

Logic in Dynamical Stabilizer Quantum Error Correction Codes

by

Sascha Zakaib-Bernier

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Physics - Quantum Information

Waterloo, Ontario, Canada, 2026

© Sascha Zakaib-Bernier 2026

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis is based on the arXiv preprint *Chutes and Ladders: Dynamical Automorphisms via the ZX-calculus* [1]. Sascha Zakaib-Bernier and Dr. Alexander Frei are joint-first authors, while Zach Mann, Dr. Michael Vasmer, and Prof. Victor V. Albert are co-authors. The author of this thesis contributed to the writing and the development of the formalism, proofs and protocols found in the Letter [1] with her joint-first author Dr. Frei. They benefited from the support and advice from the co-authors. MSc Mann helped to compute the LC orbits of the codes used in subsection 4.2.4, and edited the preprint. Dr. Vasmer and Prof. Albert helped with some numerical simulations to compute the code automorphism group in subsection 4.2.4 as well as editing the preprint.

Portions of chapter 1, subsection 2.2.4, section 2.4, section 2.5, chapter 4, and chapter 5 are modified from and overlap with the preprint. The portions that overlap between the two bodies of work were written by the author of this thesis and edited by the co-authors.

Abstract

Quantum error correction enables the protection of quantum information from noise arising from imperfect control and unwanted interactions with the environment. Manipulating logical information in quantum error-correcting codes remains challenging, as we can only directly access physical qubits, not the encoded logical qubits. In fact, we know it is not possible to get the full logical universal gate set by doing transversal physical operations. As a solution to this problem, many schemes for implementing encoded logical gates have been developed, including dynamical automorphism gates (DA). These logical protocols are realized by measurement sequences that transfer logical information between different codespaces. In parallel, there has been a growing interest in the intersection between quantum error correction and the ZX-calculus, a graphical representation of quantum processes. This direction has proved to be fruitful. For example, it has allowed the development of new dynamical codes. However, logic within such codes has not yet been tackled. In this thesis, we offer a formalism for dynamical automorphisms based on the ZX-calculus, implementing logical Clifford gates for stabilizer codes. We begin by defining the basics of quantum information, the stabilizer formalism and the ZX-calculus. Then, we build upon this formalism for gauge fixing via the ZX-calculus. Since gauge fixing may be implemented via measurement, our formalism allows us to protocols implementing dynamical automorphisms. In particular, we start by investigating a DA for a 3-qubit motivating code and its LC orbit. Then, we turn to an example of a logical CZ for the $[[4, 2, 2]]$ code. Finally, we show an example of a distance-preserving dynamical automorphism for the $[[7, 1, 3]]$ bare code implementing a phase gate, which was inaccessible using previous techniques.

Acknowledgements

Foremost, I want to thank my boyfriend Raphaël Berthiaume. Every week, you gave the love and emotional support I needed to complete this degree. This is so cheesy, but you are my rock, and you rock my world. Living life with you is such a privilege.

I want to thank my family, mom and dad, and my friends (Nicholas, Kaleb, Sean, Eames, Dev, Emilia, Kimia, Anya, and Sungjai just to name a few) for picking me up when I'm down. It was hard to move out of the house and start over in a new city far from home, but you made it all easier to get through. Now, I'm ready for round two, PhD!

Next, I want to thank my supervisor Prof. Graeme Smith, and also Dr. Ningping Cao, and Dr. Michael Vasmer. Thank you for taking me under your wing. You have shown me grace and support when I felt most vulnerable. I am incredibly grateful for the time and support you have given me in the past two years. I couldn't have done it without you!

I want to thank Dr. Alexander Frei for guiding me through the open sea, which is the QEC field. I was very overwhelmed by the amount of information I needed to take in coming into this degree, and I really value the time you took to teach me the essentials of the field and work with me on our project.

I also want to thank Zach Mann and Prof. Victor V. Albert for your help with the project, and bearing the innumerable messages I sent about it. I look forward to crossing paths again!

I want to thank Prof. Laflamme without whom this thesis would not have been possible. Your big smile and warm heart made me come to Waterloo. I have felt your support resonate through the last year, and I am deeply grateful for it. Your presence is deeply missed.

Finally, I have to thank Raph again, I love you!

Table of Contents

Author's Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xii
List of Abbreviations	xiii
1 Introduction	1
2 Background	3
2.1 Introduction to Quantum Information Principles	4
2.1.1 Quantum States	4
2.1.2 Quantum Channels	6
2.1.3 Representations of Quantum Channels	7
2.2 Stabilizer Formalism	10
2.2.1 Pauli Operators and Clifford Gates	10

2.2.2	Symplectic Representation	12
2.2.3	Stabilizer States	14
2.2.4	Graph States	15
2.3	Stabilizer codes	19
2.3.1	3-bit Repetition Code	19
2.3.2	3-qubit Repetition Code	19
2.3.3	Quantum Error Models	21
2.3.4	Stabilizer Codes: Definition and Examples	22
2.3.5	Subsystem codes	24
2.3.6	Dynamical Codes	27
2.4	Introduction to the ZX-calculus	31
2.5	Stabilizer Codes in the ZX-Calculus	35
2.5.1	ZX-Encoding Graphs	35
2.5.2	Map-State Duality	38
2.5.3	Finding the Stabilizer Tableau of a ZX-Encoding Graph	39
2.5.4	Constructing the ZX-Encoding Graph of a Stabilizer Code	41
2.5.5	Normal Form of the ZX-Encoding Graphs	43
3	Review of Logic in Stabilizer QEC	46
3.1	Fault-tolerant Quantum Computation	47
3.2	Magic State Distillation	50
3.2.1	Magic State Injection	51
3.2.2	Magic State Distillation using the 15-qubit Reed-Muller Code	52
3.2.3	Contributions and Outlooks	53
3.2.4	Magic State Cultivation	54
3.3	Lattice Surgery	57
3.3.1	Primer on the Surface Code	57
3.3.2	Lattice Merging	59

3.3.3	Lattice Splitting	61
3.3.4	FT Protocol for the CNOT	61
3.3.5	FT protocol for the Phase Gate	62
3.3.6	Contributions and Outlooks	63
3.4	Code Automorphisms	64
3.5	Dynamical Automorphisms	66
4	Dynamical Automorphisms via the ZX-Calculus	69
4.1	Gauge Fixing via the ZX-calculus	70
4.1.1	From Subsystem Codes to Stabilizer Codes	71
4.1.2	From Stabilizer Codes to Subsystem codes	73
4.1.3	A Counting Argument	76
4.1.4	Comparison with Stabilizer Tableau Formalism	78
4.2	Dynamical Automorphisms and ZX-encoding graphs	81
4.2.1	A Simple Nontrivial Logical Action	81
4.2.2	LC Orbits	82
4.2.3	Entangling through Gauge Fixing	84
4.2.4	Bare Code Example	85
4.2.5	Generalization to Logical 2-Qubit Cliffords	88
5	Discussion and Conclusion	90
	References	93

List of Figures

2.1	Encoding and decoding circuit. The encoding channel $\mathcal{E}$ maps k logical qubits $ \psi\rangle$ and $m = n - k$ ancilla qubits to n physical qubits in an encoded state $ \psi_\ell\rangle$ described by the QEC code. Then, the qubits undergo some noise $\mathcal{N}$. In this simplified scenario, the logical information is retrieved using the recovery map $\mathcal{R}$ associated with the QEC code.	4
2.2	Illustration of a state $\rho = \frac{1}{2}(I + \vec{n} \cdot \vec{\sigma})$ in the Bloch Sphere.	11
2.3	Stabilizer state represented by its stabilizer group (a), its graph state (b), and its encoding circuit (c). One may choose any ordering for the vertices, as the graph state is symmetric.	16
2.4	Example of a (a) 5-qubit graph state G onto which we apply a (b) local complementation about the first qubit v_1 .(c) The effect on the subgraph of the neighbouring vertices $N(v_1)$ is isolated from the rest of the graph. . . .	17
2.5	Stabilizers (a) and stabilizer matrix in symplectic representation (b) of the 5-qubit code [83].	23
2.6	Parent subsystem code S with a single gauge qubit, so it bridges three child codes C_1 , C_2 & C_3 for each gauge fixing.	27
2.7	Honeycomb code [33]. (a) Physical qubits are on the vertices of the hexagonal lattice. The three types of plaquette stabilizers are coloured red, green and blue. Edges that connect the same coloured plaquettes are themselves the same colour. The plaquette on the top left illustrates the types of 2-qubit gauge operators along each direction of the lattice. (b) Periodic measurement sequence with time steps r . The regions represent the codespace of each instantaneous stabilizer code. The coloured arrows represent the measurement of gauge operators of the same colour. One of two sets of logical operators is illustrated after each step r	29

2.8	Rules of the ZX-calculus	32
3.1	Active QEC protocol. The state $ \psi\rangle$ holding k logical qubits is encoded into the QEC code as $ \psi_\ell\rangle$ using m ancilla qubits. Then a logical gate $\mathcal{G}$ is performed, followed by a noise channel $\mathcal{N}$. Then a round, or cycle, of QEC is performed by measuring the stabilizers $\mathcal{M}$, and decoding the information to apply the correction $\mathcal{D}$. For circuits of depth D , these last four steps may be repeated D times. Finally, the recovery map decodes the encoded state $U_{\text{circ}} \psi_\ell\rangle$ to get the final output of the quantum circuit implemented. . . .	47
3.2	Illustration of the T state $T +\rangle$ distilled in subsection 3.2.2 on the Bloch sphere. The volume reachable by Clifford operations from a Pauli eigenstate is outlined in the red octahedron [24]. The curved arrows indicate that a π rotation around this state swaps the X and Y operators, so the arrow lies on the XY plane.	50
3.3	Magic state injection protocol for T states. The circuit inputs a given state $ \psi\rangle$, and the magic state $T +\rangle$, and outputs $T \psi\rangle$. The outcome of the Bell measurement in the computation basis is given by the bit a . With this bit, a classical control- V gate is performed on the magic state, where $V = TXT^* = e^{i\pi/4}S^*Y$	51
3.4	Example of a $[[25, 1, 5]]$ rotated surface code. The white dots represent physical qubits. The green(blue) plaquettes represent the Z -type(X -type) stabilizers. The red lines represent the logical operators: a representative of Z_ℓ is a tensor product along the lower X -boundary, while the X_ℓ lies on the right-hand side Z -boundary.	58
3.5	Lattice surgery with the rotated surface code [99]. The stabilizers with bold contour are the ones that were measured (b) to merge the codes or (a) to split the code. The logical operators of each code have support along the red lines.	60
3.6	Sketch of a dynamical automorphism implementing a gate U in the 2-qubit Clifford group using 3 gauge fixings. The blue bubbles represent stabilizer codes. The red arrows represent gauge fixings which use the subsystem codes in the yellow squares. The last arrow represents some LC gate that maps the encoding back to itself up to the logical action U	67

4.1	ZX-encoding graphs of subsystem codes that gauge fix into a stabilizer code. From left to right, we have the ZX-encoding graph of a stabilizer code and, in brackets, the subsystem codes that Z/X/Y gauge fix into the stabilizer code.	75
4.2	Local Clifford (LC) orbit of the 3-qubit motivating code from subsection 4.2.1, whose ZX-encoding graph is on the far right side of the diagram. Here, the equalities between ZX-encoding graphs refer to LC equivalence, whereas the looping arrows represent LC gates that bring the graph to itself. The gates above them keep track of the logical gate induced by the given LC, and we do not keep track of the physical gates they induce. The red arrows correspond to the gauge fixings seen in equations (4.7) & (4.8) for the Y gauge fixing and (4.9) for the X gauge fixing.	83
4.3	Dynamical automorphism of the $[[4, 2, 2]]$ code. We extend the original ZX-encoding graph (a) to construct a subsystem code (b) that Y gauge fixes into the $[[4, 2, 2]]$ code up to some local Clifford gates and a logical CZ gate (highlighted in gray).	85
4.4	Dynamical automorphism for the bare code implementing a logical S gate. (a) Space of $[[7, 1, 3]]$ stabilizer codes, excerpt from [158, Fig. 12]. The yellow squares are subsystem codes S_i that Z/X/Y gauge fix into the stabilizer codes C_i (blue bubbles). (b) ZX-encoding graph of the 7-qubit bare code [157]. We label the logical qubit wire ℓ and the physical qubit wires $1 \dots 7$. (c) Y gauge fixing of S_1 into C_2 . (d) Y gauge fixing of S_2 into C_3 . (e) Y gauge fixing of S_3 into C_4 . (f) X gauge fixing of S_1 into the bare code. (g) Final local Clifford gates and qubit permutations.	87
4.5	Breadth-first search to implement a 2-qubit Clifford gate U . The gray boxes represent the LC orbits of each stabilizer code A , B , C , D , E , & S . On the far left, there is the initial ZX-encoding graph G_C associated with the code C , and on the far right there is the goal ZX-encoding graph $U \circ G_C$. The arrows represent different gauge fixings, performed from both the initial and final ZX-encoding graphs. They eventually meet in the LC orbit of the code S	89

List of Tables

2.1	Syndromes for the 3-qubit repetition code $[3,1,3]$ for bit-flips errors.	20
3.1	Stabilizers of the 15-qubit Reed-Muller code. Each row represents a generator.	52

List of Abbreviations

CP Completely positive

CPTP Completely positive trace-preserving

DA Dynamical automorphism

FT Fault-tolerant

LC Local Clifford

QEC Quantum error correction

QLDPC Quantum low density parity check

TP Trace-preserving

Chapter 1

Introduction

Quantum information promises computations that are intractable with classical computers and communication in a way that allows eavesdroppers to be detected [2]. Despite these promises, quantum technologies are still in their infancy. The main reason for this is the fragility of quantum information; slight perturbations due to the environment can destroy quantum advantages. Quantum error correction (QEC) is an essential step toward scalable quantum computing, as it encodes logical information across many physical qubits in a subspace where errors can be detected and corrected. Recent experiments have demonstrated QEC across various platforms, including superconducting qubits [3–7], trapped ions [8–12], and neutral atoms [13, 14].

While there is still room for improvement in the hardware and QEC techniques, the next step is performing fault-tolerant quantum algorithms. The standard approach is to decompose an algorithm into a small number of quantum gates forming a universal set [2], and then to design fault-tolerant implementations of these gates in a given QEC code. Transversal gates [15] have a tensor-product structure and are therefore naturally fault tolerant, but no QEC code possesses a universal and transversal set of logical gates [16]. As a result, for any QEC code, at least one logical gate must be implemented using a more complex fault-tolerant scheme, such as, e.g. lattice surgery [17–22], magic state distillation [23–27], and code-switching [28–32]. Even so, no consensus yet exists on how to best perform the logical operations necessary to compute any quantum algorithm fault-tolerantly.

The usual QEC paradigm is to encode the logical information into a single subspace: the codespace. However, recent proposals have explored QEC techniques where the system is dynamically driven between different codespaces [33–37]. These dynamical codes enable

QEC using lower-weight measurements, and can also be used to implement logical Clifford gates [38].

Creating dynamical codes out of static codes requires flexibility in terms of resources and depth of the QEC circuits. Decoding also becomes less straightforward when transitioning from a subspace to another. Many have thus turned to the ZX-calculus [39–41], a graphical representation of quantum circuits, to describe these types of protocols.

The ZX-calculus has proved to be useful for perfecting QEC protocols in many instances, mostly for its optimization strengths and ease of interpretation: QEC codes have been translated into the ZX-calculus [42, 43], allowing us to construct new codes [44, 45], and to perform lattice surgery [46] and other techniques [47, 48] to generate logic in QEC codes. Also, Pauli webs, emerging from the ZX-calculus, form detecting regions, which inform us of the measurements needed to detect errors in particular circuits [37]. Furthermore, a lot of space-time, Floquet and dynamical codes have emerged from the ZX-calculus representation [49–53].

In this thesis, we develop a ZX-calculus-based formalism for dynamical automorphisms for the implementation of logical Clifford gates. Complementing prior ZX-calculus applications that address the *construction* of dynamical codes [33, 34, 37, 53–55], we leverage ZX-calculus to design and implement their *automorphisms*. We expect that the machine-interpretability of this formalism will facilitate the search for new dynamical codes and logical gates.

The key ingredient of our approach is a ZX-calculus representation of gauge fixing [29, 56], a technique to transfer encoded information between stabilizer codes via measurement-based shortcuts, which may fault-tolerantly implement high depth unitary circuits. We give three examples of dynamical automorphisms via gauge fixing: the logical Clifford group for a $[[3, 1]]$ motivating code, an entangling logical CZ gate for the error-detecting $[[4, 2, 2]]$ code, and a logical phase gate for the error-correcting $[[7, 1, 3]]$ bare code [57].

This manuscript is structured as follows. First, we lay the foundations for the results in chapter 2. We introduce the basics of quantum information, stabilizer formalism, stabilizer codes, the ZX-calculus, and finally stabilizer codes represented in the ZX-calculus. Next, chapter 3 reviews some well-known protocols for fault-tolerant logical gates, such as magic state distillation, lattice surgery, code automorphisms and dynamical automorphisms. This will lead us into the results, i.e. dynamical automorphisms via the ZX-calculus. Chapter 4 first explains gauge fixing in the ZX-calculus before giving the concrete examples of dynamical automorphisms listed above. Finally, chapter 5 closes off this manuscript with a discussion of the results, future work, and comparisons with other protocols.

Chapter 2

Background

This chapter establishes the foundational concepts leading up to the results. We start off by introducing some quantum information basics in section [2.1](#). Then, section [2.2](#) discusses the stabilizer formalism onto which stabilizer codes are based, which is the subject of the next section, section [2.3](#). After, we take a detour to introduce the ZX-calculus, a graphical representation of quantum processes in section [2.4](#). Finally, this guides us to section [2.5](#), where we represent stabilizer codes in the ZX-calculus through their ZX-encoding graphs.

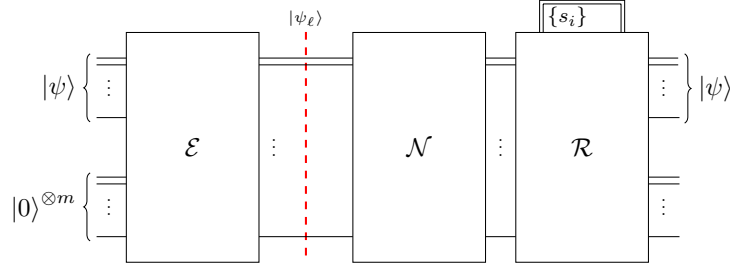


Figure 2.1: Encoding and decoding circuit. The encoding channel $\mathcal{E}$ maps k logical qubits $|\psi\rangle$ and $m = n - k$ ancilla qubits to n physical qubits in an encoded state $|\psi_{\ell}\rangle$ described by the QEC code. Then, the qubits undergo some noise $\mathcal{N}$. In this simplified scenario, the logical information is retrieved using the recovery map $\mathcal{R}$ associated with the QEC code.

2.1 Introduction to Quantum Information Principles

In quantum error correction, we aim to encode a k -qubit quantum state into a larger space of n -qubits in a way that we can correct and detect errors due to noise. The process is illustrated by the circuit in figure 2.1: the encoding map $\mathcal{E}$ projects the logical state and the ancilla $|\psi\rangle \otimes |0\rangle^{\otimes m}$ into $|\psi_{\ell}\rangle$. Once protected, the qubits may undergo some known noise $\mathcal{N}$, and the recovery map $\mathcal{R}$ decodes and corrects the logical information. Each map brings a quantum state to another, i.e. they are channels. The goal of this section is to introduce all the relevant information to understand QEC encoding maps as quantum channels. Before defining them, we introduce some of the notation used to describe quantum states. We will end this section by introducing representations of quantum channels: the Choi-representation of a quantum channel, and the Kraus representation. For a complete introduction to quantum information, we refer the reader to John Watrous's book, *The Theory of Quantum Information* [58], on which this section is based.

2.1.1 Quantum States

Although quantum states are first introduced in undergraduate courses as vectors of a Hilbert space $|\psi\rangle \in \mathcal{H}$, quantum information lends itself better to using density operators $\rho \in \mathcal{B}(\mathcal{H})$, where the Banach space $\mathcal{B}$ is a complete normed vector space [59]. These linear operators allow us to describe probabilistic mixtures of quantum states.

Suppose we are given either $|\psi_1\rangle$ with probability p_1 or $|\psi_2\rangle$ with probability p_2 , such that $\sum_{i \in \{1,2\}} p_i = 1$. Then, we measure the observable A , a Hermitian operator. Using the

spectral decomposition, we know

$$A = \sum_{a \in \Lambda} a \Pi_a, \quad (2.1)$$

where the Π_a are projectors and Λ is the set of real eigenvalues of A . So, the probability of obtaining an outcome a after the measurement is given by

$$\Pr(a) = p_1 \Pr(a|\psi_1) + p_2 \Pr(a|\psi_2). \quad (2.2)$$

The conditional probability for each pure state is given by $\Pr(a|\psi_i) = \langle \psi_i | O_a | \psi_i \rangle = \text{Tr}(O_a |\psi_i\rangle \langle \psi_i|)$. Because the trace is a linear function, we have that

$$\Pr(a) = \text{Tr}(O_a [p_1 |\psi_1\rangle \langle \psi_1| + p_2 |\psi_2\rangle \langle \psi_2|]). \quad (2.3)$$

The expression in square brackets corresponds to the density operator ρ of our original state. So the density operator is a convex combination of pure states, normalized such that $\text{Tr}(\rho) = 1$ [59].

Let us now introduce formal definitions. The operators of the complex Euclidean space $\mathcal{X}$ are all the linear mappings to itself, denoted $L(\mathcal{X})$, and the complex conjugate of X is denoted X^* . We know that Hermitian operators are equal to their complex conjugate, but we will need to restrict more the set of operators we work with if we aim to describe states.

Definition 1 ([58]). *An operator is positive semidefinite if $X = Y^*Y$ for some Y operator $Y \in L(\mathcal{X})$. The set of positive semi-definite operators acting on $\mathcal{X}$ is denoted $\text{Pos}(\mathcal{X})$.*

$$\text{Pos}(\mathcal{X}) = \{Y^*Y : Y \in L(\mathcal{X})\} \quad (2.4)$$

We note that every positive semidefinite operator is Hermitian: $X^* = (Y^*Y)^* = Y^*(Y^*)^* = Y^*Y = X$.

Quantum states can be represented as state vectors $|\psi\rangle$ or as density operators $\rho \in L(\mathcal{X})$. For pure states, $\rho = |\psi\rangle \langle \psi|$, while for mixed states, $\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i|$ where $\{p_i\}$ represent probabilities. As such, density operators are Hermitian, but their eigenvalues must be real, and also add to 1 since they represent probabilities.

Definition 2 ([58]). *Density operators are positive semidefinite operators that trace to 1. The set of density matrices of a complex Euclidean space $\mathcal{X}$ is denoted $D(\mathcal{X})$.*

$$D(\mathcal{X}) = \{\rho \in \text{Pos}(\mathcal{X}) : \text{Tr}(\rho) = 1\} \quad (2.5)$$

In this work, we will focus on qubits. Each qubit's state is described by two complex numbers. As such, a n -qubit system lives in $\mathcal{X} = \mathbb{C}^{2^n}$. Without loss of generality, we will choose to work in the computational basis $\{0, 1\}^n$.

When the qubits go through a QEC protocol, the tool used to compare the input and output is the fidelity. This measure can also be used to compare quantum operators.

