the other values of L.

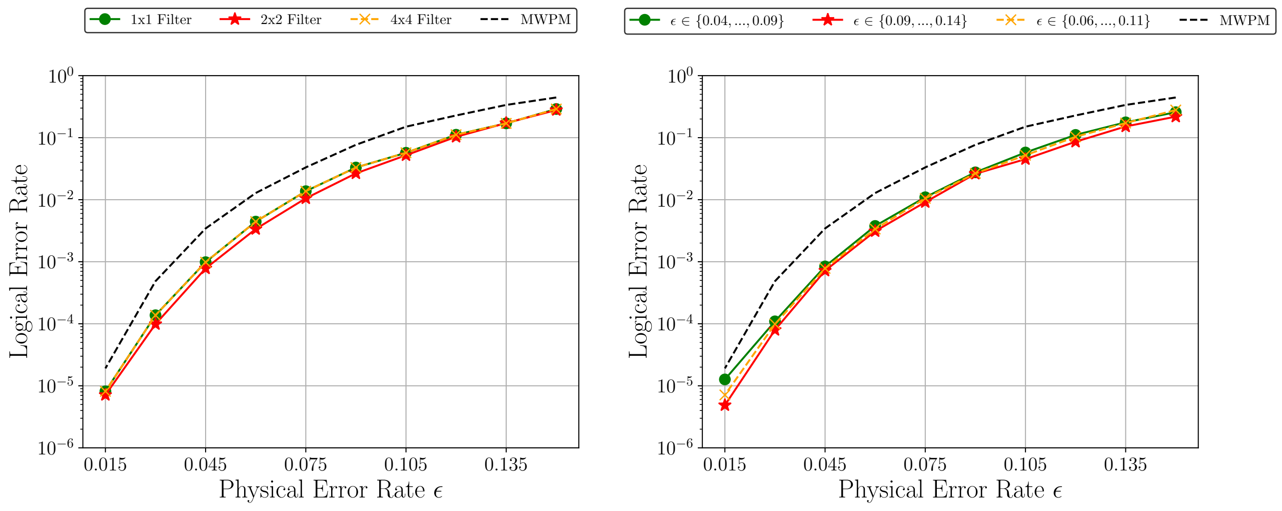


Figure 7.3: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimension $L = 8$ using different filter sizes and ϵ interval.