Definition 3 ([58]). *Let $P, Q \in \text{Pos}(\mathcal{X})$, for $\mathcal{X}$ complex Euclidean space. The fidelity $F(P, Q)$ between P and Q is defined as*

$$F(P, Q) := \left\| \sqrt{P} \sqrt{Q} \right\|_1 = \text{Tr} \left(\sqrt{\sqrt{Q} P \sqrt{Q}} \right), \quad (2.6)$$

where $\|\cdot\|_1$ is the trace norm defined as $\|A\|_1 := \text{Tr} \left(\sqrt{A^* A} \right)$.

In QEC, it will be used to compare the ideal logical state ρ_{ideal} from the noisy one ρ_{noisy} . If a QEC scheme works well, then fidelity $F(\rho_{\text{ideal}}, \rho_{\text{noisy}})$ is close to 1. On the other hand, if $F(\rho_{\text{ideal}}, \rho_{\text{noisy}}) = 0$, then it means that the two states are orthogonal to each other, so the logical information was lost completely.

2.1.2 Quantum Channels

Quantum channels, or simply channels, are a subset of linear maps of operators Φ , denoted by the set

$$\mathcal{T}(\mathcal{X}, \mathcal{Y}) = \{\Phi : \mathcal{L}(\mathcal{X}) \rightarrow \mathcal{L}(\mathcal{Y})\}, \quad (2.7)$$

with specific properties. We first introduce positive maps, which send positive operators to positive operators.

Definition 4 ([58]). *A map $\Phi \in \mathcal{T}(\mathcal{X}, \mathcal{Y})$ is positive if $\Phi(P) \in \text{Pos}(\mathcal{Y})$ for every $P \in \text{Pos}(\mathcal{X})$.*

This allows us to understand completely positive maps.

Definition 5 ([58]). *A map Φ is completely positive (CP) if $\Phi \otimes \mathbb{1}_{\mathcal{L}(\mathcal{Z})}$ is positive for all complex Euclidean spaces $\mathcal{Z}$.*

This definition may seem redundant comparatively to positive maps, but there exist positive maps that are not completely positive, notably the transpose map. This means

that taking the partial transpose of a bipartite state $\rho \in D(\mathcal{X} \otimes \mathcal{Y})$ could produce send the initial density matrix outside of $D(\mathcal{X} \otimes \mathcal{Y}) \subseteq L(\mathcal{X} \otimes \mathcal{Y})$ ¹. These types of map aren't channels, as they don't send all quantum states to other quantum states.

Finally, we need one last type of map to define channels, i.e. trace-preserving maps.

Definition 6 ([58]). *A map $\Phi \in T(\mathcal{X}, \mathcal{Y})$ is trace-preserving (TP) if $\text{Tr}(\Phi(X)) = \text{Tr}(X)$ for all $X \in L(\mathcal{X})$.*

A TP map will thus send a density operator to another operator σ with trace equal to 1. For σ to be in $D(\mathcal{Y})$, it needs to also be in $\text{Pos}(\mathcal{Y})$, so the map needs to also be CP. So we conclude that a quantum channel is a CPTP map and vice versa.

Definition 7 ([58]). *A quantum channel is a linear map $\Phi : L(\mathcal{X}) \rightarrow L(\mathcal{Y})$ for some choice of complex Euclidean spaces $\mathcal{X}$ & $\mathcal{Y}$, satisfying two properties:*

1. *Completely positive,*
2. *Trace-preserving.*

We denote the set of quantum channels, or CPTP maps, as $\text{CPTP}(\mathcal{X}, \mathcal{Y})$.

2.1.3 Representations of Quantum Channels

So, we now understand that QEC encoding maps are CPTP maps. This subsection introduces a more concrete description of such objects, i.e. their representation. There are many ways to represent channels: the natural representation, the Choi representation, the Kraus representation, and the Stinespring dilation. While every representation may be useful, we will only discuss the Choi and the Kraus representations, as they will be relevant in this work. The Choi representation will be useful to understand the mapping between quantum channels and quantum states, while the Kraus representation will be used to describe noise. We point out that all representations can be converted into one another.

¹In the case of the partial transpose, all states ρ that are sent to a state that's not positive semidefinite operator are entangled, so the transpose can be used as a criterion to test for entanglement of a general bipartite state. We point out that the converse isn't guaranteed: if the partial transpose does send the state to a positive semidefinite operator, the state may be entangled or not [58].

Choi Representation

The Choi representation $J(\Phi)$ of a linear map $\Phi \in \mathcal{T}(\mathcal{X}, \mathcal{Y})$ is an operator acting on a bigger space $\mathcal{L}(\mathcal{Y}, \mathcal{X})$:

$$J(\Phi) = \frac{1}{2^n} \sum_{a,b \in \{0,1\}^n} \Phi(|a\rangle \langle b|) \otimes |a\rangle \langle b|, \quad (2.8)$$

where we have a superposition of all possible the inputs for our n -qubit system on the right-hand side, and all possible outputs of the channel on the left-hand side. Since the Choi representation $J(\Phi)$ is entirely determined by Φ , there is an isomorphism between a linear map and its Choi representation.

The Choi representation is particularly interesting for CPTP channels because then $J(\Phi)$ is a density operator itself. In fact, if $\Phi \in \text{CPTP}(\mathcal{X}, \mathcal{Y})$, then $\Phi(|a\rangle \langle b|) \in \text{Pos}(\mathcal{Y})$ since $|a\rangle \langle b| \in \text{Pos}(\mathcal{X})$. By linearity, $J(\Phi)$ itself is positive semidefinite. Also, the trace of these operators is preserved. Since $\text{Tr}(|a\rangle \langle b|) = \delta_{ab}$, then

$$\text{Tr}(J(\Phi)) = \text{Tr}\left(\frac{1}{2^n} \sum_{a,b \in \{0,1\}^n} \Phi(|a\rangle \langle b|) \otimes |a\rangle \langle b|\right), \quad (2.9)$$

$$= \frac{1}{2^n} \sum_{a,b \in \{0,1\}^n} \text{Tr}(\Phi(|a\rangle \langle b|)) \text{Tr}(|a\rangle \langle b|) \text{ by linearity}, \quad (2.10)$$

$$= \frac{1}{2^n} \sum_{a \in \{0,1\}^n} 1 = 1. \quad (2.11)$$

So, we conclude $J(\Phi) \in \mathcal{D}(\mathcal{Y} \otimes \mathcal{X})$. This isomorphism between channels and states has many names, like the Jamiolkowski isomorphism, the Choi-Jamiolkowski isomorphism, the channel-state duality or the map-state duality.

In particular, we take the example where $\mathcal{X} = \mathbb{C}^2$, i.e. we have a single qubit as input to our CPTP map $n = 1$. Then, by pulling out Φ from the state, its Choi representation may be elegantly simplified:

$$J(\Phi) = \frac{1}{2}(\Phi \otimes \mathbb{1}_{\mathcal{L}(\mathbb{C}^2)}) \sum_{a,b \in \{0,1\}} |aa\rangle \langle bb|, \text{ where } |aa\rangle := |a\rangle \otimes |a\rangle, \quad (2.12)$$

$$= \frac{1}{2}(\Phi \otimes \mathbb{1}_{\mathcal{L}(\mathbb{C}^2)})(|00\rangle \langle 00| + |00\rangle \langle 11| + |10\rangle \langle 00| + |11\rangle \langle 11|), \quad (2.13)$$

$$= (\Phi \otimes \mathbb{1}_{\mathcal{L}(\mathbb{C}^2)}) |\varphi^+\rangle \langle \varphi^+|, \quad (2.14)$$

The Choi representation corresponds to applying the map Φ on half of a 2-qubit maximally entangled state, $|\varphi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. More generally, for a finite arbitrary number of qubits as input, the Choi representation will be to apply the map on half of a bipartite, maximally entangled state.

Kraus Representation

The Kraus representation of a quantum channel $\Phi \in \mathcal{T}(\mathcal{X}, \mathcal{Y})$ acting on $\rho \in \mathcal{D}(\mathcal{X})$ is the conjugation of an input state ρ by some set of Kraus operators $\{A_i\}$:

$$\Phi(X) = \sum_{i \in \{0,1\}^n} A_i X A_i^* \quad \forall X \in \mathcal{L}(\mathcal{X}), \text{ where } \sum_{i \in \{0,1\}^n} A_i^* A_i = \mathbb{1}_{\mathcal{X}}. \quad (2.15)$$

The Kraus operators conjugate the operator X instead of having different left and right operators because the channel is CP. Since this proof is a bit longer, we refer the reader to Watrous's textbook [58]. The condition $\sum_{i \in \{0,1\}^n} A_i^* A_i = \mathbb{1}_{\mathcal{X}}$ comes from the fact that the channel is TP. The proof is simple enough for CPTP channels:

$$\text{Tr}(\Phi(X)) = \text{Tr}\left(\sum_{i \in \{0,1\}^n} A_i X A_i^*\right), \quad (2.16)$$

$$= \sum_{i \in \{0,1\}^n} \text{Tr}(A_i X A_i^*) \text{ by linearity}, \quad (2.17)$$

$$= \sum_{i \in \{0,1\}^n} \text{Tr}(A_i^* A_i X) \text{ by cyclicity of the trace}, \quad (2.18)$$

$$= \text{Tr}\left(\left[\sum_{i \in \{0,1\}^n} A_i^* A_i\right] X\right) \text{ by linearity}, \quad (2.19)$$

$$\text{but also, } = \text{Tr}(X) \implies \sum_{i \in \{0,1\}^n} A_i^* A_i = \mathbb{1}_{\mathcal{X}}. \quad (2.20)$$

It can be easily adapted to the case where the map is only TP, so the left and right operators may be different.

The Kraus operators may describe any map in $\mathcal{T}(\mathcal{X}, \mathcal{Y})$, but they are not unique.

In this work, Kraus operators will be used to describe noise channels $\mathcal{N}(\rho) = \sum_{i \in \{0,1\}^n} A_i \rho A_i^*$. Then, the Kraus operators can be interpreted as $A_i = \sqrt{p_i} E_i$, where p_i is the probability of the error E_i occurring on our system.

2.2 Stabilizer Formalism

This section introduces stabilizer formalism, on which stabilizer codes are based. In subsection 2.2.1, we define the Pauli operators and Clifford gates, operators that make up this formalism. Then, subsection 2.2.2 defines the symplectic representation that will be used for the rest of this manuscript. Finally, we will discuss stabilizer states in subsection 2.2.3, which leads us to a particular subclass, the graph states, in subsection 2.2.4.

2.2.1 Pauli Operators and Clifford Gates

The Pauli operators are the generators of a non-Abelian group, the Pauli group :

$$\mathcal{P} = \langle X, Y, Z \rangle, \quad (2.21)$$

where we have our usual matrices

$$X := \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y := \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z := \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad (2.22)$$

which have a period of 2, and the following commutation relations: $[\sigma_i, \sigma_j] = 2i\epsilon_{ijk}\sigma_k$, or the anti-commutation relation using the Poisson brackets $\{\sigma_i, \sigma_j\} = 0$ [58]. The Pauli operator σ_i corresponds to π -rotation about the i th axis of the Bloch sphere $\exp -i\pi\sigma_i/2$.

It is important to note that the Pauli operators, with identity, form a complete orthonormal basis for the operators on $\mathbb{C}^2$ [59]. In fact, one can write the density matrix of a single-qubit system as $\rho = \frac{1}{2}(I + \vec{n} \cdot \vec{\sigma})$, where $\vec{n}$ is a vector and $\vec{\sigma}^T = (\sigma_x, \sigma_y, \sigma_z)$. The when $\|\vec{n}\| = 1$, the state is pure, while if $0 < \|\vec{n}\| < 1$ the state is mixed. Thus, the arrow lies either on the surface or within the volume of the Bloch sphere. Similarly, one can express any unitary operator with its determinant equal to 1 as $U = e^{-i\varphi\vec{n} \cdot \vec{\sigma}} \in SU(2)$, with θ and $\vec{n}$'s components being real, and with $\|\vec{n}\| = 1$.

The n -qubit Pauli group is spanned by Pauli strings of length n [59]:

$$\mathcal{P}_n = \langle X, Y, Z \rangle^{\otimes n}. \quad (2.23)$$

Combined with identity, it also forms a complete and orthogonal basis for n -qubit states. The weight w of a Pauli string is defined as the number of non-identity operators in the string [60], meaning ZXI has weight $w = 2$.

The Clifford group is the set of operators that normalize $\mathcal{P}_n$, i.e. that map $\mathcal{P}_n$ to itself under conjugation [61].

$$\mathcal{C}_2^{(n)} = \{G \in \mathcal{U} | G\mathcal{P}_n G^* = \mathcal{P}_n\}, \quad (2.24)$$

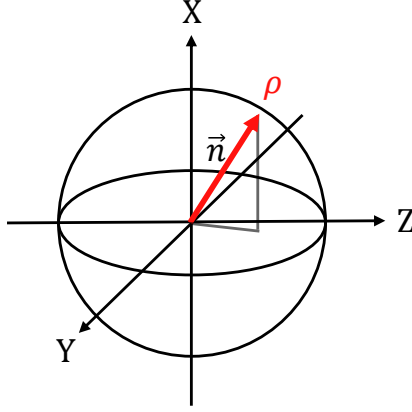


Figure 2.2: Illustration of a state $\rho = \frac{1}{2}(I + \vec{n} \cdot \vec{\sigma})$ in the Bloch Sphere.

where $\mathcal{U}$ is the set of unitary operators on $\mathbb{C}^{2n}$.² We call the operators in this set the Clifford gates. The single-qubit Clifford group is generated by $\{\sqrt{X}, \sqrt{Z}\}$. We will also refer to this group as the local Clifford (LC) gates, as it acts on a single qubit. The gates correspond to the $\pi/2$ -rotation about the X or Z axis, respectively:

$$\sqrt{X} := e^{-i\pi X/4} = \frac{1}{\sqrt{2}}(\mathbb{1} - iX), \text{ and } \sqrt{Z} := e^{-i\pi Z/4} = \frac{1}{\sqrt{2}}(\mathbb{1} - iZ). \quad (2.25)$$

The $\sqrt{Z}$ gate is equivalent up to a global phase to the phase gate S :

$$S := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \text{ and } S = e^{i\pi/4}\sqrt{Z}. \quad (2.26)$$

These operators generate a $\pi/2$ -rotation about the Y axis $\sqrt{Y} = \sqrt{X}\sqrt{Z}\sqrt{X} = \sqrt{Z}\sqrt{X}\sqrt{Z}$. These $\sqrt{\sigma_i}$ gates generate $\pi/2$ -rotations on the Bloch sphere about the i th axis that swaps the two other Pauli operators:

$$\sqrt{\sigma_i}\sigma_j\sqrt{\sigma_i}^{-1} = \epsilon_{ijk}\sigma_k \text{ for } i \neq j \neq k \neq i, \quad (2.27)$$

where $\sqrt{A}^{-1} := \sqrt{A}^*$. In particular, $\sqrt{Y}$ will swap X and Z . Another well-known gate has the same effect: the Hadamard gate.

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}}(Z + X). \quad (2.28)$$

²The lower subscript 2 refers to the second level of the Clifford hierarchy discussed below.

Its determinant is -1, so it cannot be in $SU(2)$ unlike $\sqrt{Y}$ which corresponds to a rotation [62]. Instead, H is a reflection that swaps X and Z . It is possible to define the equivalent operators for the other two rotation operators $\sqrt{\sigma_i} \leftrightarrow 1/\sqrt{2}(\sigma_j + \sigma_k)$ with $i \neq j \neq k \neq i$.

Since the Hadamard gate has been historically used rather than $\sqrt{Y}$, we will exclusively use it from now on. To summarize, the 1-qubit Clifford gates, or LC gates, are $\mathcal{C}_2^{(1)} = \{\sqrt{X}, H\} = \{\sqrt{Z}, H\}$.

The LC gates's usefulness is limited due to their local action. To start to include entangling operations, one must look at the 2-qubit Clifford group. The single-qubit gates are part of this group, since they also work on each qubit individually, but there are additional gates that map 2-qubit Pauli operators to each other. More precisely, the 2-qubit Clifford group is generated by $\mathcal{C}_2^{(2)} = \{\sqrt{Z}, H, \text{CNOT}\}$ or a CZ can be used equivalently:

$$\begin{array}{c} \bullet \\ | \\ \oplus \\ | \\ \bullet \end{array} = (|0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes X) \quad \& \quad \begin{array}{c} \bullet \\ | \\ \bullet \\ | \\ \bullet \end{array} = (|0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes Z). \quad (2.29)$$

We point out that the SWAP operator is a product of three CNOTs, so it is a Clifford operator:

$$\text{SWAP} = \begin{array}{c} \bullet \quad \oplus \quad \bullet \\ | \quad | \quad | \\ \oplus \quad \bullet \quad \oplus \end{array}. \quad (2.30)$$

Entangling operations such as the CNOT increase the range of computation, but to achieve a universal gate set, one must have the 2-qubit Clifford group *and* a non-Clifford gate [63]. To define non-Clifford gates, we discuss the Clifford hierarchy, first introduced by Gottesman and Chuang [64]. The first level is the Pauli group $\mathcal{C}_1^{(n)} = \mathcal{P}_n$. Next, the conjugation of the Pauli group by gates in the n -th level of the Clifford hierarchy is in the previous level. So the second level is $\mathcal{C}_2^{(n)}$ as defined in equation (2.24), and the third level is

$$\mathcal{C}_3^{(n)} = \{G \in \mathcal{U} | G\mathcal{P}_n G^* \subseteq \mathcal{C}_2^{(n)}\}, \quad (2.31)$$

and so on. Non-Clifford gates are gates that are in the third level of the hierarchy or higher. Notable examples are the T gate, the Toffoli gate, and the CCZ gate. We take the time to define the T gate:

$$T := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} = \sqrt{S}. \quad (2.32)$$

2.2.2 Symplectic Representation

In this work, we aim to morph Pauli strings to generate logical Clifford gates. Since Pauli strings conjugated by anti-commuting Pauli operators change sign, which is not the action

we are looking for, we will choose to ignore conjugation by Pauli operators. In other words, we will identify the Clifford group modulo Pauli operators to the symplectic representation [15, 65, 66]. We will first introduce the symplectic representation for completeness, but the important takeaways from the subsection in the context of this work are discussed after.

Introduction to the Symplectic Representation

To introduce the symplectic representation and its connection to the Clifford group, we follow Aram Harrow's lecture notes [67, 68]. Suppose one has two n -qubit Pauli strings P & Q . Then, one can express them as

$$P = (-1)^{a_1} X^{b_1} Z^{c_1} \text{ \& } Q = (-1)^{a_2} X^{b_2} Z^{c_2}, \text{ for } a_1, a_2 \in \mathbb{F}_2, b_1, b_2, c_1, c_2 \in \mathbb{F}_2^n. \quad (2.33)$$

Then, the product of the two operators may be written as

$$PQ = (-1)^{(a_1+a_2)} X^{b_1} Z^{c_1} X^{b_2} Z^{c_2} \quad (2.34)$$

$$= (-1)^{(a_1+a_2+(b_2, c_1))} X^{(b_1+b_2)} Z^{(c_1+c_2)}, \quad (2.35)$$

where (b_2, c_1) refers to the inner product of the $\mathbb{F}_2^n$ -linear algebra. Similarly,

$$QP = (-1)^{(a_1+a_2+(b_1, c_2))} X^{(b_1+b_2)} Z^{(c_1+c_2)}, \quad (2.36)$$

such that one may write PQ in function of QP :

$$PQ = (-1)^{(a_1+a_2+(b_1, c_2)+(b_2, c_1))} QP. \quad (2.37)$$

From this phase, the symplectic inner product emerges:

$$(b_1, c_2) + (b_2, c_1) = (b_1^T c_1^T) \begin{pmatrix} 0_n & I_n \\ I_n & 0_n \end{pmatrix} \begin{pmatrix} b_2 \\ c_2 \end{pmatrix} \quad (2.38)$$

$$= (b_1^T c_1^T) \Lambda \begin{pmatrix} b_2 \\ c_2 \end{pmatrix} \quad (2.39)$$

$$= (v, w)_S \quad (2.40)$$

where the symplectic inner product is defined for the vectors $v^T = (b_1^T c_1^T)$, $w^T = (b_2^T c_2^T)$. If $(v, w) = 0$, then equation (2.37) implies that the operators commute, $[P, Q] = 0$.

Now, how does this relate to the Clifford group? We know that Clifford gates map Pauli operators to each other through conjugation. Let us rename $\sigma^v = X^{b_1} Z^{c_1}$ and $\sigma^w = X^{b_2} Z^{c_2}$. Then for some $U \in \mathcal{C}_2^{(n)}$,

$$U \sigma^v U^* = (-1)^{f(v)} \sigma^{g(v)}, \quad (2.41)$$

for some functions f, g . We do not define f as it will not be useful. Instead, the linear property of g is examined. From equations (2.35) and (2.40), one may see that $\sigma^v \sigma^w \propto \sigma^{(v+w)}$ up to a phase. Let us apply U to this operator:

$$U \sigma^{(v+w)} U^* \propto U \sigma^v \sigma^w U^* \quad (2.42)$$

$$= U \sigma^v U^* U \sigma^w U^*, \text{ but using equation (2.41),} \quad (2.43)$$

$$\propto \sigma^{g(v)} \sigma^{g(w)}, \quad (2.44)$$

but also, $U \sigma^{(v+w)} U^* \propto \sigma^{g(v+w)}$, so we conclude that g is linear.

Then, it can be expressed as $g(v) = Mv$, where $M \in \mathbb{F}_2^{2n}$. Since the conjugation by Clifford gates doesn't change the commutation relations of the Pauli strings:

$$(v, w)_S = v^T \Lambda w, \text{ but also,} \quad (2.45)$$

$$= (Mv, Mw) = (Mv)^T \Lambda Mw = v^T M^T \Lambda Mw, \quad (2.46)$$

implying that $\Lambda = M^T \Lambda M$. This is the definition of the symplectic form. Symplectic matrices M preserve the inner product defined by Λ , the symplectic inner product. More importantly, we have shown that it is possible to find such symplectic matrices to describe the effect of Clifford gates on Pauli strings represented by binary vectors.

The Consequences of the Representation

The obvious implication of the symplectic representation is that we lose information about the phase of the Pauli strings. However, as mentioned at the beginning of the subsection, we do not care about this phase. In practical settings, it can easily be monitored by classical bookkeeping.

A more subtle consequence is that we will not distinguish between single-qubit Clifford gates and their inverses. In fact, normally $\sqrt{X}^{-1} = e^{i\pi/4X} = iX\sqrt{X}$. Since the Pauli operators change the sign of the Pauli strings under conjugation—and we do not keep track of the phase—we will effectively consider $\sqrt{X}^{-1} = \sqrt{X}$. Then, all the single-qubit Cliffords will square to identity: $I = \sqrt{X}^{-1} \sqrt{X} = (\sqrt{X})^2$, and similarly for $\sqrt{Z}$.

2.2.3 Stabilizer States

The stabilizer group is an Abelian subgroup of the n -qubit Pauli group without $-I$ [15, 69]. The stabilizers, s_i are the generators of the stabilizer group:

$$S = \langle s_1, s_2, \dots, s_m \rangle, \quad (2.47)$$

for some non-zero integer m . We give an example of such a generating set in figure 2.3a. Because Pauli strings are Hermitian, the stabilizers are observables, and can be measured. Because they commute, they span a common subspace $\{|\psi_S\rangle\}$, for which

$$s_i |\psi_S\rangle = \lambda_i^S |\psi_S\rangle, \forall s_i \in S, \quad (2.48)$$

where $\{\lambda_i^S\}$ are the eigenvalues associated with the state $|\psi_S\rangle$. This subspace comprises the stabilizer states of this stabilizer group. Since the s_i are Pauli strings, their eigenvalues are ± 1 . A stabilizer state can be prepared from an arbitrary state by measuring all the stabilizers, since the projector onto that subspace is a product of all the stabilizers

$$\Pi_C = \prod_{i=1}^m \frac{1}{2}(\mathbb{1} + s_i). \quad (2.49)$$

While the measurement outcomes are random, we may define them as our $+1$ eigenspace reference set. Assuming a noiseless environment and perfect control, any further measurement of s_i will give the same outcome λ_i as the one in our reference set. In other words, the stabilizer state is *stabilized* by the stabilizers. So one may choose without loss of generality to work with the representative of this subspace that has all $+1$ eigenvalues, i.e. $s_i |S\rangle = |S\rangle \forall s_i \in S$. So from now on, we will assume that a stabilizer state always has $+1$ eigenvalues.

2.2.4 Graph States

Graph states are a subset of stabilizer states that can be described by a graph $G = (V, E)$ composed of vertices or nodes V connected by edges E . The vertices represent qubits, while the edges represent CZ gates in the state preparation circuit. We illustrate an example of a graph state and its circuit preparation in figure 2.3b & 2.3c.

In the figure, we can also find the generators of the corresponding stabilizer state (figure 2.3a). In fact, the graph state's generators, called vertex stabilizers, are always of the following form [70]:

$$K_v = X_v \bigotimes_{u \in N(v)} Z_u, \quad (2.50)$$

where $N(v)$ refers to the neighbourhood of the vertex v , i.e. the set of vertices to which it is connected. So graph states represent a subset of stabilizer states with this specific structure of stabilizers.

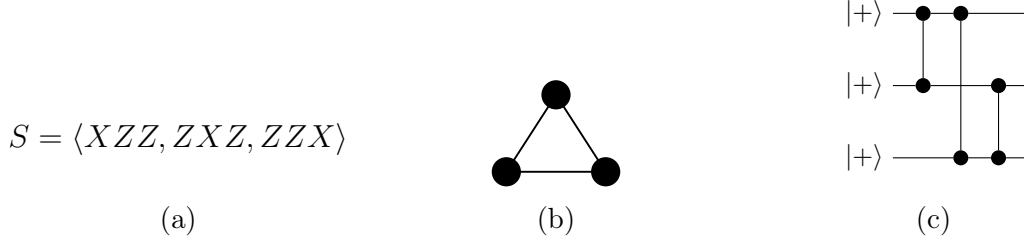


Figure 2.3: Stabilizer state represented by its stabilizer group (a), its graph state (b), and its encoding circuit (c). One may choose any ordering for the vertices, as the graph state is symmetric.

While the set of graph states does not span the whole set of stabilizer states by themselves, they can do so with the help of LC gates applied to the qubits [71].

$$\text{span}(S = \langle s_1, s_2, \dots, s_m \rangle) \Leftrightarrow \text{span}(G(V, E), LC). \quad (2.51)$$

The effect of the LC gates is to change the basis of the stabilizers through conjugation. For example, if we have the same graph state as in figure 2.3 with a H on the first qubit (you may choose any qubit in figure 2.3b because of symmetry), then this gate would send $S \mapsto \langle ZZZ, XXZ, XZX \rangle$ since it swaps X and Z .

So we know what the effect of these LC gates is on the stabilizers, but what about their effect on the graph itself? It turns out that LC gates and graph states can be redundant, i.e. two graph states and sets of LC gates map to a single stabilizer state. This redundancy can be expressed through the LC equivalence, a rule that corresponds LC gates to a graph operation called local graph complementations [71].

Definition 8 (Local complementation [72]). *Let $G = (V, E)$ be a graph. The local complementation about the vertex v is the operation that inverts the subgraph generated by the neighbourhood of v (but not including v itself). Formally, a local complementation about $v \in V$ sends G to the graph*

$$G \star v = (V, E \triangle \{\{b, c\} | \{b, v\}, \{c, v\} \in E \wedge b \neq c\}), \quad (2.52)$$

where $\triangle$ denotes the symmetric set difference, i.e. $A \triangle B$ contains all elements that are contained either in A or in B but not in both.

For example, we illustrate a local complementation and the action on its subgraph in figure 2.4. The local complementation toggles the edges of the subgraph: the edge between

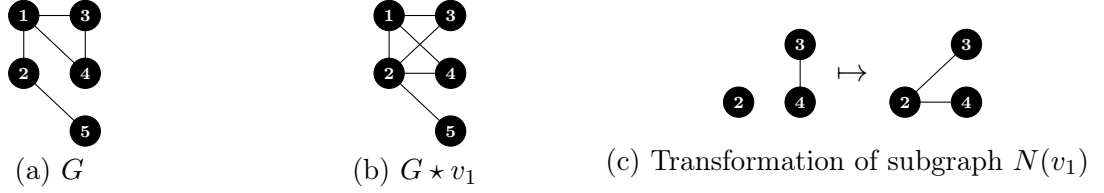


Figure 2.4: Example of a (a) 5-qubit graph state G onto which we apply a (b) local complementation about the first qubit v_1 . (c) The effect on the subgraph of the neighbouring vertices $N(v_1)$ is isolated from the rest of the graph.

vertices 3 and 4 in G was removed, but the edges between vertex 2 and the two other vertices were added because they weren't connected in G .

Now, the rule connecting this to local Clifford gates is that, given a graph state, a local complementation about a vertex v is equivalent to applying a $\sqrt{X}^{-1}$ on v and $\sqrt{Z}$ on all its neighbours.

Theorem 9 (LC Equivalence [71]). *Given a graph G and a vertex $v \in V(G)$ with a set of neighbouring vertices N ,*

$$R_x^{(v)}(-\pi/2) \bigotimes_{i \in N} R_z^{(i)}(\pi/2) |G\rangle = |G \star v\rangle.$$

For the purposes of this work, we do not distinguish between $R_x^{(v)}(\pm\pi/2)$, i.e between $\sqrt{X}$ and $(\sqrt{X})^{-1}$ (see subsection 2.2.2). Although the LC equivalence was first found for stabilizer codes (see subsection 2.2.3) [73, 74], it was only expressed into the LC equivalence discussed above for graph states.

Another useful rule is the equivalence between Hadamard gates on two vertices and pivoting around their common edge. Pivoting, or edge complementation, is a composition of three local complementations [75] about two vertices $x, y \in V$,

$$G \wedge xy = G \star x \star y \star x = G \star y \star x \star y. \quad (2.53)$$

However this series of operations can be performed in one go [75]:

$$\text{Given } G = \begin{array}{c} x \\ | \\ y \end{array} \begin{array}{c} X \\ | \\ Z \\ | \\ Y \end{array}, \quad G \wedge xy = \begin{array}{c} y \\ | \\ x \end{array} \begin{array}{c} X \\ | \\ Z \\ | \\ Y \end{array}, \quad (2.54)$$

where X is the set of vertices connected only to x , Y is the set of vertices connected only to y , and Z is the set of vertices connected to both x and y . On the right-hand side of the equation, we first point out that pivoting swaps the two vertices x and y . Also, we have toggled the edges between all three sets, but not those within a set. In general, there may be vertices connected to neither x nor y . Their edges remain untouched by edge complementation.

2.3 Stabilizer codes

As a means of introducing QEC codes, we first take a look at a classical 3-bit repetition code in subsection 2.3.1, and then to the 3-qubit repetition code in subsection 2.3.2. This will lead us to discuss the noise channels that create the errors we want to correct in. Finally we will define stabilizer codes in subsection 2.3.4, subsystem codes where we introduce gauge fixing 2.3.5, and dynamical codes 2.3.6. For subsections (2.3.1-2.3.2, 2.3.4), we take inspiration from Gottesman’s *An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation* [60].

2.3.1 3-bit Repetition Code

In a classical computer, information is stored in bits. An error presents itself as a bit flip with probability p . We may assume that the noise is independent and identically distributed, i.i.d. noise. To correct errors in this noise model, the 3-bit repetition strategy is to copy every bit $b \in \{0, 1\}$ 3 times. So with figure 2.1 in mind,³ the encoding is $\mathcal{E}(b) = bbb \in \{000, 111\}$. After the bits have gone through a noise channel $\mathcal{N}$, the logical information can be recovered $\mathcal{R} \circ \mathcal{N}([[[]]])$ by measuring all bits and taking the majority vote as the outcome bit b' . This decoding strategy bets that the errors are small and so most bits didn’t flip, meaning $b' = b$. The probability of the majority being wrong is $p_{\text{wrong}} = p^3 + p^2(1 - p)$. So, this scheme exhibits a quadratic suppression of error rates from $p \mapsto p^2$.

2.3.2 3-qubit Repetition Code

When trying to adapt the classical scheme for a quantum computer, a roadblock quickly appears: because of the no-cloning theorem, the mapping $|\psi\rangle \mapsto |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle$ is impossible to realize. Instead, the trick is to use entanglement. For $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $|\alpha|^2 + |\beta|^2 = 1$, the encoded state is

$$\mathcal{E}(|\psi\rangle \otimes |00\rangle) = \alpha|000\rangle + \beta|111\rangle = \alpha|0_\ell\rangle + \beta|1_\ell\rangle, \quad (2.55)$$

where we explicitly identify the logical basis $\{|0_\ell\rangle = |000\rangle, |1_\ell\rangle = |111\rangle\}$. We will name the encoded state $|\psi_\ell\rangle := \mathcal{E}(|\psi\rangle \otimes |00\rangle)$. The encoding can be realized through a CNOT

³Even though figure 2.1 is a quantum circuit, the classical version follows the same structure.

circuit entangling the three qubits. To perform a logical bit flip for this encoding, all three qubits must be flipped, so the logical operator of this code is $X_\ell = XXX$.

Now, to correct bit flip errors generated by the noise channel $\mathcal{N}$, it is not possible to measure the qubits in the computational basis as in the classical case because it will destroy the precious information we encoded. A smarter approach is needed: the parity of the qubits can be measured safely. In fact, the operators ZZI and IZZ preserve ψ_ℓ and our logical operator X_ℓ , so they are the code's *stabilizers*, $S = \langle ZZI, IZZ \rangle$. However, if an error $E = X$ occurs on one of the qubits, at least one of the stabilizers will have a negative eigenvalue. For example, if $E = XII$ we have:

$$ZZI \cdot XII |\psi_\ell\rangle = -XII \cdot ZZI |\psi_\ell\rangle = -XII |\psi_\ell\rangle, \quad (2.56)$$

where the sign changed because $\{X, Z\} = 0$. For each single-qubit X error, there is a unique combination of eigenvalues for each stabilizers that diagnoses the error, as in Table 2.1. These combinations of eigenvalues are called the *syndromes*. So the recovery map for this code $\mathcal{R}$ is to perform stabilizer measurements to get the syndromes and apply the diagnosed error back to the state, since the Pauli operators square to 1.

$S \backslash E$	XII	IXI	IIX
ZZI	-1	-1	1
IZZ	1	-1	-1

Table 2.1: Syndromes for the 3-qubit repetition code [3,1,3] for bit-flips errors.

The stabilizers and syndromes presented in Table 2.1 can only correct single bit flips. If one has two errors XXI , then only IZZ would have a negative eigenvalue, but that syndrome is already associated to the error IIX . So in that situation, the recovery map would "correct" the state ψ_ℓ by applying what it thinks to be the error, i.e. IIX . This would result in a *logical* error: $IIX \cdot XXI |\psi_\ell\rangle = X_\ell |\psi_\ell\rangle$. The code would *fail* with the same probabilities as in the 3-bit repetition code.

Another point in common with the code discussed above is that it doesn't detect phase-flip errors Z . This can easily be seen by the fact that all the stabilizers are made up of only I and Z , so they commute with phase-flip errors. A *quantum* error correction code is needed to correct *quantum* noise.

2.3.3 Quantum Error Models

To understand how to correct noise in a quantum computer, we first need to describe it. There are many possible models to describe noise in quantum systems. It can be i.i.d., coherent [76], biased [77], correlated [78], non-Markovian where the environment retains memory of past operations [79], qubit loss [80], leakage into higher energy states [81], and circuit-level [82]. The former considers noise on every element in your circuit, from state preparation to gates and to measurements, so it is a corner stone of noise estimation in QEC.

Different QEC codes perform better under different types of noise. For the purpose of this work, we will only go over the depolarizing channel, an i.i.d. noise model that can be decomposed in terms of Pauli operators. As such, it lends itself well to stabilizer QEC.

Because in stabilizer QEC, we measure stabilizers expressed as Pauli strings to find syndromes and correct errors, we effectively decompose and discretize errors into the Pauli basis. If the error is some unitary $e^{i\theta\vec{n}\cdot\vec{\sigma}/2} = \cos(\theta/2)\mathbb{1} + \sin(\theta/2)\vec{n}\cdot\vec{\sigma}$, measurement projects this operator onto the Pauli σ_i with probability $\sin(\theta/2)^2 n_i^2$. So the depolarizing channel is a useful way to describe noise in the context of stabilizer QEC.

The depolarizing channel brings the single-qubit input state ρ to the maximally mixed state $\mathbb{1}_{\mathbb{C}^2}$ with probability p ⁴: $\mathcal{N}_D(\rho) = (1-p)\rho + p\frac{1}{2^2}\mathbb{1}_{\mathbb{C}^2}$. It can also be represented using the Kraus representation [59]:

$$\mathcal{N}_D(\rho) = (1 - \frac{3}{4}\tilde{p})\rho + \frac{1}{4}\tilde{p} \sum_{\substack{a,b \in \{0,1\} \\ (a,b) \neq (0,0)}} X^a Z^b \rho (X^a Z^b)^*, \quad (2.57)$$

where $\tilde{p}$ is a probability, and we have the following Kraus operators: $A_{0,0} = \sqrt{1 - \frac{3}{4}p}\mathbb{1}_{\mathbb{C}^2}$ and $A_{a,b} = \frac{\sqrt{p}}{2}X^a Z^b$ for $(a,b) \neq (0,0)$. This noise model is a generalization to qubits of the classical error model for bits presented before, as the Kraus operators span through the Pauli operators.

The Kraus operators inform us on our goal to correct errors: every Pauli error happens with equal probability $\tilde{p}/4$. Since $Y = iXZ$, we only need to care about correcting X and Z errors on our qubits [60].

⁴Surprisingly, this transformation stays CPTP for $0 \leq p \leq \frac{4}{3}$, but here we only care about p as a probability.

2.3.4 Stabilizer Codes: Definition and Examples

An n -qubit stabilizer code [15, 65] is specified by an Abelian subgroup of the n -qubit Pauli group that does not contain $-I$, i.e. the stabilizer group $S = \{s_1, s_2, \dots, s_m\}$. The states that are in the codespace made up of the stabilizers are called codewords. Since the complex Euclidean space of the physical qubits holds the stabilizer and logical degrees of freedom, the code $k = n - m$ logical qubits. Each logical qubit is represented by sets of anticommuting logical operators $\{X_\ell, Z_\ell\}$ that commute with S , but are not contained within it. In other words the logical operators of the code are in $N(S)/S$, where $N(S) = \{P \in \mathcal{P}_n | PSP^* = S\}$ is the normalizer of the group. As such, stabilizers are sometimes called the centre of the code, since they commute with each other and all the logical operators. The logical operators map between different codewords, which are all in the codespace and thus protected from the noise.

The tolerance of error correction codes is described by the code distance $d = 2t + 1$, where t is the number of errors tolerated. The code distance comes from classical coding theory and is defined as the minimum weight of a logical Pauli operator.⁵ This is because it represents the number of Pauli operators needed to map to each other two elements of the codespace, the space spanned by the stabilizers. We refer to the minimum weight because a given representative of the logical operator is equivalent to another up to stabilizers. In the case of the 3-qubit repetition code, 3 bit flips (X_ℓ) mapped the logical $|0_\ell\rangle$ to $|1_\ell\rangle$, meaning $d = 3$. This matches with the previous conclusion that a only single bit-flip error can be corrected using that code.

Stabilizer codes are denoted using the shorthand $[[n, k, d]]$, where d is the code distance. In contrast, classical codes are denoted using simple brackets $[n, k, d]$. When $d = 2$, this means the code can only *detect* single-qubit errors, and a $d = 1$ code doesn't achieve any error correction.

The information about a stabilizer code can be compactly written in many ways. In the symplectic representation, the stabilizers can be put into a binary matrix $H = (S_X | S_Z)$, where the rows span the stabilizers and the entries are the $X(Z)$ support of each stabilizer on the right(left) side of the matrix [65]. This matrix is often called the parity check matrix, a term coming from classical coding theory. For example, we list the stabilizers of the 5-qubit code [83] and the stabilizer matrix in symplectic representation.

Another useful example to note is CSS codes [84, 85]. Their stabilizers are either made of Z Pauli strings or X Pauli strings. Then, the stabilizers are called Z -type or X -type.

⁵If there are multiple logical qubits, the minimum weight could be that of a joint logical operator $X_{\ell_i} X_{\ell_j}$.

$$\begin{array}{ccccc}
X & Z & Z & X & I \\
I & X & Z & Z & X \\
X & I & X & Z & Z \\
Z & X & I & X & Z
\end{array}
\quad
\begin{array}{c}
\left(\begin{array}{ccccc|ccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1
\end{array} \right)
\end{array}$$

(a)
(b)

Figure 2.5: Stabilizers (a) and stabilizer matrix in symplectic representation (b) of the 5-qubit code [83].

So, their parity check matrices are of a special form:

$$H = (S_X | S_Z) = \left(\begin{array}{c|c} 0 & H_Z \\ H_X & 0 \end{array} \right), \quad (2.58)$$

where $H_Z(H_X)$ are composed of the generators with $Z(X)$ support in symplectic representation. Actually, H_Z and H_X correspond to classical linear codes such that $H_Z H_X^\top = 0$. So if a Z error occurs, it will be detected and corrected by the X linear code described by H_X and similarly for X errors and thus Y errors. It may even be that these two codes are the same $H_Z = H_X$. These are self-dual CSS codes. Notable examples of CSS codes are surface codes [86, 87] and colour codes [88, 89].

Another label often accompanies these codes: they are quantum Low Density Parity Check (QLDPC) codes [90, 91]. This family of codes has the following property: the weight of the stabilizers and the number of stabilizers with support on a given qubit are bounded by a constant. It has received a lot of attention recently because this property makes it easier to measure the stabilizers. We will introduce the surface code in subsection 3.3.1 for lattice surgery, and the smallest 3D colour code will be used in magic state distillation in subsection 3.2.2.

As a final note on the diversity of types of stabilizer codes, we mention, as there exist graphs representing stabilizer *states*, there exist graphs representing stabilizer *codes*. These are called graph codes [92]. QLDPC codes can be seen as sparse graph codes [93].

In this work, we will often make use of stabilizer tableaux [94] to gather the stabilizers and logical operators of a code:

$$\left\langle s_1 \ s_2 \ \dots \ s_m \ \left\| \begin{array}{c} Z_{\ell_1} \ Z_{\ell_2} \ \dots \ Z_{\ell_k} \\ X_{\ell_1} \ X_{\ell_2} \ \dots \ X_{\ell_k} \end{array} \right. \right\rangle, \quad (2.59)$$

We point out the choice of representatives for the logical operators. Since they commute with the stabilizers, $s_i \cdot Z_{\ell_j}$ is an equivalent Pauli string, for example.

We close off this long, but important subsection with an overview of the stabilizer QEC process: logical information is encoded into a QEC code's codespace $|\psi_\ell\rangle$ using multiple physical qubits. It experiences noise $\mathcal{N}$ which may be modelled as applying Pauli operators to the physical qubits with a small probability p . Then, the stabilizers of the code are measured to obtain the syndromes. The circuits designed for this step are called syndrome extraction circuits. From the syndrome information, a possible error is diagnosed and can be corrected by applying some Pauli operators onto our physical qubits. Before measuring the stabilizers and applying a correction, there may be some quantum computation performed, which will be explored in Chapter 3. All this enables the protection of our logical qubits.

The process of finding the appropriate correction for given syndromes is called *decoding*. Different QEC codes call for different decoding strategies or algorithms. In general, this problem is NP-hard [95], implying that it is unlikely there exists a polynomial-time algorithm that can decode a QEC code. Even more so, finding an optimal decoding algorithm is $\#$ -P hard, so it is harder than decoding classical codes [96]. However, it is possible to choose our encoding carefully such that decoding is possible. Many schemes have been devised over the years, notably minimum weight perfect matching decoders for topological codes [82, 97] and tensor network-based maximum likelihood decoders [98].

2.3.5 Subsystem codes

In subsystem codes [56], the codespace of a stabilizer code is separated into two subsystems $C = A \otimes B$. In the first subspace A , the logical information $\rho_A \in \mathcal{B}(A)$ is stored, while the second subspace B remains a degree of freedom. More concretely, this means that some of the logical qubits in C become labelled as gauge qubits $\rho_B \in \mathcal{B}(B)$. If r qubits become gauge qubits, the code is denoted by the four variables $\llbracket n, k, r, d \rrbracket$.

Gauge is an overloaded word, so what does it mean here? Two codewords $\rho_a \otimes \rho_B$ and $\rho_a \otimes \rho'_B$ hold the same information even if $\rho_B \neq \rho'_B$, implying they are connected by a *gauge* transformation. This transformation can be defined by a set of gauge operators $g \in \mathcal{G}$ that map through conjugation the gauge states to each other $g\rho_B g^* = \rho'_B$. This behaviour might ring a bell: the gauge operators act on the gauge subspace like the logical operators do on the logical codewords. So, they are the gauge's old logical operators, and we use the notation $\{X_g, Z_g\}$ instead of g from now on.

After removing a few qubits from their lot, the logical operators $\{X_\ell, Z_\ell\}$ are now in $\mathcal{L} = N(S)/S/G$. The logical and gauge operators combined with the stabilizers can be

represented in a stabilizer tableau, where we separate the gauge operators

$$\left\langle \begin{array}{c|cc} s_1 & s_2 & \dots & s_m \\ \hline Z_{g_1} & Z_{g_2} & \dots & Z_{g_k} \\ X_{g_1} & X_{g_2} & \dots & X_{g_k} \end{array} \left\| \begin{array}{c|cc} Z_{\ell_1} & Z_{\ell_2} & \dots & Z_{\ell_k} \\ X_{\ell_1} & X_{\ell_2} & \dots & X_{\ell_k} \end{array} \right. \right\rangle, \quad (2.60)$$

where we have m stabilizers, k logical qubits and r gauge qubits, such that $n = m + r + k$. The code distance of a subsystem code is the minimum weight of a logical operator representative up to stabilizers and/or gauge operators [99].

The gauge qubits, since they hold no information, can be used as a fault-tolerant resource for different QEC protocols. For example, they can enable simpler recovery procedures [100] and the reduction of the weights of the stabilizer measurements, potentially making their implementations less noisy [101]. Gauge qubits can also be integrated into protocols to generate logical gates by fixing their states [28]. This *gauge fixing* technique is at the centre of the work presented in this manuscript [1]. In practice, gauge fixing may be realized by measuring the logical operators associated with a given state.