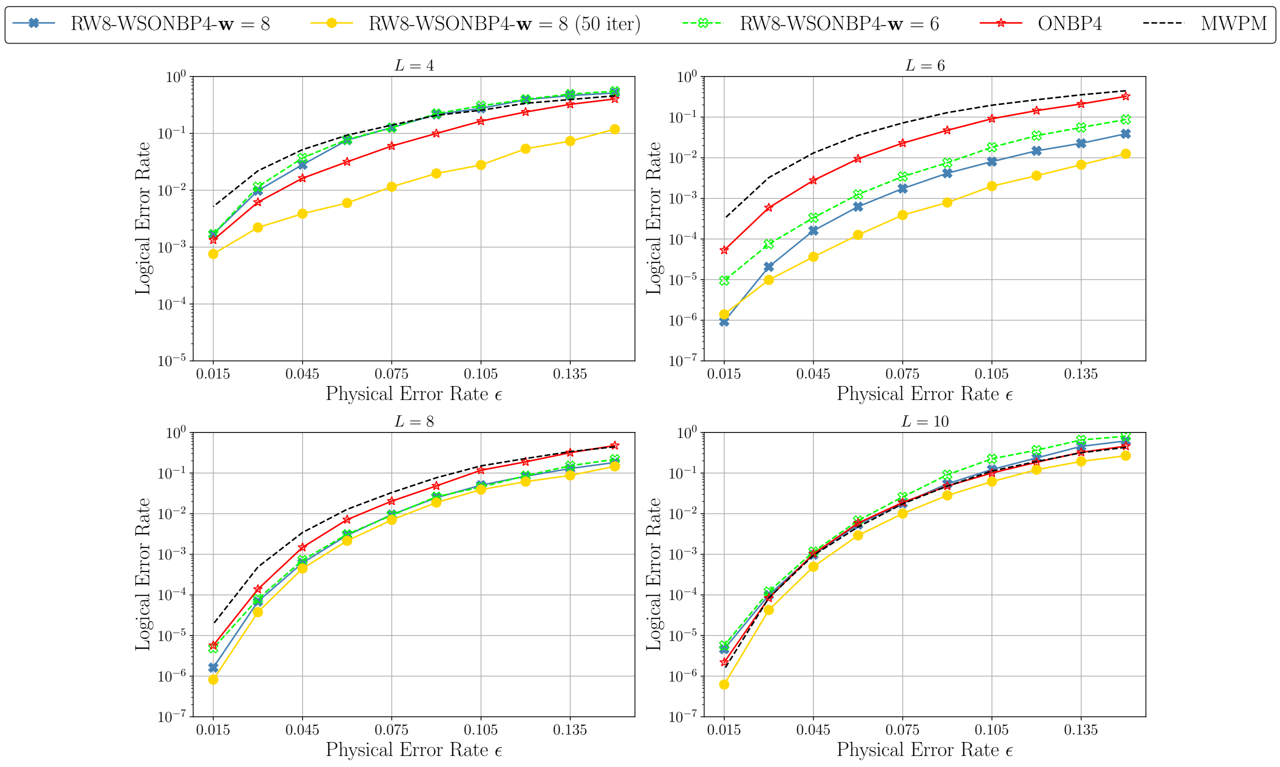


Figure 7.4: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric codes with dimensions $L \in \{4, 6, 8, 10\}$ reusing the weights of L=8 from WSONBP4 decoder.