The gauge qubit's states can be fixed by measuring one of its gauge operators: X_g , Y_g or Z_g will project the qubit into their respective ± 1 eigenstates. Depending on the procedure at hand, there may be a correction necessary to get the correct eigenvalue, but this may always be done by classical bookkeeping. Effectively, by projecting into the eigenbasis of a gauge operator, gauge fixing promotes that operator to become part of the centre of the code, i.e. the stabilizers [102].

Gauge fixing reduces the dimensions of the subsystem B , and enlarges S , but the flow can go in the other direction too. To create a subsystem code from a stabilizer code, one of its stabilizers may be demoted by convention to become a gauge operator Z_g . There still needs another Pauli string to form an anticommuting set. The destabilizers [94] lends itself perfectly for this purpose. Every destabilizer is a Pauli string that commutes with all but one stabilizer. With the stabilizers, they form a complete basis for $\mathcal{P}_n$. Coming back to gauge qubits, to create more of them, one can demote a stabilizer to become Z_g and its destabilizer to become X_g .

We close off this subsection with a gauge fixing example, as this technique will be a central part of the results. We start simply with the smallest error detecting code,⁶ the $[[4, 2, 2]]$ code [103, 104]:

$$C^{(1)} = \left\langle \begin{array}{c} s_1^{(1)} = XXXX \quad s_2^{(1)} = ZZZZ \\ \hline Z_{\ell_1}^{(1)} = XIII \quad Z_{\ell_2}^{(1)} = IIXX \\ X_{\ell_1}^{(1)} = ZZII \quad X_{\ell_2}^{(1)} = IZZI \end{array} \right\rangle, \quad (2.61)$$

⁶This code can only detect errors because it has a code distance of 2.

where we added the top index to differentiate the operators from successive ones below. It might appear to be counter-intuitive for the $Z_\ell^{(1)}(X_\ell^{(1)})$ representative to be a Pauli $X(Z)$ string. This is still a valid choice, since the operators respect the necessary anticommuting relations. We made this choice because it will match up better with the logical operators found looking at the ZX-calculus representation of stabilizer codes (see section 2.5).

From this stabilizer code, we may demote $s_2^{(1)}$ to become a gauge operator Z_g . We choose its corresponding destabilizer to be $X_g = IXII$. It might appear bad for the code distance that this operator has weight 1, but this is a gauge operator holding no information. Instead, one can look if the gauge operator composed with any product of logical operators has a lower weight than originally. Here, one can check this is not the case, so this is a safe choice.

Another problem one might have with $X_g = IXII$, is that it doesn't commute with all the operators. In particular, $X_{\ell_1}^{(1)}$ and $X_{\ell_2}^{(1)}$ representatives shown above don't commute with it. However, there always exists a representative that will commute, namely $Z_g \cdot X_{\ell_i}^{(1)}$: since both Z_g and $X_{\ell_i}^{(1)}$ anticommute with X_g , their product commutes. We point out that this choice is allowed because $Z_g = s_2^{(1)}$, so $Z_g \cdot X_{\ell_i}^{(1)}$ was always one of the representatives of the X logical operators of $C^{(1)}$.

The subsystem code after demoting $s_2^{(1)} \mapsto Z_g$ is the following:

$$C^{(2)} = \left\langle s_1^{(2)} = XXXX \left| \begin{array}{c} Z_{g_1} = ZZZZ \\ X_{g_1} = IXII \end{array} \right\| \begin{array}{c} Z_{\ell_1}^{(2)} = XII X \quad Z_{\ell_2}^{(2)} = IIX X \\ X_{\ell_1}^{(2)} = IIZZ \quad X_{\ell_2}^{(2)} = ZIIZ \end{array} \right\rangle, \quad (2.62)$$

where the two $X_{\ell_i}^{(2)}$ have been updated to commute with the destabilizer X_g . Next, we gauge fix our new degree of freedom. We will choose to perform a Y gauge fixing,⁷ $Y_g = Z_g \cdot X_g = ZYZZ$. By measuring this operator, we get:

$$C^{(3)} = \left\langle s_1^{(3)} = XXXX \quad s_2^{(3)} = ZYZZ \left\| \begin{array}{c} Z_{\ell_1}^{(3)} = XII X \quad Z_{\ell_2}^{(3)} = IIX X \\ X_{\ell_1}^{(3)} = IIZZ \quad X_{\ell_2}^{(3)} = ZIIZ \end{array} \right. \right\rangle. \quad (2.63)$$

By comparison with our original stabilizer tableau (2.61), one may notice that the space generated by the X logical operators and the stabilizers are different for the two codes. For example, the two codes' $X_{\ell_1}^{(1)}$ representatives $X_{\ell_1}^{(1)} \in \{ZZII, YYZZ, IIZZ, XXYX\}$ and $X_{\ell_1}^{(3)} \in \{IIZZ, ZZYY, ZYII, YZXX\}$ are different except for a single representative

⁷This exact gauge fixing from the $[[4, 2, 2]]$ to the new encoding will be used in section 4.2.3

$IIZZ$ which allowed the bridge between the codes. In fact, this is exactly the $X_{\ell_1}^{(2)}$! So, we conclude that the two codespaces are different, but have some overlap.

Not any two codes have this overlap, but any codes with a common *parent subsystem* do. We illustrate an example with a single gauge qubit in figure 2.6. If a parent subsystem were to have more than one gauge qubit, then there would be more possible combinations of gauge fixings, which would lead to different codes. As will become clear in section 4.2, it may be that two codes C_1 & C_2 from the figure are the same.

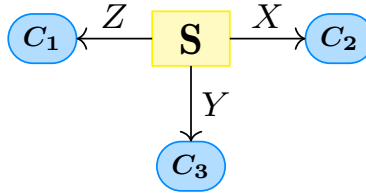


Figure 2.6: Parent subsystem code S with a single gauge qubit, so it bridges three child codes C_1 , C_2 & C_3 for each gauge fixing.

To summarize, subsystem codes are stabilizer codes that have gauge qubits in which no information is stored. They may be used to connect different stabilizer codes through gauge fixing.⁸ Information in logical qubits may be transferred from one encoding to another using this trick.

It is worth mentioning that subsystem codes may not be exclusively used as bridges between stabilizer codes. In fact, they were first introduced as a way to facilitate decoding with the Bacon-Shor code [100], to better absorb errors [105] or to reduce the weight of the measurement performed to collect syndromes [101].

2.3.6 Dynamical Codes

Stabilizer codes presented so far have been static: the logical information is encoded into the codespace associated with a given QEC code. Now that we understand how to switch between codes, one may wonder: why stay confined within a single code? Different codes have different properties, so it could be useful to exploit complementary ones, as will be explored in section 3.5. Also, as mentioned at the end of the last subsection, playing with

⁸This may also be called code switching, but we refrain from discussing it in this section because it was originally introduced in the context of logical gates. As such, we will discuss them further in section 3.5.

different measurement schedules than the direct syndrome measurements may allow you to facilitate state recovery or to make the measurements needed to get syndromes easier.

Dynamical codes represent this new paradigm where we let the code evolve in time based on a measurement schedule. At each time step, there is an instantaneous stabilizer code generated by the stabilizers and the gauge operators that were just measured. This scheme, proposed by Hastings and Haah [33], attracted a lot of attention recently. In their work, the authors exploit a parent subsystem code and the logical information flows between three child codes. Interestingly, the parent subsystem can't hold any logical information, meaning that if treated statically, any of the three child stabilizer codes also holds no logical qubits. However, a well-crafted gauge measurement schedule between the three stabilizer codes allows us to create 2 logical qubits.

To understand this measurement schedule, we first point to the code construction illustrated in figure 2.7a. The qubits are on a hexagonal lattice, which is 3-colourable. If the lattice has n plaquettes, or face, then there are $2n$ physical qubits and $3n$ edges. Each edge represents a 2-qubit gauge operator. At a given vertex, three edges meet and they all represent anticommuting 2-qubit Pauli operators. Each plaquette represents a stabilizer by taking the product of the operators on its border. With two stabilizers that span the length of the periodic space, the stabilizer group is of dimension $n + 1$. Now we can count the number of gauge qubits as the number of independent anticommuting sets of 2-qubit Pauli operators $(3n - 1)/2^9$ without the centre stabilizers, so $(3n - 1 - (2n - 1))/2 = n - 1$.

However, a measurement schedule can be created where all the gauge operators of colour c are measured at once in a periodic loop: red, green, blue, red, green blue, etc. This periodic sequence is illustrated in figure 2.7b, where the logical operators get updated up to a stabilizer of the previous code as in the gauge fixing example of subsection 2.3.5. We mention that the actual period of this code involves 6 measurements and not 3, as the figure suggests, because each round generates a -1 sign. However, we work in the symplectic representation, so this phase difference is ignored.

After the first period of measurements, we have enough information such that the instantaneous stabilizer code may be defined. Since the subset of c gauge operators has disjoint support on the qubits, the operators all commute with each other. Their measurements promote them to become stabilizers. This code has n plaquettes and n c gauge operators, but the product of all plaquettes and the product of all c gauge operators with c coloured plaquettes are both 1. So the stabilizer group is of dimension $2n - 2$. This means that there are $k = n - m - g = 2n - (2n - 2) = 2$ logical qubits.

⁹There is a redundancy represented by the product of edge operators. The number is divided by 2 is because there are 2 operators in each set.

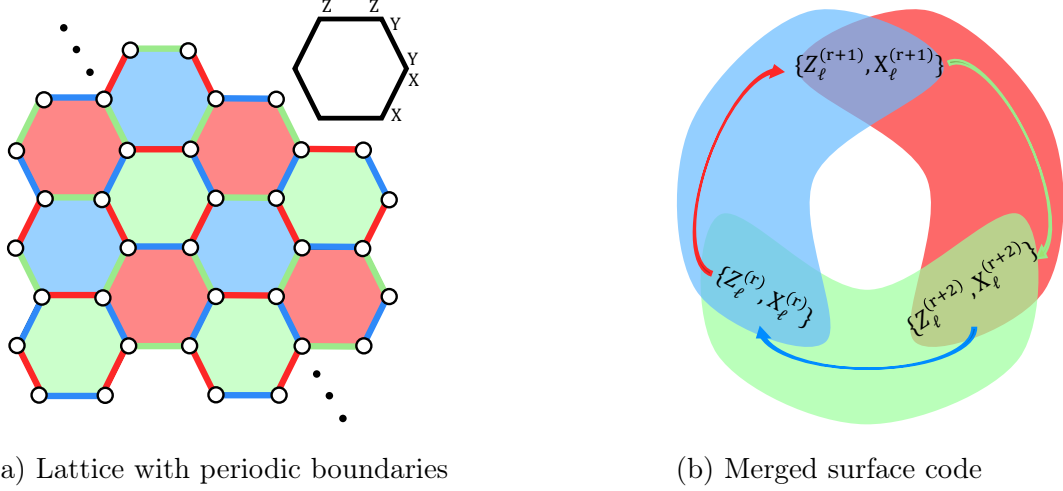


Figure 2.7: Honeycomb code [33]. (a) Physical qubits are on the vertices of the hexagonal lattice. The three types of plaquette stabilizers are coloured red, green and blue. Edges that connect the same coloured plaquettes are themselves the same colour. The plaquette on the top left illustrates the types of 2-qubit gauge operators along each direction of the lattice. (b) Periodic measurement sequence with time steps r . The regions represent the codespace of each instantaneous stabilizer code. The coloured arrows represent the measurement of gauge operators of the same colour. One of two sets of logical operators is illustrated after each step r .

As one can see by comparing with the parent subsystem code, the periodic measurement sequence makes all the difference. The authors give a name for periodic dynamical code: Floquet codes.¹⁰ However, the evolution need not have one parent subsystem code as Davydova et al. found [34]. In fact, they construct such a dynamical code. So the authors define dynamical tree codes as a broader type of dynamical code that is aperiodic. They don't define dynamical codes per se. Instead, another paper by Davydova et al. [38] introduces the concept of dynamical codes and dynamical automorphisms as one: dynamical automorphism codes.

In this manuscript, we make the distinction between dynamical codes and dynamical automorphisms. We study specifically the logic where the codespace eventually comes back to itself up to a logical gate, while dynamical automorphism codes may be aperiodic. This concept will be further explained in section 3.5.

¹⁰If there are only two codes in the loop, then the procedure is called code switching. Because it is usually used to generate logic, it will be addressed later in section 3.5.

Davydova et al. [38] give a formalism for this based on anyon, which is not covered in this manuscript. Such a formalism builds a subset of stabilizer codes called topological codes [106]. Instead of using this formalism, we will build on the subsystem and gauge fixing formalism introduced in the previous section. One of this work's outlooks relevant in the context of this manuscript is to broaden the set of measurement sequences, which may lead to dynamical codes and automorphisms, and to better understand the QEC constraints on those measurements.

2.4 Introduction to the ZX-calculus

The ZX-calculus, originally introduced by Coecke and Duncan [40], is a tool used to describe quantum circuits through a graphical representation. We refer the reader to [41] for a detailed exposition, as only information necessary for the manipulation of graph-like ZX-diagrams will be presented here.

In the ZX-calculus, vertices and edges are called spiders and wires. There are two different colours of spiders: green Z spiders and red X spiders. They can have as many input and output wires as one might want:

$$m \text{ --- } \textcircled{\alpha} \text{ --- } n = |0\rangle^{\otimes n} \langle 0|^{\otimes m} + e^{i\alpha} |1\rangle^{\otimes n} \langle 1|^{\otimes m}, \quad (2.64)$$

$$m \text{ --- } \textcircled{\alpha} \text{ --- } n = |+\rangle^{\otimes n} \langle +|^{\otimes m} + e^{i\alpha} |-\rangle^{\otimes n} \langle -|^{\otimes m}. \quad (2.65)$$

Notice how the number of input and output wires in a spider dictates the number of tensor products that form the input and output states of a linear map. In other words, spiders are tensors with varying dimensions, and connecting spiders together through wires corresponds to tensor contractions. So ZX-diagrams are tensor networks.

More concretely, if we have $n = 1$ and $m = 0$, then these spiders correspond to the superposition of states in the computational or the Hadamard basis with a relative phase α . For example, the $|0\rangle$ state can be represented as $\textcircled{\alpha}$. These one-legged spiders are also used to represent projective measurements:

$$\textcircled{\alpha} = \langle -|0\rangle = \frac{1}{\sqrt{2}}. \quad (2.66)$$

From the equation above, it is clear that scalar quantities are represented by spiders with no wires. In this manuscript, we will drop all scalar factors and normalization will be implied, as is standard in the ZX-calculus.

If instead we have a spider with $n = m = 1$ and $\alpha = \pi$, then they represent the familiar X and Z Pauli matrices:

$$\textcircled{\pi} = |0\rangle \langle 0| - |1\rangle \langle 1| = Z \quad \& \quad \textcircled{\pi} = |+\rangle \langle +| - |-\rangle \langle -| = X. \quad (2.67)$$

Spiders with different phases give a description of other known gates, notably

$$\textcircled{\frac{\pi}{2}} = |0\rangle \langle 0| + i |1\rangle \langle 1| = S = \sqrt{Z} \quad \& \quad \textcircled{\frac{\pi}{2}} = |+\rangle \langle +| + i |-\rangle \langle -| = \sqrt{X}. \quad (2.68)$$

Composing these gates, we can build a Hadamard gate:

$$H = \sqrt{Z} \sqrt{X} \sqrt{Z} = \textcircled{\frac{\pi}{2}} \textcircled{\frac{\pi}{2}} \textcircled{\frac{\pi}{2}} \quad \text{or} \quad H = \sqrt{X} \sqrt{Z} \sqrt{X} = \textcircled{\frac{\pi}{2}} \textcircled{\frac{\pi}{2}} \textcircled{\frac{\pi}{2}}. \quad (2.69)$$

This gate deserves its own symbol, as it is frequently used: $H = \text{---}\blacksquare\text{---}$. Using the phase gate and the Hadamard gate, one can now build all 1-qubit Clifford circuits using spiders with phases $\alpha \in \frac{\pi}{2}\mathbb{Z}$. Circuits made up of only such spiders are in the Clifford fragment. We work within this set of ZX-diagrams for the entirety of the results in Section 4. To move on to circuits supported on more qubits, we introduce the rules of the ZX-calculus in figure 2.8.

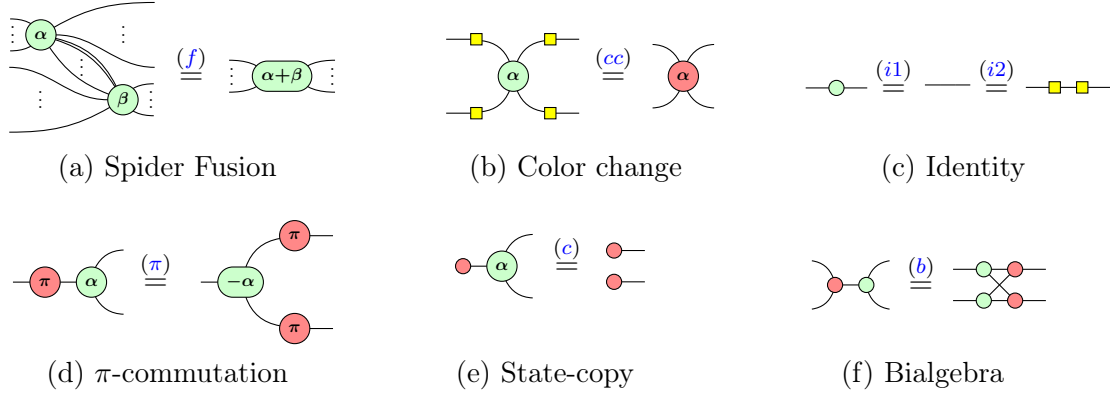


Figure 2.8: Rules of the ZX-calculus

Most of the ZX-calculus rules translate the familiar quantum mechanics into graphs: the spider fusion rule (f) is equivalent to $Z = \sqrt{Z} \cdot \sqrt{Z}$; the colour change rule (cc) generalizes to tensor networks the action of the Hadamard gate on the Paulis, $HX = ZH$; the identity rules ($i1$) & ($i2$) trivially come from $I = |0\rangle\langle 0| + |1\rangle\langle 1|$ and $H^2 = I$; the π -commutation (π) rule reflects the Pauli property $XZ = -ZX$; and the state-copy rule (c) is associated with the action of measurements of a Bell pair, where two entangled qubits become uncorrelated after a measurement in the computational basis, for example.

These rules are sufficient to manipulate any ZX-diagram. For example, we introduce the CNOT circuit. One can recognize the propagation of Z and X errors acting on the control qubit:

$$\begin{array}{c} \bullet \\ \oplus \end{array} = \begin{array}{c} \bullet \\ \oplus \end{array} : \begin{array}{c} \pi \\ \oplus \end{array} \begin{array}{c} \pi \\ \oplus \end{array} \stackrel{(f)}{=} \begin{array}{c} \pi \\ \oplus \end{array} \begin{array}{c} \pi \\ \oplus \end{array} \quad \& \quad \begin{array}{c} \pi \\ \oplus \end{array} \begin{array}{c} \pi \\ \oplus \end{array} \stackrel{(\pi)}{=} \begin{array}{c} \pi \\ \oplus \end{array} \begin{array}{c} \pi \\ \oplus \end{array} \stackrel{(f)}{=} \begin{array}{c} \pi \\ \oplus \end{array} \begin{array}{c} \pi \\ \oplus \end{array} . \quad (2.70)$$

From this ZX-diagram, it is straightforward to find the one for CZs:

$$\begin{array}{c} \bullet \\ \bullet \end{array} = \begin{array}{c} \bullet \\ \bullet \end{array} \stackrel{(cc)}{=} \begin{array}{c} \bullet \\ \bullet \end{array} \stackrel{(i2)}{=} \begin{array}{c} \bullet \\ \bullet \end{array} = \begin{array}{c} \bullet \\ \bullet \end{array} . \quad (2.71)$$

In the last equality above, we have introduced a new notation, Hadamard edges [107] $\text{---}\text{---}\text{---} := \text{---}\text{---}\text{---}$. Unlike normal black wires, the spider fusion property does not work on these wires due to the implicit Hadamard gate. This notation allows for the reduction of ZX-diagrams to graph-like diagrams. Such diagrams are made up of Z spiders and Hadamard edges that keep the Z spiders from fusing, thus retaining a graph-like shape. We follow the definition of graph-like ZX-diagrams in [107].

Definition 10. *A ZX-diagram is graph-like when:*

1. *All spiders are Z-spiders.*
2. *Z-spiders are only connected via Hadamard edges.*
3. *There are no parallel Hadamard edges or self-loops.*
4. *Every input or output is connected to a Z-spider and every Z-spider is connected to at most one input or output.*

Graph-like ZX-diagrams come with some extra rules. The action of local Cliffords on an open wire is equivalent to local complementations, defined in subsection 2.2.4. This was first found in the context of graph states by Van den Nest et al. [71] (see Definition 8 and theorem 9 subsection 2.2.4), and was introduced to the ZX-calculus by Duncan and Perdrix [108].

We illustrate the LC equivalence from theorem 9 in the ZX-calculus with the following example:

$$\text{Given } G_{ZX} = \begin{array}{c} 1 \text{---} \text{---} 3 \\ 2 \text{---} \text{---} 4 \\ \quad \quad \quad \text{---} 5 \end{array} , \quad \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array} = \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array} . \quad (2.72)$$

On the left-hand side of the equation, we applied $\sqrt{X} = -\text{---}$ on vertex 1, and $\sqrt{Z} = -\text{---}$ on all its neighbours, vertices 2, 3 and 4. On the right-hand side, we applied a local complementation about vertex 1.

This local complementation rule will often be used in a context where one is applying a $\sqrt{X}$ gate on a given vertex, without the $\sqrt{Z}$ s. Then, we may adjust example (2.72) by moving these gates to the left-hand side of the equation instead, using the convention $(\sqrt{Z})^2 = I$.

An advantage of expressing graph states as ZX-diagrams becomes apparent: since the graph-like ZX-diagrams graphs have input and output wires, it becomes easy to integrate the local Clifford gates into the same object. This is especially useful for the LC equivalence, in comparison with the equivalent equations in figure 2.4.

The other graph state rule called pivoting, introduced in subsection 2.2.4, can also be integrated into the ZX-calculus [109]. We illustrate using ZX-diagrams the same example of the equivalence between $H_x \otimes H_y$ and $G_{ZX} \wedge xy$ as in equation (2.54), keeping in mind we do not keep track of the Pauli operators in our conventions.

$$\text{Given } G_{ZX} = \begin{array}{c} 1 \text{---} \text{---} 3 \\ \text{---} \text{---} 4 \\ 2 \text{---} \text{---} 5 \end{array}, \quad \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array} = \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array}. \quad (2.73)$$

Here, because the order of vertices is used as a label, we do not permute vertices 1 & 2 like in equation (2.54), but rather we exchange their connectivity. Again, this rule will be used to move H gates around the connected vertices of the graph-like ZX-diagrams.

2.5 Stabilizer Codes in the ZX-Calculus

Now, we assemble all the background information to represent stabilizer codes in the ZX-calculus, i.e. as ZX-encoding graphs. The latter will be defined first in subsection 2.5.1. Next, subsection 2.5.2 discusses the bijection called map-state duality between ZX-encoding graphs and stabilizer states, which correspond to graph states and local Clifford gates. This allows us to use the LC equivalence on ZX-encoding graphs and find the stabilizer code they represent in subsection 2.5.3. Then, in subsection 2.5.4, we explain how to construct the ZX-encoding graph given a set of stabilizers and logical operators. Finally, we discuss the existence of a normal form for ZX-encoding graphs and how to obtain it in subsection 2.5.5.

2.5.1 ZX-Encoding Graphs

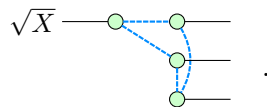
In the ZX-calculus, stabilizer codes are represented by their encoder, i.e. by Clifford isometries called ZX-encoding graphs. Starting from the encoding circuit of a stabilizer code, its ZX-encoding graph may be constructed by translating each gate in the ZX-calculus to find the equivalent ZX-diagram, and simplifying the ZX-diagram into a graph-like ZX-diagram with some local Clifford operators on input or output wires [107, theorem 5.4]. It is also possible to build the graph from the stabilizers and the logical operators, as will be explained in subsection 2.5.4.

To define ZX-encoding graphs, we take inspiration from previous work [42, 44, 45].

Definition 11. *A ZX-encoding graph is a bipartite graph-like ZX-diagram, where the bi-adjacency matrix is full rank and 1-qubit local Clifford gates can be added to the input or output wires.*

We highlight that this includes subsystem codes: a logical input may simply be labelled as a gauge input. From now on, we will choose to only consider a single gauge qubit at a time for simplicity. In any case, the gauge qubits are only measured one at a time in dynamical codes and dynamical automorphisms.

Here is an example of a ZX-encoding graph with three physical qubits and a single logical qubit:


(2.74)

matrix is $B = (110)$. This matches up with the connectivity in the first layer of equation (2.75).

In the definition of ZX-encoding graphs 11, there was a condition that the biadjacency matrix of the ZX-encoding graph must be full rank, meaning every row in B must be linearly independent. This comes from the fact that the ZX-encoding graph represents an isometry mapping the logical qubits to the physical qubits. Every set of logical operators must remain distinct, so the rows must be linearly independent.

Proposition 12 (Biadjacency matrix: Full rank [1]). *A graph-like ZX-diagram defines an isometry if and only if its biadjacency matrix between its input and output vertices is full rank. In particular, for any realization of a Clifford encoding isometry as a graph-like ZX-encoding graph, every input vertex is connected to at least one output node.*

In Ref. [1, Appendix C2], proof of this condition is given. We do not present it here even if it is simple because it relies on concepts not introduced in this manuscript's already long background chapter.

In our example, this is trivially true. So we add another logical to illustrate this rule for ZX-encoding graphs:

$$\begin{array}{c} \sqrt{X} \text{---} \bullet \text{---} \bullet \text{---} \\ \text{---} \bullet \text{---} \bullet \text{---} \end{array} \rightarrow B = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}. \quad (2.78)$$

Finally, we look at the LC gate on the logical qubit in the ZX-encoding graph in (2.74). In general, as stated in Definition 11, every input or output may have an LC gate. Since the LC gates sandwich a graph-like ZX-diagram, we may use the LC equivalence rules (2.72) and (2.73) to make pass $\sqrt{X}$ and H through the ZX-encoding graph:

$$\begin{array}{c} \sqrt{X} \text{---} \bullet \text{---} \bullet \text{---} \\ \text{---} \bullet \text{---} \bullet \text{---} \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \bullet \text{---} \sqrt{Z} \\ \text{---} \bullet \text{---} \bullet \text{---} \sqrt{Z} \end{array}. \quad (2.79)$$

The LC equivalence makes use of the correspondence between ZX-encoding graphs, graph-like ZX-diagrams, and thus graph states. This link will be further explored in the following subsection.

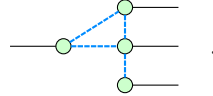
We note that there is also a close relation between ZX-encoding graphs and graph codes [45], but this will not be explored in this manuscript. Instead, we will often compare the ZX-calculus-based formalism to the stabilizer tableaux formalism (see subsection 2.5.3, 2.5.4 & 4.1.4).

2.5.2 Map-State Duality

Via the map-state duality, it is possible to associate a given ZX-encoding graph with a stabilizer state [45], which corresponds to a graph state (up to local Clifford equivalence) [71]. This duality enables us to use results from graph states in the context of stabilizer codes. Notably, it is possible to translate local Clifford operators into local complementations as in section 2.4. Also, the vertex stabilizers of a graph state can be used to find the stabilizer code represented by a given ZX-encoding graph.

Before delving into the applications, we take a moment to define the map-state duality. The ZX-encoding graph represents a CPTP map $\mathcal{E}$ because it maps the input logical qubits onto the physical qubits. So its Choi representation $J(\mathcal{E})$ is unique. Also, since $\mathcal{E}$ is TP, $\text{Tr}(J(\mathcal{E})) = 1$, so $J(\mathcal{E})$ is a density matrix [58]. As a consequence, we conclude that there is a bijection between the linear map represented by the ZX-encoding graph and the quantum state represented by $J(\mathcal{E})$.

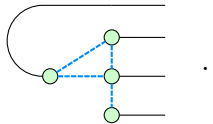
We aim to find the Choi representation, in the ZX-calculus, of the following ZX-encoding graph:


(2.80)

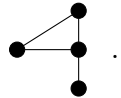
Referring to section 2.1, we have $J(\mathcal{E}) = (I \otimes \mathcal{E}) |\Phi^+\rangle \langle \Phi^+|$, where $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. In the ZX-calculus, we then have

$$|\Phi^+\rangle = \text{C}_{(i1)} = \text{C} . \quad (2.81)$$

When applying the map onto the second qubit, the logical input from the ZX-encoding graph in equation 2.80 will be transformed into an output:


(2.82)

This diagram represents a state now, since there are only output wires. Because the ZX-encoding graphs are graph-like ZX-diagrams, it is possible to make a direct association with the graph state formalism:


(2.83)

As a reminder, the edges in equations 2.82 & 2.83 both represent CZ gates, making the correspondence between the ZX-encoding graphs and the graph states trivial.

While associating a ZX-encoding graph with a graph state is straightforward, associating a ZX-encoding graph with a given graph state may be less obvious if the input and output vertices are not known. Then, many different choices of input vertices lead to different linear maps, but some choices may not lead to valid ZX-encoding maps. More precisely, the linear map must be an isometry, meaning the connectivity of the logical input vertices must be linearly independent. However, the physical output vertices do not have such restrictions.

2.5.3 Finding the Stabilizer Tableau of a ZX-Encoding Graph

Knowing ZX-encoding graphs may be mapped to graph states, we now use the vertex stabilizers introduced in subsection 2.2.4 to find the stabilizer code represented by the ZX-diagram. To do so, we adopt a notation to keep track of the support of the physical, logical and gauge qubits in the vertex stabilizers:

$$v_i = (O_1 \dots O_i \dots O_n | O_{l_1} \dots O_{l_k} O_g). \quad (2.84)$$

This separation between the support of the physical qubits from the support of the logical and gauge qubits is useful because the logical and gauge operators only have support on the physical qubits. So, we will also denote $v_i|_{\text{physical}} = O_1 \dots O_i \dots O_n$ as the restricted the support of the vertex stabilizer to the physical qubits.

As a first example, we take a look at a simple ZX-encoding graph and its associated graph state:

$$\ell \text{ --- } \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \end{array} \begin{array}{c} 1 \\ 2 \end{array} \mapsto \begin{array}{c} \ell \quad 1 \quad 2 \\ \text{---} \text{---} \text{---} \end{array} = \begin{array}{c} \bullet \quad \bullet \quad \bullet \end{array} . \quad (2.85)$$

This graph state has three vertex stabilizers:

$$\{v_1 = (XZ|Z), v_2 = (ZX|I), v_\ell = (ZI|X)\}. \quad (2.86)$$

These operators also stabilize the corresponding ZX-diagram. Here we illustrate the action of v_2 :

$$\begin{array}{c} \pi \quad \pi \quad \pi \\ \text{---} \text{---} \text{---} \end{array} \xrightarrow{(f)+(cc)} \begin{array}{c} \pi \quad \pi \quad \pi \\ \text{---} \text{---} \text{---} \end{array} \xrightarrow{(\pi)+(f)} \begin{array}{c} \pi \quad \pi \quad \pi \\ \text{---} \text{---} \text{---} \end{array} \xrightarrow{(i1)} \begin{array}{c} \pi \quad \pi \quad \pi \\ \text{---} \text{---} \text{---} \end{array} . \quad (2.87)$$

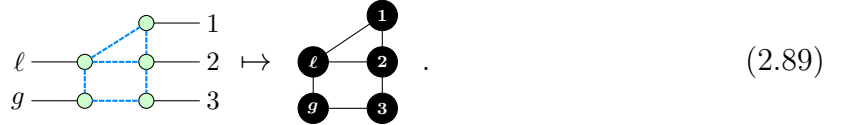
The vertex stabilizers can be used to find the stabilizer tableau associated with this ZX-encoding graph by composing them. Since the stabilizers have no effect on the logical qubits, we must find the combination of vertex stabilizers that give a logical identity when composed. Here, the only non-trivial combination is $S_1 = v_2|_{\text{physical}}$. It is expected to have

a single stabilizer, since $m = n - k = 2 - 1 = 1$. To find the logical operators, we must find a combination of vertex stabilizers that compose to X and Z on the logical output. Again, here the combinations are straightforward: $X_\ell = v_\ell|_{\text{physical}}$ & $Z_\ell = v_1|_{\text{physical}}$. So we find the following stabilizer tableau:

$$\left\langle ZX \left| \begin{array}{l} Z_\ell = XZ \\ X_\ell = ZI \end{array} \right. \right\rangle. \quad (2.88)$$

It is easy to check that the stabilizer and operators have the correct commutation relations.

Next, we tackle an example with more than a single input vertex. We go back to our subsystem code:



The vertex stabilizers of the graph state above are the following:

$$\{v_1 = (XZI|ZI), v_2 = (ZXZ|ZI), v_3 = (IZX|IZ), \\ v_\ell = (ZZI|XZ), v_g = (IIZ|ZX)\}. \quad (2.90)$$

First, we find the single stabilizer. Here, there are no vertex stabilizers with I on both inputs, but we can compose two to cancel the unwanted support: $S_1 = (v_1 \cdot v_2)|_{\text{physical}}$.

Next, the Z operators can be taken directly from vertex stabilizers $Z_\ell = v_1|_{\text{physical}}$ & $Z_g = v_3|_{\text{physical}}$. Equivalently, we could have chosen $Z_\ell = v_2|_{\text{physical}}$, and the two options are mapped to one another via our stabilizer, confirming their equivalence.

Finally, we get to the X operators. Notice how v_ℓ and v_g do not give directly the X_ℓ and X_g , unlike the example before. This is because the logical and gauge inputs are connected, so there is an extra Z on the gauge input entry for v_ℓ and vice versa. In a similar way we found S_1 , we may use the same vertex stabilizer used to find the Z operators in order to remove the unwanted Z : $X_\ell = (v_\ell \cdot v_3)|_{\text{physical}}$ & $X_g = (v_g \cdot v_1)|_{\text{physical}}$.

We point out that it is always possible to find a vertex stabilizer that cancels unwanted operators on some input vertices. In fact, the gauge and logical connectivity must be different in order for the ZX-encoding graph to be a valid isometry, i.e. its biadjacency matrix must be full rank, as seen in subsection 2.5.1. This difference in connectivity implies there will always be a combination of vertex stabilizers that will have support on a single input.

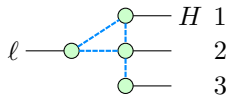
To conclude, we have all the information necessary to construct the stabilizer tableau associated with the ZX-encoding graph of equation (2.89):

$$\left\langle YYZ \left| \left| \begin{array}{l} Z_\ell = XZI \\ X_\ell = ZIX \end{array} \right| \begin{array}{l} Z_g = IZX \\ X_g = XZZ \end{array} \right. \right\rangle. \quad (2.91)$$

2.5.4 Constructing the ZX-Encoding Graph of a Stabilizer Code

While it is straightforward to find the stabilizer generators and logical operators of the stabilizer code represented by a given ZX-encoding graph, the reverse may not be obvious. In fact, one can find the stabilizers, the logical operators, and gauge operators by first transcribing them into a row matrix in the symplectic representation $M = (G_X|G_Z)$, and then performing a Gaussian elimination along with additional column operations (based on local Cliffords and qubit permutations). Van den Nest et al. [71] proved that the resulting square matrix on the right-hand side must be symmetric, and thus correspond to the adjacency matrix of a graph. The result was found to convert stabilizer states into a graph state, but we may use the map-state duality to apply it to stabilizer codes as Clifford isometries.

Every row in M represents a stabilizer, a logical operator or a gauge operator, and G_X (G_Z) is their X (Z) support. As such, both G_X and G_Z are square matrices with entries in $\mathbb{F}_2$. To illustrate this, we turn to our trusted 3-qubit code example. To create a telling example, we will start with a particular ZX-encoding graph in its LC orbit, from which we extract the vertex stabilizers, as seen in the background section of the Letter:



$\ell \rightarrow v = \{v_1 = (Z|ZZI), v_2 = (Z|XXZ),$
 $v_3 = (I|IZX), v_l = (X|XZI)\},$

(2.92)

where the support on the vertex stabilizers is ordered as follows $v_i = (\ell|123)$. From these, we derive the stabilizers $\langle YYZ, IXZ \rangle$, and the logical operators $X_\ell = XZI$ & $Z_\ell = ZZI$. Using this information, we build the row matrix:

$$M = (G_X|G_Z) = \begin{array}{c} \overline{X}_\ell \\ \overline{Z}_\ell \\ s_1 \\ s_2 \end{array} \left(\begin{array}{cccc|cccc} \ell & 1 & 2 & 3 & \ell & 1 & 2 & 3 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right). \quad (2.93)$$

The next step is to perform the Gaussian elimination such that our final row matrix will be in the form $(I|A)$, A being the adjacency matrix of the ZX-encoding graph we are looking for. In this form, every row will correspond to a vertex stabilizer. While the stabilizers and logical operators can be found from a ZX-encoding graph by composing vertex stabilizers, performing row operations on the stabilizer operators and logical operators will untangle the vertex stabilizers.

However, a standard Gaussian elimination may not be sufficient to bring the matrix into the desired form. In the example shown in equation (2.93), the second row is empty on the left-hand side, making it impossible to bring $G_X \mapsto I$. To solve this, we point out that we may choose another encoding in the LC of that stabilizer code by performing local Clifford gates and qubit permutations. The effect of a Hadamard gate on a qubit will be to swap that qubit's column on the X side and on the Z side:

$$M \xrightarrow{H_1} \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right). \quad (2.94)$$

Now, we have a matrix on the left-hand side that can be brought to I through Gaussian elimination:

$$\xrightarrow{s_1 + \bar{Z}_\ell} \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right) = (I|A). \quad (2.95)$$

The matrix on the right-hand side corresponds to the adjacency matrix of the following ZX-encoding graph:



We note that, in general, the diagonal entries may be non-zero. In that case, they can be ignored, as there are no self-edges in graph states [71].

The Hadamard gate on the first physical qubit found in the original ZX-encoding graph of equation (2.92) disappeared in our final result in equation (2.96) above. This is because we had to change the encoding in order to perform the Gaussian elimination procedure. If one wants to find exactly the same encoding as the original stabilizer code, then it is possible to compute the LC orbit of the resulting ZX-encoding graph and choose the encoding that matches the stabilizers and logical operators.

The other operations available to bring G_X to I are to perform a phase gate $\sqrt{Z}$ on a qubit. Since this maps X to $Y = iXZ$, the effect is to add the qubit's column on the Z side to the one on the X side, leaving the Z column untouched. Finally, qubit permutations swap columns within G_X and G_Z . It is important to keep track of the qubit order in order to retrieve the ZX-encoding graph with the correct input and output vertices. All these operations, including the standard row operations, may be performed in any order.

2.5.5 Normal Form of the ZX-Encoding Graphs

In this subsection, we present the normal form of ZX-encoding graphs. As a reminder, two such ZX-diagrams may look different, but represent the exact same isometry because of the equivalence between LC gates and local complementations ((2.72), 2.73), as in equation (2.79). Eventually, we will need to compare ZX-encoding graphs. There exists a normal form [111] where two ZX-encoding graphs represent the same isometry if and only if they have the same graph and set of LC gates.

Theorem 13 ([111, theorem III.4]: Normal form). *Given an order on qubits, the normal form for stabilizer states as described below is unique in the sense where,*

$$|\text{graph} + \text{LC}\rangle = |\text{graph}' + \text{LC}\rangle \implies \text{graph} = \text{graph}' \quad \text{and} \quad \text{LC} \equiv \text{LC}'. \quad (2.97)$$

In particular, the normal form of ZX-encoding graphs is unique.

The normal form of a graph may be obtained following an algorithm:

1. Rewrite all LC gates as an element of this set: $\{I, \sqrt{X}, \sqrt{Z}, \sqrt{X}\sqrt{Z}, \sqrt{X}H, H\}$.

The LC gates are generated by $\{\sqrt{X}, \sqrt{Z}\}$. The set above spans all the combinations of these gates, but written such that the $\sqrt{X}$ gate is applied to the ZX-encoding graph first. This is because we will use the local complementation rule (2.72) next. To rewrite gates, we use the rules explained in Subsection 2.2.2, where the symplectic representation is introduced. For example $\sqrt{Z}H = \sqrt{X}\sqrt{Z}$, and $\sqrt{Z}\sqrt{X} = \sqrt{X}H$. Exceptionally, we do not write H as $\sqrt{X}\sqrt{Z}\sqrt{X}$.

Warning: the time order on the input wires is reversed. The first gate applied is the one on the right, and then the one on the left can be applied. Using the map-state duality (see Subsection 2.5.2), one could get back the same time ordering as for the output wires.

More experienced readers may skip this step, as it is not strictly necessary.

2. Apply all the $\sqrt{X}$ gates as local complementations using rule (2.72).

After each application of $\sqrt{X}$, some $\sqrt{Z}$ gates will appear on the neighbouring vertices. We enumerate the effect of these new gates on the possible LC gates:

$$I \mapsto \sqrt{Z}, \quad \sqrt{Z} \mapsto I, \quad \sqrt{X}\sqrt{Z} = \sqrt{Z}\sqrt{X}\sqrt{Z} = H, \quad (2.98)$$

$$\sqrt{X} \mapsto \sqrt{Z}\sqrt{X} = \sqrt{X}H, \quad H \mapsto \sqrt{Z}H = \sqrt{X}\sqrt{Z}, \quad \sqrt{X}H \mapsto \sqrt{Z}\sqrt{X}H = \sqrt{X}. \quad (2.99)$$

The first three mappings make the $\sqrt{X}$ disappear immediately, while the last three send it to another node. The latter case may seem bad. However, the vertex, which previously had a $\sqrt{X}$ before applying it now has either I , $\sqrt{Z}$, which means the second application will end the chain of $\sqrt{X}$ or H , where there will be a third application of $\sqrt{X}$ needed to stop the chain.

So we conclude it is possible to apply all $\sqrt{X}$ onto the graph using (2.72), such that the only remaining LC gates on open wires are $\{I, \sqrt{Z}, H\}$.

3. Bring the H to the vertex with the highest index possible.

This step is where qubit ordering becomes useful. Unless two Hadamard gates are applied to connected qubits, the pivoting rule (2.73) doesn't make them disappear, but rather transfers them from one qubit to another. So the ordering fixes this remaining degree of freedom.

We go through the effect of the pivoting rule on the neighbouring qubits:

$$I \mapsto H, \quad H \mapsto I, \quad \sqrt{Z} \mapsto H\sqrt{Z} = \sqrt{X}H. \quad (2.100)$$

In the last transformation, we need to go back to step 2. and apply the $\sqrt{X}$ gate until it disappears.

In this manuscript, we will always give a qubit ordering where the input qubits are indexed lower than the outputs. This implies that, on the logical qubits' side, the LC gates can only be $\{I, \sqrt{Z}\}$, while they can be $\{I, \sqrt{Z}, H\}$ for the physical qubits. This choice will make it easier to perform gauge fixing in Section 4.1.

For the convenience of the reader, we give an example of the procedure to bring a ZX-encoding graph to its normal form. We start by introducing the qubit ordering of the

ZX-encoding graph:

(2.101)

Then, we add some LC gates and perform the procedure to get the normal form:

First, we rewrote the LC gate on the 1st qubit. Second, we performed the $\sqrt{X}$ on the 0th qubit, which removed the edge between the 1st and 2nd qubit, and added $\sqrt{Z}$ on their output wires. We directly wrote $H = \sqrt{Z}\sqrt{X}\sqrt{Z}$ on the 1st qubit. Third, we checked that the remaining H gate is as high as possible, and it is because that vertex's only other connection is of lower index. It might seem tempting to try to transfer the H onto the logical to push it to the 2nd qubit after, but the pivoting rule doesn't allow that:

where we see that the H is stuck on the 0th qubit, which is connected to a higher indexed qubit, the 1st qubit. So we would simply end up going back to the original ZX-encoding graph.

In this case, there was no chain of $\sqrt{X}$, so we give another example with the same qubit ordering that has a small chain:

$$\begin{array}{ccccccc}
 H & \text{---} & \text{---} & \sqrt{Z} & & \text{---} & H\sqrt{Z} & & \text{---} & \sqrt{X}H & & \sqrt{Z} & \text{---} & \text{---} & H \\
 \text{---} & & \text{---} & & & \text{---} & & & \text{---} & & & & \text{---} & & \\
 & & \text{---} & & & & \text{---} & & & & \text{---} & & & \text{---} & \\
 & & \text{---} & & & & & & & & & & & & \\
 & & \text{---} & & & & & & & & & & & &
 \end{array}
 =
 \begin{array}{ccccccc}
 & \text{---} & \text{---} & & & \text{---} & H\sqrt{Z} & & \text{---} & \sqrt{X}H & & \sqrt{Z} & \text{---} & \text{---} & H \\
 & & \text{---} & & & & & & & & \text{---} & & & \text{---} & \\
 & & & & & & \text{---} & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & &
 \end{array}
 =
 \begin{array}{ccccccc}
 & \text{---} & \text{---} & & & \text{---} & \sqrt{X}H & & \text{---} & \sqrt{Z} & \text{---} & \text{---} & H \\
 & & \text{---} & & & & & & & & & & & & \\
 & & & & & & \text{---} & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & &
 \end{array}
 =
 \begin{array}{ccccccc}
 & \text{---} & \text{---} & & & \text{---} & H \\
 & & \text{---} & & & & \\
 & & & & & & \\
 & & & & & & \\
 & & & & & &
 \end{array}
 .$$

(2.104)

As all the LC gates are already written in the correct form and there are no $\sqrt{X}$ gates, we skip the first two steps. Step 3 says to move the H to the highest possible index, so we perform pivoting to push the H onto the 1st qubit. This changes the LC gates on that qubit to $\sqrt{X}H$, so the pivoting triggered a chain of $\sqrt{X}$. We get rid of the $\sqrt{X}$ in the following equality, and this finishes the procedure, as the H is as high as it can be.