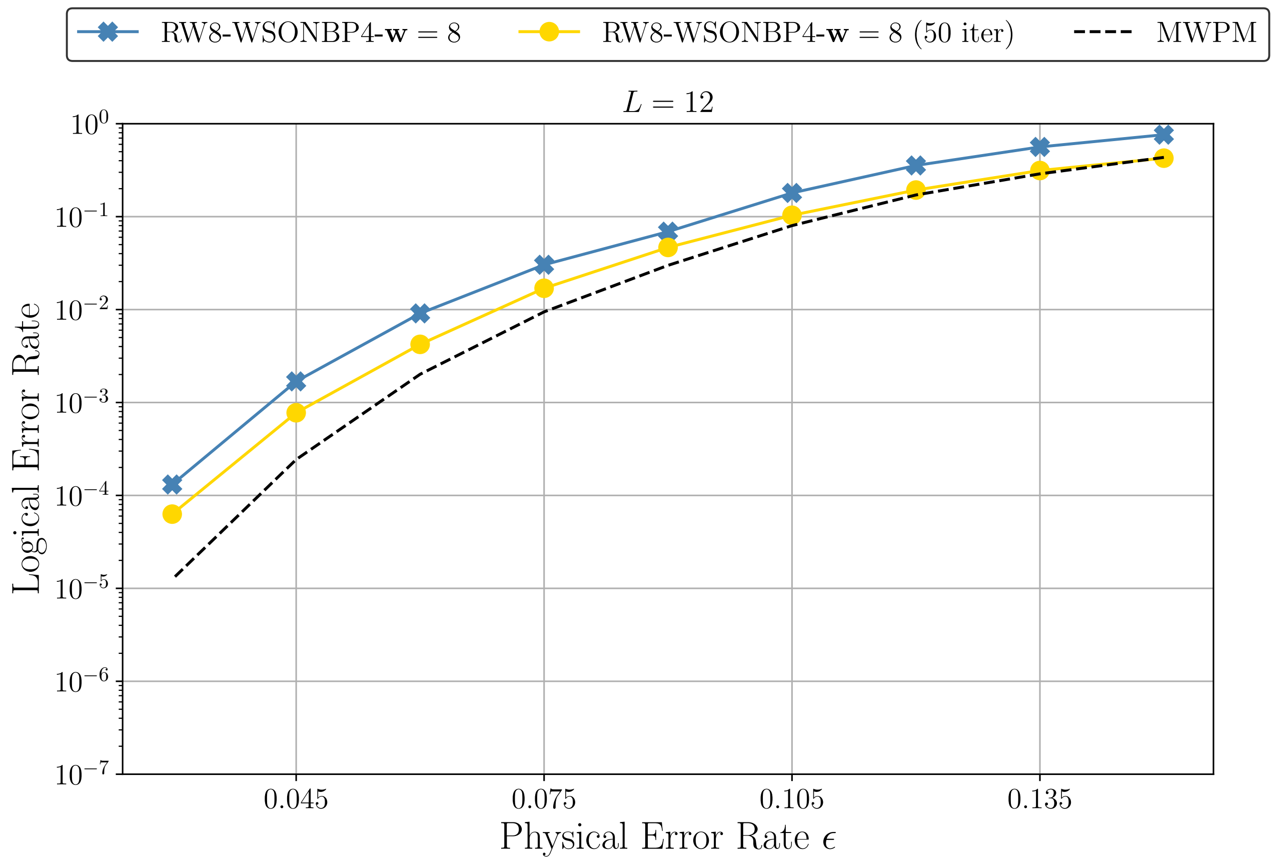


Figure 7.5: Comparison of the Logical Error Rate as a function of the depolarizing probability ϵ for toric code with dimension $L = 12$ reusing the weights of L=8 from WSONBP4 decoder.

Bibliography

[1] Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 341–346. IEEE, 2016.

[2] Sisi Miao, Alexander Schnerring, Haizheng Li, and Laurent Schmalen. Quaternary neural belief propagation decoding of quantum ldpc codes with overcomplete check matrices. *arXiv preprint arXiv:2308.08208*, 2023.

[3] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

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

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

[6] David JC MacKay, Graeme Mitchison, and Paul L McFadden. Sparse-graph codes for quantum error correction. *IEEE Transactions on Information Theory*, 50(10):2315–2330, 2004.

[7] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *Physical review letters*, 122(20):200501, 2019.

[8] Micheal Kastoryano. Lecture notes of the Quantum Error Correction course at University of Cologne. 2018.

[9] Daniel Gottesman. *Stabilizer codes and quantum error correction.* PhD thesis, California Institute of Technology Pasadena, California, 1997.

[10] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *arXiv preprint arXiv:0801.1241*, 2008.

[11] John Preskill. Fault-tolerant quantum computation. In *Introduction to quantum computation and information*, pages 213–269. World Scientific, 1998.

[12] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3(4):044002, 2018.

[13] William Ryan and Shu Lin. *Channel codes: classical and modern.* Cambridge university press, 2009.

[14] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier, 2014.

[15] William Rurik and Arya Mazumdar. Hamming codes as error-reducing codes. In *2016 IEEE Information Theory Workshop (ITW)*, pages 404–408. IEEE, 2016.

[16] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[17] Zunaira Babar, Panagiotis Botsinis, Dimitrios Alanis, Soon Xin Ng, and Lajos Hanzo. Fifteen years of quantum ldpc coding and improved decoding strategies. *IEEE Access*, 3:2492–2519, 2015.

[18] Pavithran Iyer and David Poulin. Hardness of decoding quantum stabilizer codes. *IEEE Transactions on Information Theory*, 61(9):5209–5223, 2015.

[19] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[20] Yun-Jiang Wang, Bao-Ming Bai, and Xin-Mei Wang. Feedback iterative decoding of sparse quantum codes. 2010.

[21] Dimiter Ostrev, Davide Orsucci, Francisco Lázaro, and Balazs Matuz. Classical product code constructions for quantum Calderbank-Shor-Steane codes. *Quantum*, 8:1420, July 2024.

[22] Kao-Yueh Kuo and Ching-Yi Lai. Refined belief-propagation decoding of quantum codes with scalar messages. In *IEEE Globecom Workshops, pages=1–6, year=2020, organization=IEEE.*

[23] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018.

[24] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7(1):11003, 2017.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Nikhil Ketkar et al. Convolutional neural networks. In *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pages 197–242. Manning Publications, 2021.

[27] Jacopo Pegoraro. Lecture notes of the Neural Networks and Deep Learning course at University of Padua. 2024.

[28] Rano Eza Permana. Neural belief propagation for quantum error correcting codes. Master's thesis, Karlsruhe Institute of Technology, 2023.

[29] Antonio deMarti iOlius, Patricio Fuentes, Román Orús, Pedro M Crespo, and Josu Etxezarreta Martinez. Review on the decoding algorithms for surface codes.

[30] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Trans. on Inf. Theory*, 60(2):1193–1202, 2013.