Chapter 3

Review of Logic in Stabilizer QEC

We start by introducing the concept of fault-tolerance in quantum computation before explaining specific protocols. The first of such protocols, presented in section 3.2, introduces magic state distillation, which is the most prominent technique to generate non-Clifford logical gates. The next section, section 3.3, covers lattice surgery, a technique that enables entangling logical gates for surface-like QEC codes with 2D layouts and nearest neighbour interactions. Then, we discuss in section 3.4 the simplest way to perform logical Clifford gates, code automorphisms. Finally, we end the chapter with a quick discussion of dynamical automorphisms in section 3.5, where gauge fixing is added to the set of tools that generate logic. This idea will be further explored in section 4.2.

3.1 Fault-tolerant Quantum Computation

So far, QEC has been discussed in the context of a quantum memory: logical information is encoded, experiences noise, and gets corrected according to stabilizer measurements. To perform quantum computation on the information within a QEC code, we turn to active QEC as illustrated in figure 3.1.

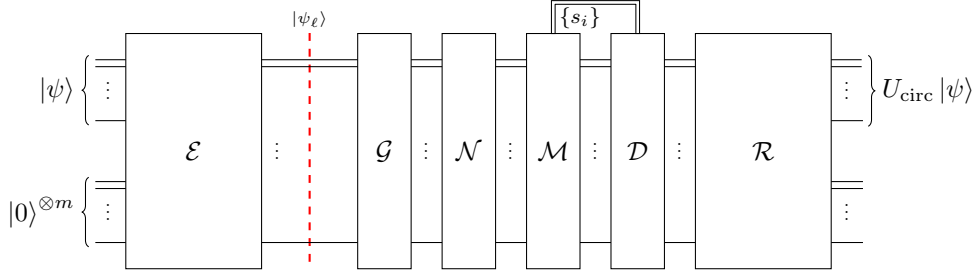


Figure 3.1: Active QEC protocol. The state $|\psi\rangle$ holding k logical qubits is encoded into the QEC code as $|\psi_{\ell}\rangle$ using m ancilla qubits. Then a logical gate $\mathcal{G}$ is performed, followed by a noise channel $\mathcal{N}$. Then a round, or cycle, of QEC is performed by measuring the stabilizers $\mathcal{M}$, and decoding the information to apply the correction $\mathcal{D}$. For circuits of depth D , these last four steps may be repeated D times. Finally, the recovery map decodes the encoded state $U_{\text{circ}} |\psi_{\ell}\rangle$ to get the final output of the quantum circuit implemented.

Because logical qubits are not directly accessible, finding protocols $\mathcal{G}$ to generate logical gates without corrupting the logical information is often hard. We'd like to avoid performing gates which "badly spread" errors. For example, performing a CNOT gate between two physical qubits could spread what was a single-qubit Pauli error into two-qubit errors (see equation (2.70)). Since a QEC code $[[n, k, d]]$ only tolerates $t = (d - 1)/2$ errors, this bad propagation of error may cause a correctable error to become uncorrectable, thus putting at risk the logical information.

More generally, every step, or gadget, described in figure 3.1 will be implemented imperfectly in some way or another, and may jeopardize the whole QEC protocol. Even the noise channel is only a model of the actual noise present in a quantum computer, and thus imperfectly represent the system.

A big piece of the puzzle that has been overlooked so far in this manuscript, but is worth mentioning, is decoding. The recovery map presented in subsection 2.3.2 was somewhat straightforward, but for a general stabilizer code, optimally decoding is $\#P$ problem [96], meaning this task is much harder than for classical linear codes. As it was alluded to,

different decoding algorithms will have different performance since they might not catch the same sets of errors. Thus, different noise models may affect the performance too.

Protocols that allow for quantum computation with logical qubits even with errors acting on the gadgets are called fault-tolerant (FT). For a quantum computer to be FT, every gadget has to work in synergy with the others. Using FT gadgets, it is possible to suppress the logical error arbitrarily low by scaling up or concatenating QEC codes, if the physical error rate p is under a certain threshold value $p < p_{\text{th}}$. This comes from the threshold theorem [106, 112, 113] that states it may be realizable to construct good enough FT quantum computers capable of useful computation, if the resources are good enough. Many experiments have shown signs that scaling is possible with state-of-the-art qubit control [5–7, 14].

According to their nature, different gadgets have different requirements to be considered FT. The gadgets we will be discussing in this chapter are FT logical gates $\mathcal{G}$. These FT gates transform a correctable error into another correctable error by definition. For a single code block encoded into a code with distance d correcting t errors, if the gate may generate an error s and the state has an incoming error e_1 , then the error after the gate must be $e_1 + s \leq t$. For gates involving two logical qubits encoded in the same code block, the same rule applies. However, it's a different story when two logical qubits are encoded in two code blocks each capable of correcting t errors. Then, if e_1, e_2 are the incoming errors on each qubit and s is the error generated by the 2-qubit gate for each qubit, an FT gate must respect $e_1 + e_2 + s \leq t$ for each code block. So errors can be leaked from another code block onto a given code block *if* the resulting error stays under that code block's capacity.

The word fault-tolerance is often thrown around without a proper definition. Here we offered a description that, although not complete, will be sufficient in the context of this manuscript. For a rigorous explanation of fault-tolerance we refer the reader to Gottesman's book on the subject *Surviving as a Quantum Computer in a Classical World* [114].

Transversal gates are the most straightforward way to build FT gates. While their definition may vary between source, we will follow Nielsen and Chuang's [115] definition: transversal gates are gates that are implemented by independent physical gates affecting a single qubit per code patch at a time. Often, the logical gate will be implemented by broadcasting a gate across the physical qubits. As such, they are trivially fault-tolerant. For example, the surface code introduced in Subsection 3.3.1 has a transversal H gate implemented by broadcasting H across all qubits. It also has a transversal CNOT between two code patches.¹ Because each CNOT is supported on a single qubit in a code patch,

¹We will explain the transversal procedure in Subsection 3.3.1.

this respects the transversal FT requirements.

The Eastin-Knill theorem [16] tells us it is impossible to have a fully transversal universal gate set. So, there must be at least one gate that requires a more complex FT protocol. These FT protocols will be the focus of this chapter.

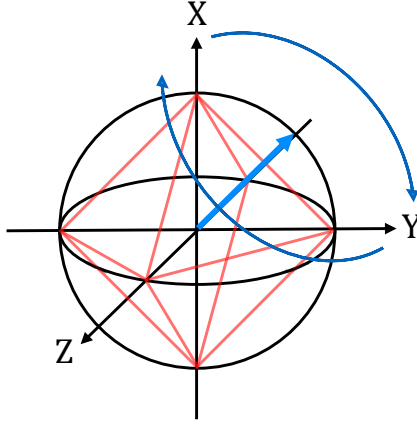


Figure 3.2: Illustration of the T state $T|+\rangle$ distilled in subsection 3.2.2 on the Bloch sphere. The volume reachable by Clifford operations from a Pauli eigenstate is outlined in the red octahedron [24]. The curved arrows indicate that a π rotation around this state swaps the X and Y operators, so the arrow lies on the XY plane.

3.2 Magic State Distillation

Magic state distillation was introduced by Bravyi and Kitaev [24] as a scheme to generate a fault-tolerant universal gate set by distilling *magic states* used to teleport a non-Clifford gate onto a logical qubit. In the context of magic state distillation, it is assumed that the lab has access to ideal Clifford gates, $|0\rangle$ state preparation, and perfect Pauli measurements [24].

Here, we will follow Gottesman's introduction to magic state distillation [114]. There are many different distillation protocols, but we focus on the one based on the 15-qubit Reed-Muller code [24, 116, 117]. The basic idea is to encode noisy magic states into QEC codes that have a transversal non-Clifford gate, measure the syndromes and discard the states if they are non-trivial. The outcome can be decoded to get a single "purer" magic state. Once we have achieved the desirable accuracy, the distilled state is encoded into the QEC code that lacks the non-Clifford gate, and a magic state injection is performed to transfer the gate from our magic state onto a logical qubit.

As mentioned above, magic states are the states used for magic state injection. This definition is vague, so we try to make it clearer by illustrating in figure 3.2 the T state, the magic state used below to generate T gates. In this rendering, the state is outside the volume covered by Clifford operations from a Pauli eigenstate [24]. All such states are often called magic states. However, not all those states are good for magic state injection.

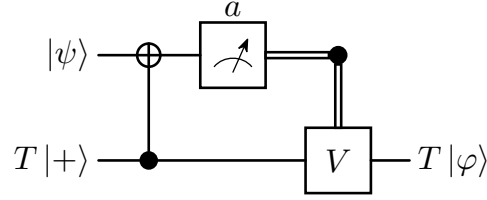


Figure 3.3: Magic state injection protocol for T states. The circuit inputs a given state $|\psi\rangle$, and the magic state $T|+\rangle$, and outputs $T|\psi\rangle$. The outcome of the Bell measurement in the computation basis is given by the bit a . With this bit, a classical control- V gate is performed on the magic state, where $V = T X T^* = e^{i\pi/4} S^* Y$.

For more details on this, we refer the reader to Section 13.2.1 of Gottesman's book [114].

3.2.1 Magic State Injection

The structure of magic state injection relies on quantum state teleportation [118], as gates can be teleported too [64]. Although we highly encourage readers unfamiliar with both circuits to take a careful look at how it works, we directly jump to the circuit of interest illustrated in figure 3.3.

To show that this circuit does perform a T gate on the given state $|\psi\rangle$, we translate it into the ZX-calculus. Looking at equations (2.64) and (2.32), the T gate is $\textcircled{\frac{-\pi}{4}}$, and its conjugate is $\textcircled{\frac{-\pi}{4}}$. Next, the classical controlled gate can be modelled as adding the classical control bit a in the phase of the target spider [41]. So keeping in mind $V = T X T^*$ and X are red spiders with a π phase, the control- V operation may be decomposed as $\textcircled{\frac{-\pi}{4}} \textcircled{a\pi} \textcircled{\frac{\pi}{4}}$. Finally, our quantum state $T|+\rangle$ can be translated as a one-legged Z spider and the $\pi/4$ spider above fused together $\textcircled{\frac{\pi}{4}}$. We are ready to take a look at the circuit equalities:

$$\begin{aligned}
 & |\psi\rangle \text{---} \textcircled{\frac{\pi}{4}} \text{---} \textcircled{a\pi} \text{---} \textcircled{\frac{-\pi}{4}} \text{---} \textcircled{a\pi} \text{---} \textcircled{\frac{\pi}{4}} \text{---} \\
 & \stackrel{(f)}{=} |\psi\rangle \text{---} \textcircled{a\pi} \text{---} \textcircled{\frac{\pi}{4}} \text{---} \textcircled{a\pi} \text{---} \textcircled{\frac{\pi}{4}} \text{---} \\
 & \stackrel{(\pi)}{=} |\psi\rangle \text{---} \textcircled{\frac{\pi}{4}} \text{---} = T|\psi\rangle . \tag{3.1}
 \end{aligned}$$

We do get back $T|\psi\rangle$.

As a reminder, this picture is on the logical level. In application, $|\psi\rangle$ is the logical qubit holding precious information, while $T|+\rangle$ is the encoded magic state. This circuit will be fault-tolerant as long as the CNOT and V gates are transversal to the code. Wait a minute, aren't there T gates hidden in V ? Yes, they are, but conjugation of Pauli operators by T only maps to gates that are in $\mathcal{C}_2^{(1)}$, thus $V = T X T^* = e^{i\pi/4} S^* Y$. So if the base code has

a transversal S gate, it is, in fact, possible to fault-tolerantly perform a classical control- V operation. This highlights a limitation of magic state injection: it only works if the control- V corresponding to a non-Clifford gate G is realizable using Clifford operations.

3.2.2 Magic State Distillation using the 15-qubit Reed-Muller Code

The 15-qubit Reed-Muller code² [24, 116, 117], whose stabilizers are shown in Table 3.1, has a transversal T^* gate implemented by performing T on the 15 physical qubits [114]. It also has a transversal T gate by performing T^* on the physical qubits instead [116]. The two are true at the same time, since $T^{15} = T^*$ and vice versa.

Z	Z	I	Z	I	I	Z	I	I	I	I	I	I	I	I
Z	Z	I	I	Z	Z	I	I	I	I	I	I	I	I	I
Z	Z	I	I	I	I	I	Z	I	I	Z	I	I	I	I
Z	Z	I	I	I	I	I	I	Z	Z	I	I	I	I	I
Z	Z	I	I	I	I	I	I	I	I	I	Z	I	I	Z
Z	Z	I	I	I	I	I	I	I	I	I	I	Z	Z	I
Z	I	Z	Z	I	Z	I	I	I	I	I	I	I	I	I
Z	I	Z	I	I	I	I	Z	I	Z	I	I	I	I	I
Z	I	Z	I	I	I	I	I	I	I	I	Z	I	Z	I
Z	I	I	Z	I	I	I	Z	I	I	I	I	Z	I	I
I	I	I	I	I	I	I	X	X	X	X	X	X	X	X
I	I	I	X	X	X	X	I	I	I	I	X	X	X	X
I	X	X	I	I	X	X	I	I	X	X	I	I	X	X
X	I	X	I	X	I	X	I	X	I	X	I	X	I	X

Table 3.1: Stabilizers of the 15-qubit Reed-Muller code. Each row represents a generator.

Since the logical T^* gate which we call $T_\ell^* = T^{\otimes 15}$ is transversal, it commutes with the codespace projector Π_c . So, if we measure the stabilizers on the 15 noisy magic states $(T|+\rangle)^{\otimes 15}$ state, we effectively perform a T_ℓ^* on the logical $|+\rangle = |+\rangle^{\otimes 15}$:

$$\Pi_c(T|+\rangle)^{\otimes 15} = \Pi_c T^{\otimes 15} |+\rangle^{\otimes 15} = T^{\otimes 15} \Pi_c |+\rangle^{\otimes 15}, \quad (3.2)$$

$$= T^{\otimes 15} |+\rangle^{\otimes 15} = T_\ell^* |+\rangle, \quad (3.3)$$

where to go from the first line to the second, we used the fact the projector stabilizes encoded states meaning $\Pi_C |+\rangle = |+\rangle$.

²This code is the same as the smallest 3D colour code.

If there is some noise, then at least one syndrome measured will be -1 rather than 1. So this means that the projection onto the codespace failed, and this batch of qubits is discarded. The procedure needs to start over from scratch.

However, if the syndromes are trivial (all +1), then we have the state we wanted (3.3). This means we have $T_\ell^* |+\ell\rangle$ and not $T_\ell |+\ell\rangle$. A phase gate S maps T^* to T : $T = \sqrt{S} = S\sqrt{S}^{-1} = ST^*$. The S gate is transversal, so it can be safely applied to the outcome state to get the magic state at the logical level.

The last step in the protocol is to decode the state to get a "purer" $T_\ell |+\ell\rangle$ state. We summarize the protocol below.

1. Start with 15 noisy $T_\ell |+\ell\rangle$ states.
2. Measure the 15-qubit Reed-Muller code stabilizers (Table 3.1). If the syndromes are non-trivial, discard the qubits and start over. If not, continue the protocol.
3. Perform a logical phase gate S .
4. Decode the 15-qubit Reed-Muller code to get a single "purer" $T_\ell |+\ell\rangle$ states. Discard the remaining 14 qubits.

To further enhance the fidelity between the output and the ideal magic state, we can use 15 distilled magic states as input into this protocol and obtain a single, even more high-fidelity magic state. If we trust that our Clifford gates are close enough to perfect, this concludes the protocol.

However, since it is assumed the lab has *perfect* Clifford gates, each physical qubit in this scheme may be encoded into a code with transversal Clifford gates such as the surface code to obtain better results. It has an encoding rate of 9 physical qubits to 1 logical for code distance $d = 3$ (see subsection 3.3.1).

It now becomes clear why there is such a big resource overhead: for each increment in fidelity for a single magic state, the scheme processes $15 \cdot 9 = 135$ qubits, or even more for a higher surface code distance achieving better fidelity. Also, each time the protocol is applied, there is a probability that the syndromes measured are non-trivial, so everything needs to be started over, including all the previous rounds of distillation.

3.2.3 Contributions and Outlooks

The first scheme developed to generate a fault-tolerant universal gate set is not magic state distillation, but Shor's Toffoli state preparation using cat states in 1996 [119]. Also,

Knill, Laflamme and Zurek [116] propose to use a code that has transversal CNOT and T , and get the missing H gate by a procedure similar to the state injection presented above. However, Bravyi and Kitaev’s magic state distillation offered a procedure to fault-tolerantly realize *any* non-Clifford gate using Clifford gates and basic state preparation, and they characterized which states are necessary to achieve this.

The magic state distillation’s assumptions might seem a bit contrived, since it assumes access to ideal Clifford gates. However, the surface code has access to FT transversal Clifford gates,³ so the only computation missing is a non-Clifford gate. The prevalence of magic state distillation is thus associated with the rise of surface codes. As for the other two assumptions, almost perfect state preparation of an initial $|0\rangle^{\otimes n}$ may be reasonable, but perfect measurement of Pauli operators may be rather hard to achieve, even approximately, depending on the implementation. That being said, syndrome extraction circuits may be optimized to consider measurement errors [120].

Magic state distillation has its advantages and drawbacks. The distillation scheme itself can be done separately from the logical information, which further protects it from our noisy computations. However, a major drawback of this procedure is resource overhead. Magic state distillation dominates resource estimations for the whole of FTQC. So effectively, the non-Clifford gates become bottlenecks in quantum computation. Since Clifford circuits can be simulated classically [15, 94], being able to compute non-Clifford gates is an essential step towards quantum advantage. The strain on the production of magic states can be alleviated by making magic state factories [82]. This architecture choice means there should always be a magic state readily available for quantum computation, but it requires a lot of resource overhead.

Up to this day, there is ongoing research to optimize the resource overhead of magic state distillation, and its resource estimate has dropped [27]. For example, it can be minimized by replacing the encoding of the 15 qubits with the Bivariate Bicycle code [121], a code which has a high threshold and low overhead, instead of the surface code [122]. A really promising different approach was optimized by Gidney et al. [123] in 2024 called magic state cultivation, which we discuss below.

3.2.4 Magic State Cultivation

As the name suggests, magic state cultivation [123] protocols make the encoded states grow. The basic structure of the magic state cultivation protocol is the following: start distillation

³See Section 3.3.

using small codes with a transversal T to initialize the magic states, and then grow it into bigger and better codes until the desired accuracy is obtained. Thus, the protocol can be separated into three stages: injection, cultivation, and escape. After elaborating the protocols, they are simulated and with circuit-level depolarizing noise with an error rate of 10^{-3} . They estimate that they reduce the resources required to achieve accuracy of the order of 10^{-9} by an order of magnitude compared to previous work. They also gain 2 orders of magnitude in accuracy by cutting the error rate in half.

First comes the injection stage. In their work, the authors explore three different ways to initialize $T_\ell |+\ell\rangle$ in the $d = 3$ 3D colour code [88], which has a transversal T gate [124]. They find that a "unitary injection" circuit performs best. This circuit is not a standard encoding circuit of the colour code because it initializes the T magic state at a point where the state is protected from Z errors.

The next goal is to grow the code to increase fault tolerance, i.e. the cultivation stage. It contains 3 steps: check, grow and stabilize. In the check steps, we check the parity of the state. So a T_ℓ^* is applied to get $|+\ell\rangle$ state, and then the X parity is measured in the $+1$ eigenspace of the SWAP operator and verified using time-reversed parity checks. The Grow step consists of preparing the new physical qubits in Bell pairs, arranged in the continuation of the hexagonal lattice typical of colour codes. To connect the new qubits, a subset of stabilizers is measured.⁴ In the Stabilize step, three colour code cycles [125] are performed to make sure the glue sticks.

The final step is the escape stage. Pretty quickly, growing the colour code becomes infeasible with current decoders. So growing the code lowers the number of faults, but it doesn't increase the code distance. As such, the code needs to grow into a code that's easier to decode: the surface code. Since the colour codes and the surface codes lie on different tessellations, i.e. a hexagonal lattice and a square lattice, this isn't straightforward, and previous work proved to be insufficiently efficient. Thus, the authors construct their own method: they first grow the colour code into a surface code with one of its corners "folded" onto itself, which they call the grafted code. This triangular section of the code corresponds to the old colour code. Once they have the grafted code, decoding becomes much easier, so now the state is protected by a QEC code with a bigger code distance than the initial $d = 3$. However, the code distance may account for smaller error strings than what the grown colour could do. They create a new code by merging some stabilizers from the colour and surface code in a way that maximizes the tolerance for faults and the code distance.

In the paper, the authors describe the decoding procedures for each step. While decod-

⁴If the colour code has grown out in the direction of the green and blue edges, then the red plaquettes are measured. For more details on colour codes, see Ref. [89].

ing is an essential step in QEC, it is not covered in this manuscript, so we do not summarize their techniques.

3.3 Lattice Surgery

Magic is a staple of FTQC, but that’s not the end of the picture. It has been proved that no QEC code has the full transversal Clifford group for more than a single qubit [126]. So it becomes essential to consider FT protocols for Clifford gates. Lattice surgery answers this question for a particular class of QEC codes.

Lattice surgery was first proposed by Horsman et al [17] as a tool to create FT protocols for performing CNOTs on surface code blocks, or patches, with a 2D layout and only nearest neighbour interactions.

In this section, we first introduce the surface code, and its transversal gates. Then, based on [17, 99, 127], we explain lattice merging and splitting operations, the FT CNOT procedure, and the S gate procedure. We finish off this section by going over the contributions and outlooks of this work.

3.3.1 Primer on the Surface Code

The surface code [86, 87] is a CSS and QLDPC code defined on a planar square lattice. The qubits can either lie on the vertices of the lattice or on the edges. Here, we place the qubits on the vertices of the grid, as shown in figure 3.4. Thus, we present the rotated surface code that has a better scaling $\llbracket d^2, 1, d \rrbracket$ [17]. The stabilizers are then placed on the faces of the lattice. The X and Z -type stabilizers alternate on the grid like a checker board.

The code was first constructed on a torus where there are no boundaries. Since this topology isn’t implementable experimentally, we present the bounded surface code. The rounded 2-qubit stabilizers are added to create boundaries for the code. There are two types of boundaries: Z -boundaries with Z -type 2-qubit stabilizers, and X -boundaries with the X -type 2-qubit stabilizers.

This surface code encodes a single logical qubit. Considering the Z -boundaries to be associated to Z -type stabilizers, the logical operator Z_ℓ connects both Z -boundaries. This is because the Z Pauli string needs to commute with all stabilizers, so it must have even overlap with all X -type stabilizers. It is possible if the strings connects the 2-qubit Z -type stabilizers, or in other words if it connects both Z -boundaries. In figure 3.4, we illustrate a representative that goes along the bottom X -boundary. Any other Pauli string equivalent to this one up to stabilizers is also a logical Z operator. Similarly, X_ℓ connects the two X -boundaries, so it lies along the Z -boundary. The code distance, which corresponds to

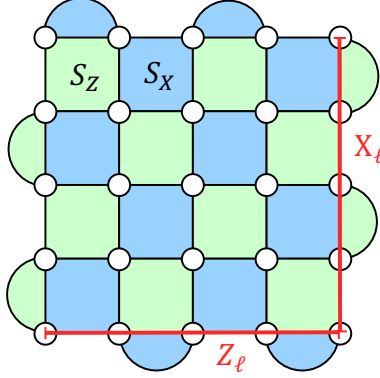


Figure 3.4: Example of a $[[25, 1, 5]]$ rotated surface code. The white dots represent physical qubits. The green(blue) plaquettes represent the Z -type(X -type) stabilizers. The red lines represent the logical operators: a representative of Z_ℓ is a tensor product along the lower X -boundary, while the X_ℓ lies on the right-hand side Z -boundary.

the minimum weight of logical operators, is then dictated by the size of the lattice. In figure 3.4, we have a 5×5 lattice, so $d = 5$.

We point out that it is possible to encode more logical qubits by poking holes the lattice, i.e. removing stabilizers from the generating set. These removed stabilizers are then promoted to logical operators in a similar fashion as the gauge qubits in subsystem codes (see Subsection 2.3.5). Then, the boundary around the removed stabilizers, corresponding to the composition of these stabilizers, create a new logical operator. Its anti-commuting counterpart connects this new boundary to the previous one of opposite type. The holes are also called defects of the lattice. This way of encoding qubits in the surface code is harder to handle because it does not support any transversal operations [17]. Logical CNOTs can be implemented via *braiding*. We will not further explain these techniques or refer to defects since lattice surgery was found to outperform every other method [128].

The surface code has a full transversal Clifford gates set [129, 130]. In fact, all CSS codes have a transversal CNOT [114], and all self-dual CSS codes have a transversal H gate [61]. The latter is pretty evident since H swaps X and Z , and self-dual CSS codes have identical X and Z -type stabilizers. From this argument, one would suspect that self-dual CSS codes also have a transversal S gate since it maps $X \mapsto Y = iXZ$. However, that additional phase adds another constraint: since it has a period of 4, the weight of the stabilizers must respect $w(s_i) = 0 \pmod{4}$ for all s_i in the group. So doubly even self-dual CSS codes have transversal S gates [61].

Looking at the bulk stabilizers, one might think the surface is a doubly even code,

but because of the 2-qubit stabilizers added to bound it, it is not. So the author glossed over the practical implementation of the Clifford gates for the surface code in Section 3.2, implying they are all transversal straightforward way. Sorry! The picture is a bit more complicated than simply transversal and non-transversal gates, and we will address this in subsection 3.3.5. For now, know that the FT protocols for the S gate don't create the same bottlenecks as for non-Clifford gates.

With a new sense of paranoia, we revisit the transversality of H gate. By looking at the symmetry of the surface code patch in figure 3.4, one can notice that Hadamard gates applied transversally maps the stabilizers to each other *up to a geometrical 90° rotation*. This does not affect the information badly, meaning the H gate is truly transversal. However, a protocol to rotate the surface code is needed to make it compatible for future lattice surgery protocols with neighbouring logical qubits [97]. It simply requires grow the code patch by measuring stabilizers with neighbouring qubits and measure out some of the qubits such that the code rotates [99, Fig. 1]. These types of operations are called *code deformation*.

The transversal CNOT is implemented by applying CNOTs between all pairs of qubits between two blocks of surface code. While CNOTs between qubits within a code block can spread errors in a non-FT way, here the CNOT connects qubits in different code blocks. If an error spreads from one code block to the other, it remains correctable for both code blocks individually.

While this is exactly what we are looking for in theory, in practice it may be difficult to implement. Many quantum computer architectures only have nearest neighbour interactions, such as superconducting quantum computers whose qubits lay on a chip. So operations that require the whole code block to move around may be out of reach. This is where lattice surgery comes in.

3.3.2 Lattice Merging

Given two code surface code patches C_1 , C_2 , it is possible to merge the two code blocks into a single bigger code [17, 99] by measuring operators along the two boundaries of the code followed the rest of the stabilizers d times. The Z -merge is explained below, and the X -boundary merge is implemented in a similar way by exchanging the Hadamard and computational bases.

Before a merging operation, the code patches must be in a specific orientation with each other. As illustrated in figure 3.5a, the 2-qubit Z -type stabilizers are aligned with each other in order to perform the Z -merge. The inversion of the lattice can be achieved

by adding a column(row) of qubits to the code patch by measuring their stabilizers and measuring out the column(row) on the opposite side [99, Fig. 2].

To fill in the gaps of the lattice and merge the patches in figure 3.5a, the new X -type stabilizers are measured. They are highlighted in figure 3.5b. They anti-commute with the original Z -type 2-qubit stabilizers, so the first measurement is random, and it cannot be corrected a priori. This, and accounting for measurement errors, is why the measurement sequence is repeated $d - 1$ other times.

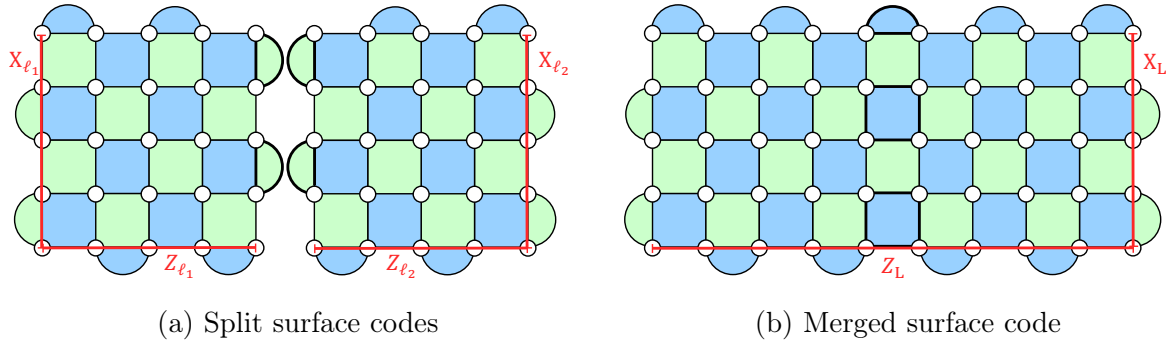


Figure 3.5: Lattice surgery with the rotated surface code [99]. The stabilizers with bold contour are the ones that were measured (b) to merge the codes or (a) to split the code. The logical operators of each code have support along the red lines.

Notice how the product of the of these new stabilizers correspond to the logical operator $X_{\ell_1}X_{\ell_2}$ of the previous stabilizers. After the d rounds of stabilizer measurements m_i , the outcome $M = \prod_{i=1}^d m_i$ of the measurement may be decoded.

Suppose the logical qubits are in the states $|\psi_{\ell_i}\rangle = \alpha_i |0_\ell\rangle + \beta_i |1_\ell\rangle$, where $|\alpha_i|^2 + |\beta_i|^2 = 1, i \in 1, 2$, and $\{|0_\ell\rangle, |1_\ell\rangle\}$ are the the basis for the codewords of the surface code. Then, the state after the measurement of $X_{\ell_1}X_{\ell_2}$ is

$$\frac{1}{\sqrt{2}}(|\psi_{\ell_1}\rangle \otimes |\psi_{\ell_2}\rangle + (-1)^M X_{\ell_1} |\psi_{\ell_1}\rangle \otimes X_{\ell_2} |\psi_{\ell_2}\rangle). \quad (3.4)$$

This state can be expanded in function of the measurement outcome:

$$\text{if } M=0, \quad \frac{1}{\sqrt{2}}[(\alpha_1\alpha_2 + \beta_1\beta_2)(|0_\ell 0_\ell\rangle + |1_\ell 1_\ell\rangle) + (\alpha_1\beta_2 + \alpha_2\beta_1)(|0_\ell 1_\ell\rangle + |1_\ell 0_\ell\rangle)] \quad (3.5)$$

$$\text{if } M=1, \quad \frac{1}{\sqrt{2}}[(\alpha_1\alpha_2 - \beta_1\beta_2)(|0_\ell 0_\ell\rangle - |1_\ell 1_\ell\rangle) + (\alpha_1\beta_2 - \alpha_2\beta_1)(|0_\ell 1_\ell\rangle - |1_\ell 0_\ell\rangle)]. \quad (3.6)$$

Then, the logical operators of the merged code may be mapped to

$$|0_L\rangle \mapsto \frac{1}{\sqrt{2}}(|0_\ell 0_\ell\rangle + (-1)^M |1_\ell 1_\ell\rangle) \quad (3.7)$$

$$|1_L\rangle \mapsto \frac{1}{\sqrt{2}}(|0_\ell 1_\ell\rangle + (-1)^M |1_\ell 0_\ell\rangle). \quad (3.8)$$

As long as the measurement outcome is classically tracked, then the logical operators of the new code are well-defined. Knowing this, the state after the measurement can be succinctly written:

$$M_Z(|\psi_{\ell_1}\rangle \otimes |\psi_{\ell_2}\rangle) = \alpha_1(\alpha_2 |0_L\rangle + \beta_2 |1_L\rangle) + (-1)^M \beta_1(\alpha_2 |1_L\rangle + \beta_2 |0_L\rangle), \quad (3.9)$$

where one can notice that the right hand-side is the state $|\psi_{\ell_2}\rangle$ weighted by α_1 , and the left hand-side is $X_{\ell_2} |\psi_{\ell_2}\rangle$ weighted by β_1 .

3.3.3 Lattice Splitting

To Z -split the merged surface code into two surface codes, so going from 3.5b to 3.5a, the contoured Z -boundary stabilizers in 3.5a are measured. They anti-commute with the contoured operators in the merged code, so the measurements kick them out of the centre of the code. If some of the new stabilizers have a -1 measurement outcome, it can be classically updated as the new reference, and no further correction is needed.

The new stabilizer measurements split Z_L in two, thus the name Z -split. In fact, $\langle Z_L \rangle = \langle Z_{\ell_1} Z_{\ell_2} \rangle$. However, the measurements leave X_L unchanged. The new X logical operators are equal to the previous ones $X_L = X_{\ell_1} = X_{\ell_2}$. So, this means that the Z -split performs the following transformation: $|0_L\rangle \mapsto |0_\ell 0_\ell\rangle$ & $|1_L\rangle \mapsto |1_\ell 1_\ell\rangle$.

As for the merge operation, the X -split behaves in a similar way with the Hadamard and computational bases swapped.

3.3.4 FT Protocol for the CNOT

The fault-tolerant protocol for the CNOT between surface code patches starts with three logical qubits, the control qubit, the ancilla qubit and the target qubit $|C\rangle \otimes |+\rangle \otimes |T\rangle$.

The first step is a X -merge between the control qubit and the target qubit. Suppose that the control target is in the following normalized state $|C\rangle = \alpha |0_\ell\rangle + \beta |1_\ell\rangle = \alpha' |+\ell\rangle + \beta' |-\ell\rangle$.

Then,

$$M_X(|C\rangle \otimes |+\rangle) = \alpha' |+_L\rangle + \beta' |-_L\rangle \quad (3.10)$$

$$= \frac{1}{\sqrt{2}}(\alpha' + (-1)^M \beta') |0_L\rangle + \frac{1}{\sqrt{2}}(\alpha' - (-1)^M \beta') |1_L\rangle. \quad (3.11)$$

If $M = 1$, X_L can transform the state into the same as if $M = 0$, $\alpha |0_L\rangle + \beta |1_L\rangle$.

The second step is to perform a Z -split between these same qubits to entangle them:

$$S_Z(\alpha |0_L\rangle + \beta |1_L\rangle) = \alpha |0_\ell 0_\ell\rangle + \beta |1_\ell 1_\ell\rangle. \quad (3.12)$$

The last step is to perform a Z -merge between the ancilla and target qubits. Suppose $|T\rangle = \gamma |0_\ell\rangle + \delta |1_\ell\rangle$:

$$\begin{aligned} (\alpha |0_\ell 0_\ell\rangle + \beta |1_\ell 1_\ell\rangle) \otimes \gamma(|0_\ell\rangle + \delta |1_\ell\rangle) \mapsto \\ \alpha\gamma |0_\ell 0_L\rangle + (-1)^M \beta\gamma |1_\ell 1_L\rangle + (-1)^M \alpha\delta |0_\ell 1_L\rangle + \beta\delta |1_\ell 0_L\rangle. \end{aligned} \quad (3.13)$$

A classical update can remove the phase difference in case $M = 1$. Then, we can write the state as

$$\alpha |0_\ell\rangle \otimes (\gamma |0_L\rangle + \delta |1_L\rangle) + \beta |1_\ell\rangle \otimes (\delta |0_L\rangle + \gamma |1_L\rangle), \quad (3.14)$$

which corresponds to the action of a CNOT on the control and target qubits.

3.3.5 FT protocol for the Phase Gate

There are many FT protocols that inject the phase gate into the surface code, which will be discussed in the outlooks of this section. For simplicity we adapt the magic state injection protocol seen in section 3.2.1 for this purpose. More precisely, the circuit represented in figure 3.3 may be reused, but with the initial states $|\psi_\ell\rangle$ and $S|+_\ell\rangle = |Y_\ell\rangle$ each encoded into a surface code patch. However, the control- V operation dependent on the Bell measurement outcome needs to be updated to implement the S gate. So V becomes $SXS^* = Y$. It is easy to reproduce equations for (3.1) since the phase gate is also a Z -spider, but with phase $\frac{\pi}{2}$ instead of $\frac{\pi}{4}$ for the T gate.

This protocol consumes $|Y\rangle$ states which are not magic states. So it does not require magic state distillation, which creates the bottleneck for producing T gate. Instead a physical $|Y\rangle$ state can be encoded, or injected, into a surface code patch followed by d rounds of QEC to get a high-fidelity $|Y_\ell\rangle$ state [17].

3.3.6 Contributions and Outlooks

The underlying FT protocol to perform a CNOT was proposed by Gottesman [131], but it was Horsman et al. [17] that found a circuit-level implementation of this idea for surface codes that respects the constraints of superconducting quantum computers: 2D layout and nearest neighbour interactions.

Following this work, lattice surgery was optimized such that it overshadows other concurring techniques such as braiding in terms of efficiency [128]. The state injection protocols were also improved upon by using twist defects [132, 133] and then optimized [134] allowing the $|Y_\ell\rangle$ state to be produced by classical updates of basis, making it trivially FT. Finally, lattice surgery may be generalized to QLDPC codes [135], and improved by looking at these schemes from a gauge fixing perspective [136].

Vuillot et al. [99] that lattice surgery and code deformation may be realized as gauge fixing. So there is some overlap between lattice surgery, and dynamical automorphisms, presented in Section 3.5. Also, lattice surgery has been represented in the ZX-calculus, making the connection with our results in Chapter 4. This will be discussed further in the discussion, Chapter 5.

3.4 Code Automorphisms

The simplest way to generate logical Clifford gates is through code automorphisms. Here, logical gates are generated using single-qubit Clifford gates and qubit permutations, such that the resulting gates are transversal.

Given a unitary operator U acting on the encoded state $|\psi_\ell\rangle$ associated with the stabilizer group S , then its action on the encoded state can be mapped to its action on the stabilizers:

$$U |\psi_\ell\rangle = U s_i U^* U |\psi_\ell\rangle \implies U : s_i \mapsto U s_i U^* \forall s_i \in S. \quad (3.15)$$

The set of unitary operators that normalize the stabilizers is denoted $N_U(S)$. It is different from $N(S)/S$ because that group is defined within the n -qubit Pauli group, whereas $N_U(S)$ may be generated by any unitary operator, i.e. Clifford or non-Clifford operators. Since the latter are harder to deal with, code automorphisms are defined as the intersection of $N_U(S)$ and the n -fold tensor product of the single-qubit Clifford group [15].

In other words, code automorphisms are transformations generated by LC gates and qubit permutations that bring the codespace back to itself $\mathcal{A} = \mathcal{C}_2^{(1)} \rtimes S_n$, where S_n is the symmetric group generated by permutations. Since LC gates generated by $\{\sqrt{Z}, H\}$ each swap two Pauli operators (see equation (2.27)), code automorphism is well suited to be represented in the symplectic representation, where the phases are disregarded (see Subsection (2.2.2)). Because this type of logic is generated by Clifford operations, it is not thought to be able to generate logical non-Clifford gates, but only Clifford gates.

We note that, unlike the logical operators defined as $N(S)/S$, code automorphisms don't necessarily have a nontrivial action on the code space. The action of a code automorphism must then be checked afterwards by performing the Clifford gates and qubit permutations of that automorphism. Since qubit permutations can be kept track of classically, code automorphisms, if they generate logic, form a subset of transversal gates.

There exist other definitions for automorphisms: sometimes only LC gates or only qubit permutations are considered. Interestingly, for some symmetric codes, these lead to non-trivial logical action even if they are more constrained. This implies logical entanglement may be created without touching the physical qubits by classically relabelling them. For example, in the $[[4, 2, 2]]$, qubit permutation can entangle logical qubits [137]. The class of codes that have entangling operations as qubit permutations is called phantom codes [138].

The code automorphisms may be computed by mapping the stabilizer code to binary linear codes and finding the latter code's automorphism group [139]. We will use this work done by Sayginel et al. and the python package to compute code automorphisms.

We briefly acknowledge that fold-transversal [\[140\]](#) gates have recently emerged as a generalization of folding the surface code for CSS codes. It carefully adds physical 2-qubit gates as a possible tool to generate FT logic. The concepts required to understand this technique are outside the scope of this manuscript. However, one may wonder what else could be added to our arsenal in order to perform logic in stabilizer codes beyond code automorphisms, as we know that

3.5 Dynamical Automorphisms

While code automorphisms are useful because of their inherent FT properties, they can't give the full logical n -qubit Clifford group [126]. This raises the need to go beyond this simple tool. While we can look to magic state injection or lattice surgery (for certain codes), in this section we introduce a new paradigm based on dynamical codes: the measurement sequence induced by the dynamical code may generate logic, all while measuring relevant information for decoding. [28, 38]. In other words, the logical information may go beyond the base code and into other codes in order to perform logical gates.

This idea was first explored by Paetznick and Reichardt [28], where they use a $[[15, 7, 3]]$ Hamming code [25] and use 6 qubits as gauge qubits to generate a H gate. This is because the $H^{\otimes 15}$ applied to the code may have a nontrivial action on a gauge qubit. This can be corrected by measuring their Z operators and performing a bit flip if necessary. Since these codes have a transversal CCZ gate [25], this generates the full universal gate set [141].

This concept, at its simplest, has been further explored in the form of code switching, first suggested by Anderson et al. [142]. As the name indicates, this groups procedures that transfer information from a base code into another code. The goal of switching to another code is to access its set of transversal gates, so the two codes are chosen such that their respective sets of transversal gates form a universal gate set.

In their work, Anderson et al. explain a code switching procedure between the Steane's 7-qubit code [85] and the 15-qubit Reed-Muller code (see Table 3.1), which are respectively the smallest 2D and 3D colour codes. Their work highlights that these two codes have a parent subsystem code, so code switching may be seen as gauge fixing. As mentioned in section 3.2.2, the latter has a transversal non-Clifford, while the Steane code has the rest of the single-qubit Clifford gates transversally [119, 143]. The code switching between the 2D and 3D colour codes has been generalized to larger code size and dimensions [29, 30].

code switching need not necessarily be implemented through gauge fixing in order to be fault-tolerant. In fact, it is possible to convert the 5-qubit code (figure 2.5) and the Steane 7-qubit code discussed above in a FT way using 2-qubit gates [144], even if 2-qubit gates within the same code patch are generally prohibited in FTQC. This technique is called code deformation. However, these can be replaced by gauge fixing [1], so we will not further expand on it.

code switching has more to offer than going back and forth between two codes. In fact, it is possible to find a sequence of gauge fixings that connects any two stabilizer codes [145]. Dynamical automorphisms (DAs) [38] generalize this idea in order to explore a bigger

space of codes to create logical action: to achieve this, DAs make use of physical single-qubit Clifford gates, qubit permutations and measurements. The tool used to navigate the space is gauge fixing, implemented through measurement, which connects codes that have common parent subsystem codes. We summarize the scheme proposed in figure 3.6.

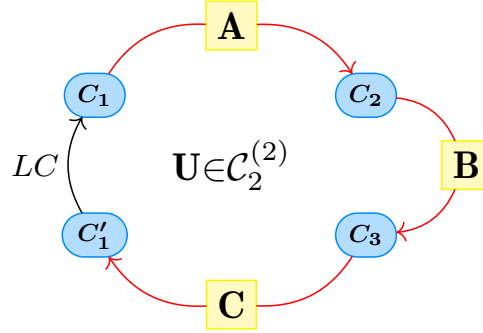


Figure 3.6: Sketch of a dynamical automorphism implementing a gate U in the 2-qubit Clifford group using 3 gauge fixings. The blue bubbles represent stabilizer codes. The red arrows represent gauge fixings which use the subsystem codes in the yellow squares. The last arrow represents some LC gate that maps the encoding back to itself up to the logical action U .

We allow ourselves measurements, rather than 2-qubit Clifford gates, because the measurement circuits may be more easily made fault-tolerant. Fault-tolerant Pauli measurements are non-destructive. Instead of measuring the physical qubits themselves, the desired information is transferred through CNOTs onto ancilla qubits that are measured [114]. While the CNOTs may cause bad error propagation as highlighted in Section 3.1, since the ancilla qubits are not part of the code block, this can be more easily mitigated. These types of circuits were first studied in the context of syndrome measurements [119, 146–148]. The current standard technique makes use of flag qubits [149, 150], extra ancilla qubits that catch potential errors in the measurement circuit. It is important to note that high weight measurements may be reduced decomposed into a sequence of smaller weight operators [151], making their FT circuits more easily realizable.

After this detour on measurement circuits, we go back to DAs proposed by Davydova et al. They develop a formalism to describe DAs based on the anyon perspective. In fact, there exists a subfield of QEC where topological codes, such as the stabilizer code and the colour code, are seen as anyon models. While an important part of the literature, this approach to QEC is beyond the scope of this manuscript. Instead, we will look at gauge fixing in the stabilizer tableau formalism (subsection 2.3.5) and the ZX-calculus perspective

(chapter 4).

The authors give an example of a FT DA for the colour code based on 2-qubit measurements and give the decoding procedure for it. By using the 3D colour code, which has a transversal non-Clifford gate, they are able to create a DA implementing a non-Clifford gate. In general, DAs are not expected to implement non-Clifford gates because they are procedures represented with the stabilizer formalism, which is in the Clifford fragment, meaning it may be represented as Clifford circuits [15].

That being said, the authors don't offer a general approach to building DAs for other codes. Furthermore, it remains unclear how to systematically build DAs where the logical information is guaranteed to be stay protected throughout the procedure, as it does in the colour code example they give.

Dynamical automorphisms are based on measurements of gauge operators to implement gauge fixing, so one might think they are in the category of measurement-based quantum computing [152]. There exists a mapping between them [37], but these two schemes are very different in spirit. The former uses dynamical logical qubits in order to implement logical gates in a QEC protected way. The latter uses cluster states with $n \times D$ qubits to represent circuits with n physical qubits and D time steps, where the single qubit measurements are consumed to perform the logical operations of the circuit.

Chapter 4

Dynamical Automorphisms via the ZX-Calculus

Brief introduction for the paper and the content that will be discussed more in details

The ZX-calculus is a graphical language for reasoning about quantum circuits, which has recently found many applications in quantum error correction.

Here, we develop a formalism based on the ZX-calculus that enables us to design dynamical automorphisms for stabilizer codes. This method is based on gauge fixing, in which measurements are used to switch between different stabilizer codes. We combine gauge fixing steps to implement a loop in the space of stabilizer codes, returning to the original codespace with a logical Clifford gate applied. We find gauge fixing acts as “chutes and ladders” between different encodings of stabilizer codes, enabling dynamical automorphisms. This work provides three examples of such logical gates, including a distance-preserving phase gate for a code whose code automorphisms generate no non-trivial action. We conclude by suggesting a machine-interpretable method for constructing dynamical automorphisms and facilitates the search for implementations of desired logical gates.

4.1 Gauge Fixing via the ZX-calculus

In this section, we introduce gauge fixing using ZX-encoding graphs. First, we will define gauge fixing. Then, we will show how gauge fixing projects a subsystem code into a stabilizer code. This will lead us to consider the inverse direction, i.e. how to add a gauge qubit to create a subsystem code that gauge fixes into a given stabilizer code. In particular, we will prove that one can find such subsystem codes by carefully adding a gauge input to the ZX-encoding graph. Finally, we will compare gauge fixing via the ZX-calculus to the more familiar stabilizer tableau formalism.

Before defining gauge fixing, we go back to our subsystem code example introduced in subsection 2.5.3:

(4.1)

We consider only a single gauge qubit at a time, but this does not restrict the possible range of gauge fixing. The operators of any two different gauge qubits commute, so fixing the states of the gauge qubits to one of their eigenvectors can be done independently.

We now aim to promote either one of these three gauge operators to become a centre stabilizer, which is performed in experiments by measuring out the desired gauge operator. Indeed we may achieve this gauge fixing via ZX-calculus by simply fixing the gauge input wire in the corresponding eigenstate,

$$\bullet - I = (|+\rangle + |-\rangle) = |0\rangle, \quad \bullet - H = |+\rangle, \quad \bullet - \sqrt{X} = |+i\rangle, \quad (4.2)$$

where we dropped the scalar normalization factors.

We have deliberately chosen to represent each eigenstate in terms of an X spider $\bullet$ —followed by $I, H, \sqrt{X}$ for a reason, which will become clear below. As such, gauge fixing into the corresponding gauge operator becomes in the ZX-calculus formalism:

$$\begin{array}{c} \bullet \\ | \\ \text{---} \\ | \\ \bullet - I \\ | \\ \bullet \end{array}, \quad \begin{array}{c} \bullet \\ | \\ \text{---} \\ | \\ \bullet - H \\ | \\ \bullet \end{array}, \quad \begin{array}{c} \bullet \\ | \\ \text{---} \\ | \\ \bullet - \sqrt{X} \\ | \\ \bullet \end{array}. \tag{4.3}$$

For instance, consider the second gauge fixing, then we obtain the gauge operator $X_{\text{gauge}} = XZZ$ as found in the stabilizer tableau of this subsystem code (2.91):

[illegible]

and similarly for the gauge operator $Z_{\text{gauge}} = IZX$ and $Y_{\text{gauge}} = XIY$. The reason behind this phenomenon is simply that an X spider $\bullet$, freely spawns and absorbs Pauli Z due to the state-copy rule (c). We thus introduce the following definition for gauge fixing in the ZX-calculus:

Definition 14 ([1]). *A gauge input in a subsystem code's ZX-encoding graph is gauge-fixed if its wire is terminated with a $Z/X/Y$ eigenstate, respectively, $\bullet-I$, $\bullet-H$, $\bullet-\sqrt{X}$.*

From this point onward, we will use this definition to refer to gauge fixings in general, unless explicitly stated otherwise.

We point out that, since the states used for gauge fixing are all the $+1$ eigenstates of their corresponding gauge operators, there is some implied post-selection. In practice, if we measure the -1 eigenvalue of the gauge operator, a classical state update may be performed to get back to the $+1$ eigenspace.

4.1.1 From Subsystem Codes to Stabilizer Codes

Let us take a closer look now at these three different gauge fixing and their effect on the ZX-encoding graph when invoking ZX-calculus rewrite rules. We begin with gauge fixing into the Z gauge. For this recall the state-copy rule (c) from section 2.4:

$$\begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \xrightarrow{(c)} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \xrightarrow{(cc)} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} . \quad (4.5)$$

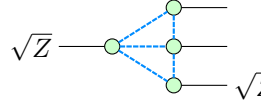
So by fusing the leftover Z spiders, performing a Z gauge fixing effectively cuts the gauge input wire from the ZX-encoding graph of the subsystem code.

$$\begin{array}{c} \bullet-I \\ \vdots \\ \bullet-I \end{array} \xrightarrow{\text{fusing}} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \quad (4.6)$$

If instead we implement the Y gauge fixing by performing a $\sqrt{X}$ before the measurement represented as an X spider, then the $\sqrt{X}$ acts on the encoding graph as a local complementation.

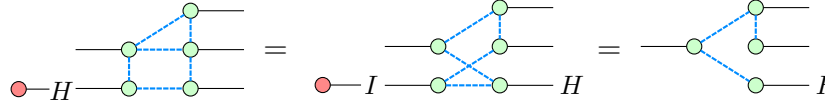
$$\begin{array}{c} \bullet-\sqrt{X} \\ \vdots \\ \bullet-\sqrt{X} \end{array} \xrightarrow{\text{local complementation}} \begin{array}{c} \sqrt{Z} \\ \vdots \\ \sqrt{Z} \end{array} \quad (4.7)$$

The measurement of the gauge qubit is now in the same form as in the Z gauge fixing, but it is attached to a different ZX-encoding graph because some edges have been toggled and some local Clifford gates have appeared too due to the rule (2.72). The gauge can now be cut off, as in the Z gauge fixing, yielding the following ZX-encoding graph.


(4.8)

This graph does not represent the same encoding as the one found after the Z gauge fixing because both encodings are in the normal form (see 2.5.5) and are different.

Finally, the last gauge fixing that can be performed is the X gauge. As mentioned earlier, it is done by applying a Hadamard gate on the gauge qubit node and a measurement in the Z basis. The H gate acts as an edge complementation on the encoding graph. Since this graph is a Clifford encoding circuit, the biadjacency matrix between input nodes, logical and gauge, and physical output nodes is full rank as in proposition 12. This implies a gauge qubit has to share an edge with at least one physical qubit. Hence we may always choose to perform the edge complementation on an edge shared between the gauge qubit and a physical qubit.


(4.9)

The edge complementation produced an Hadamard gate on the third physical qubit, and toggled some edges. Again, this ZX-encoding graph is also in normal form, so we can say with certainty it is not the same as the encodings found by Z or Y gauge fixing.

A gauge qubit may share an edge with more than one physical qubit, so there is ambiguity as to the equivalence of the ZX-encoding graphs after X gauge fixing, given one choice of edge complementation or another. However, all these choices, including edges shared with logical qubits, are exactly equivalent.

Proposition 15. *Let S be a subsystem code that has a gauge qubit and G_S be the ZX-encoding graph of S . Suppose an X gauge fixing is performed via ZX-calculus on G_S as in definition 14. Then, the Hadamard gate can be applied to the graph by performing an edge complementation around any edge connected to the gauge qubit, and will always yield a ZX-encoding graph representing the same stabilizer code.*

Proof. Recall the gauge fixing procedure as a ZX diagram, where the gauge qubit in G_S is measured in the X eigenbasis. One can find an alternative subsystem G'_S including the

Hadamard gate, as in the brackets in the example below.

$$\begin{array}{c} \text{red circle} - H - \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \text{green circle} \\ \text{green circle} \\ \text{green circle} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} = \left[\begin{array}{c} \text{red circle} - H - \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \text{green circle} \\ \text{green circle} \\ \text{green circle} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \right] \quad (4.10)$$

Then, one can choose to perform the Hadamard on any edge to displace the gate from the gauge input wire to another input or output wire. However, all of these choices will have the same normal form, where the H will be placed according to some ordering. Note we can always choose the ordering to avoid placing the H on the gauge qubit as discussed in section 2.5.5. Because the normal form is unique, then all of the ZX-diagrams resulting from each possible choice of edge represent exactly the same isometry, i.e. the same subsystem code.

In each of these diagrams, the gauge is ready to be cut off by the measurement in the Z eigenbasis since we got rid of the H on its wire. Because all the ZX-encoding graphs represent the same subsystem code, then measuring its Z gauge will generate the same stabilizer code. As a consequence, the different ZX-encoding graphs following the measurement will have the same normal form too. $\square$

Given the ZX-encoding graph of a subsystem, three gauge fixings give three ZX-encoding graphs, which represent three stabilizer codes. One may now ask for the reverse: given a stabilizer code as a ZX-encoding graph, how may one find a subsystem code that Z , Y or X gauge fixes into it?

4.1.2 From Stabilizer Codes to Subsystem codes

In order to find a subsystem code from the ZX-encoding graph of a given stabilizer code, we look back to the Z gauge fixing example, and more precisely at the equality in equation (4.6) between the subsystem code before and after cutting off the gauge. One may notice how the Z gauge fixing did not change the connectivity of the logical and physical qubits. By reading this equality from right to left, it becomes clear that building the ZX-encoding graph of *some* subsystem Z gauge fixing into the stabilizer code C from the ZX-encoding graph of C requires to simply add a gauge qubit with some connectivity to the logical and physical qubits. This begs the question: how are we allowed to add the gauge qubit to the graph to make a valid subsystem code?

Definition 16. We define extending a ZX-encoding graph G as adding an input gauge qubit that must be connected to at least one physical qubit, and must not have the same connectivity as that of another combination of logical operators in G .

The requirements on the gauge qubit's connectivity come from the fact that the subsystem code is an isometry, so its biadjacency matrix must be full rank (see Proposition 12). Because the rows of the matrix represent exactly the connectivity between the input and the output of the ZX-encoding graph, the constraints are equivalent to stating a linear combination of other rows is not allowed in a full rank matrix.

We remark that, in the normal form of ZX-encoding graphs, the input qubits can have both I or $\sqrt{Z}$, but in our definition, we have only included I on the gauge input. The extended graph with I or $\sqrt{Z}$ represents two different encodings of the same subsystem code. More precisely, the $\sqrt{Z}$ gate only exchanges the X and Y gauge operators. Because we are creating this degree of freedom, we can always relabel the gauge operators in order to only consider I on the gauge input. Also, this choice is consistent with the gauge fixing definition 14.

Using definition 16, we generalize the construction of the subsystem ZX-encoding graphs that gauge fix into a given ZX-encoding graph of a stabilizer code.

Theorem 17 ([1]). *Let C_{stab} be a stabilizer code, and G_{stab} be its ZX-encoding graph. The ZX-encoding graph of any subsystem code C_{sub} that gauge fixes into C_{stab} can be built by extending G_{stab} and adding a local Clifford gate I (resp. $H, \sqrt{X}$) to the gauge input. The resulting ZX-encoding graph G_{sub} Z (resp. X, Y) gauge fixes into G_{stab} .*

Proof. We begin by proving the statement for any G_{stab} in normal form into which G_{sub} , also in normal form, Z gauge fixes. Then, the statement for general G_{stab} & G_{sub} that Z , X or Y gauge fixes into the former follows.

Any stabilizer operation can be written as a ZX-diagram in the Clifford fragment [72], including all encoding maps. So, for any C_{stab} and C_{sub} , there exists a ZX-diagram corresponding to their encoding procedure, and it can be transformed into its normal form through graph-theoretic simplifications [107] followed by the algorithm presented in subsection 2.5.5.

By choosing the appropriate qubit ordering of the normal form, we may always assume that a gate on the gauge qubit input in G_{sub} must be I or $\sqrt{Z}$ as seen in subsection 2.5.5.

The Z gauge fixing procedure $\bullet - I$ uses the state-copy rule of the ZX-calculus, where the Z spider can have any phase possible. Since $I = -\circ-$ and $\sqrt{Z} = -\textcircled{\frac{\pi}{2}}-$ are Z spiders with a phase in the ZX-calculus, the Z gauge fixing is always possible on G_{sub} . It will always cut off the gauge qubit from the rest of the graph without any further graph or local Clifford gate modifications. Because both G_{stab} and G_{sub} are in normal form, which are unique, and by construction C_{sub} Z gauge fixes into C_{stab} , G_{sub} after the Z gauge fixing

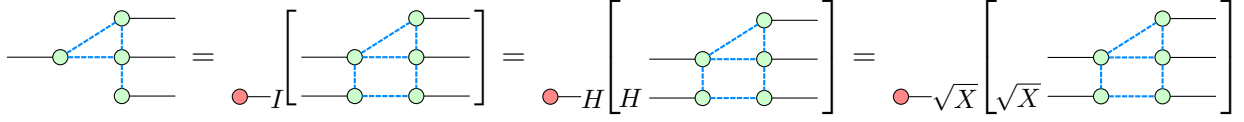


Figure 4.1: ZX-encoding graphs of subsystem codes that gauge fix into a stabilizer code. From left to right, we have the ZX-encoding graph of a stabilizer code and, in brackets, the subsystem codes that Z/X/Y gauge fix into the stabilizer code.

must be exactly equal to G_{stab} . We conclude G_{sub} in normal form is built by *extending* G_{stab} in normal form, i.e. adding a gauge input.

We now generalize this to all ZX-encoding graphs G_{stab} by observing that G_{stab} in normal form can be transformed into a given G_{stab} by applying LCs on the input and output vertices. These same LCs can necessarily be applied to G_{sub} in the normal form without changing the gauge fixing procedure because none of them will be applied to the gauge qubit. To illustrate this, we refer to our recurring example:

$$\begin{array}{c}
 \text{---} O_\ell \text{---} \text{---} O_1 \\
 \text{---} \text{---} O_2 \\
 \text{---} I \text{---} \text{---} O_3
 \end{array}
 =
 \begin{array}{c}
 \text{---} O_\ell \text{---} \text{---} O_1 \\
 \text{---} \text{---} O_2 \\
 \text{---} \text{---} O_3
 \end{array}, \quad (4.11)$$

where $\{O_i\}$ are for some LC gates. Thus, the resulting G_{sub} will Z gauge fix into G_{stab} as it did before the LCs.

For the X and Y gauge fixings, we first recall that $H \cdot H = I = \sqrt{X} \cdot \sqrt{X}$. Then, these Clifford gates squared may be spawned on the gauge input of a G_{sub} that Z gauge fixes into G_{stab} , as in figure 4.1. As illustrated by the brackets, we consider the inner local Clifford gate to be part of the ZX-encoding. Notice how the local Clifford gate leftover on the gauge wire transforms the initial Z gauge fixing into the X and Y gauge fixings of definition 14. We conclude that the ZX-encoding graphs of the subsystem codes that X or Y gauge fix into C_{stab} are the extended G_{stab} with H or $\sqrt{Z}$ respectively added on the gauge input. Considering there are as many subsystems that Z as one that X or Y gauge fix into C_{stab} , we can find the ZX-encoding graph of all subsystems that gauge fix into C_{stab} using this procedure. $\square$

This theorem allows us to find subsystems that Z/X/Y gauge fix into a stabilizer code using the ZX-encoding graphs. We refer the reader to the next subsection 4.1.3 for an alternate proof using a counting argument.

Knowing theorem 17 and given a ZX-encoding graph, we can find any subsystems that gauge fix into it by extending the graph and adding $I/H/\sqrt{X}$ to the gauge input wire. So,

one can devise a measurement procedure to go from one stabilizer code into another by constructing a particular subsystem code from the initial encoding graph and gauge fixing into a new one.

4.1.3 From Stabilizer Codes to Subsystem Codes: A Counting Argument

The proof of Theorem 17 relies on the normal form of ZX-encoding graph, behind which lies a counting argument. Here, we highlight this counting argument by providing another proof of theorem 17. We compare the ZX-encoding graphs to the stabilizer tableaux.

Theorem 18 ([1]: A Counting Argument). *Let C_{stab} be a stabilizer code, and G_{stab} be its ZX-encoding graph. The ZX-encoding graph of any subsystem code C_{sub} that gauge fixes into C_{stab} can be built by extending G_{stab} and adding a local Clifford gate I (resp. $H, \sqrt{X}$) to the gauge input. The resulting ZX-encoding graph G_{sub} Z (resp. X, Y) gauge fixes into G_{stab} .*

Proof. We begin by proving the statement for any G_{stab} into which G_{sub} Z gauge fixes by a counting argument.

The proposition states it is possible to build the ZX encoding graph for any subsystem code S which Z gauge fixes into a given stabilizer code C from the ZX encoding graph of C by extending it as in definition 16. The counting argument will compare the number of possible ZX encoding graphs for S built from the ZX encoding graph of C that yields a valid subsystem code Z gauge fixing into the C , and the number of possible subsystems that Z gauge fixing into C found using the stabilizer tableau formalism.

First, we count the number of valid ZX encoding graphs of S . When extending G_C by adding a gauge qubit to the ZX encoding graph of C , we leave the encoding connectivity in C intact. This guarantees that the Z gauge fixing, which simply removes the gauge qubit and all its edges, as explained in section 4.1, will give back G_C . In addition, we need to add connections to the logical and the physical qubits. All the possible connectivities with the logical qubits are simply all possible subsets of the set of k logical qubits. This sums up to 2^k .

When counting all possible connectivities to the physical qubits, one must not forget the constraints of definition 16 to ensure the encoding graph yields a valid subsystem. We start by counting all possible subsets of the n physical qubits, which gives 2^n . However, we need to subtract the subsets that are the same as the connectivity of a logical or any

combination of logical operators in order to have a valid isometry. Luckily, G_C is already a valid isometry, implying its biadjacency matrix is full rank (see Proposition 12). So we need to remove exactly 2^k subsets from the total number of subsets. To conclude, we multiply all the possible gauge qubit connectivities to the logical and to the physical qubits, and this amounts to $2^k(2^n - 2^k)$.

We proceed to count the number of subsystem codes S which Z gauge fix into C by counting all possible choices of logical operators Z_g and X_g in the stabilizer tableau formalism. Given the stabilizer code C , one can build its stabilizer tableau, including destabilizers introduced in subsection 2.3.5,

$$T_C = \left\langle \begin{array}{c|c} s_1 & \dots & s_m \\ \hline d(s_1) & \dots & d(s_m) \end{array} \parallel \begin{array}{c|c} X_{\ell_1} & \dots & X_{\ell_k} \\ \hline Z_{\ell_1} & \dots & Z_{\ell_k} \end{array} \right\rangle \quad (4.12)$$

where $m = n - k$ and $d(s_i)$ is a destabilizer of s_i .

To get a subsystem code which Z gauge fixes into C , one chooses a stabilizer and promotes it to become Z_g . It is important to note we may choose to promote a stabilizer to Z_g directly as written or a product of any combination of stabilizers. Therefore, there are $2^m - 1 = 2^{n-k} - 1$ possible choices of Z_g , where we removed the identity string, as it trivially is not a valid choice of logical operator. Without loss of generality, we can choose to promote s_m as an example because we can freely relabel stabilizers and multiply them to adjust the tableau to this choice.

Having $Z_g = s_m$, we now need to find all the valid choices of X_g . It must anticommute with s_m , but commute with all the $m - 1$ stabilizers still in the centre of the code. The logical operators $\{X_{\ell_i}, Z_{\ell_i}\}$ do not impose constraints because if one does not commute with X_g , it can be multiplied by Z_g , which was a stabilizer before, thus making that logical representative commute with X_g .

The strings in the tableau form a complete basis for $\mathcal{P}_n$, so we can use them to build X_g :

$$X_g \in (s_1^{a_1} \dots s_m^{a_m}) \cdot (d(s_1)^{b_1} \dots d(s_m)^{b_m}) \cdot (X_{\ell_1}^{c_1} \dots X_{\ell_k}^{c_k}) \cdot (Z_{\ell_1}^{d_1} \dots Z_{\ell_k}^{d_k}), \quad (4.13)$$

where $a = a_1 \dots a_m$, $b = b_1 \dots b_m$, $c = c_1 \dots c_k$, and $d = d_1 \dots d_k$ are bit strings. The first requirement $[X_g, s_1] = 0$ means $b_1 = 0$, eliminating all the Paulis with $b_1 = 1$, i.e. half of them. The second requirement $[X_g, s_2] = 0$ also requires $b_2 = 0$ eliminating half the Pauli strings, and so on until $[X_g, s_{m-1}] = 0$. The last requirement is different because $s_m = Z_g$, so $\{X_g, s_m\} = 0$, but this simply imposes $b_m = 1$ also eliminating half the Pauli strings. Originally, there were $4^n = 2^{2n}$ available choices of X_g , but now we have only $2^{2n}/2^m = 2^{2n}/2^{n-k} = 2^{n+k}$.

There are still too many options of X_g because logical operators are the same up to stabilizers. Also, choosing X_g fixes Y_g . This is to say any Pauli string equal to a given X_g up to $Z_g = s_m$ and the stabilizers $\{s_1 \dots s_{m-1}\}$ will not generate a new subsystem code. The number of these degeneracies is exactly the number of subsets of $\{s_1 \dots s_m\}$. Thus, the adjustment is simply to divide by $2^m = 2^{n-k}$, and the total number of possible X_g for a choice of Z_g is $2^{n+k}/2^{n-k} = 2^{2k}$.

Now, we can put all the pieces together: the number of possible subsystem codes that Z gauge fix into a stabilizer code is the number of possible choices of Z_g $2^{n-k} - 1$ multiplied by the number of possible X_g given Z_g 2^{2k} , and this is $2^{2k}(2^{n-k} - 1) = 2^k(2^n - 2^k)$. This exactly coincides with what we found by counting all graphs earlier, so we conclude that it is possible to find all the G_{sub} which Z gauge fix into G_{stab} may be found by extending G_{stab} with a I gate on the gauge qubit.

The rest of the proof, justifying that the previous result implies that any G_{sub} which gauge fixes into G_{stab} may be built by extending G_{stab} , goes the same as for the proof of Theorem 17: we may spawn $H \cdot H(\sqrt{X} \cdot \sqrt{X})$ from the I on the gauge qubit, thus transforming the Z eigenstate $\bullet - I$ into a X eigenstate $\bullet - H(Y \text{ eigenstate } \bullet - \sqrt{X})$. So, the resulting extended subsystem code $X(Y)$ gauge fixes into the original ZX -encoding graph. Since there are as many subsystem codes that $Z/X/Y$ gauge fix into a given ZX -encoding graph, we may build all such subsystem codes by extending the ZX -encoding graph of a stabilizer code. $\square$

4.1.4 Comparison with Stabilizer Tableau Formalism

Before delving into the applications of gauge measurements to go from one stabilizer code to another using the ZX -calculus formalism, we compare the gauge fixing via ZX -calculus to a familiar formalism, the stabilizer tableaux. More precisely, we focus on the Y gauge fixing presented in equations 4.7 & 4.8. This example illustrates how the graphical updates of the ZX -encoding graphs mirror the operator updates in the stabilizer tableaux.

First, we go back to the stabilizer tableau of the original encoding:

$$\begin{array}{c} \text{---} \bullet \text{---} \bullet \text{---} \bullet \\ \text{---} \bullet \text{---} \bullet \text{---} \bullet \\ \text{---} \bullet \text{---} \bullet \text{---} \bullet \end{array} \rightarrow T^{(1)} = \left\langle YYZ, IZX \left\| \begin{array}{l} \frac{Z_\ell^{(1)} = XZI}{X_\ell^{(1)} = ZZI} \end{array} \right. \right\rangle. \quad (4.14)$$

We also build the stabilizer tableau of the subsystem code used for the gauge fixing:

$$\begin{array}{c} \text{---} \bullet \text{---} \bullet \text{---} \bullet \\ \text{---} \bullet \text{---} \bullet \text{---} \bullet \\ \text{---} \bullet \text{---} \bullet \text{---} \bullet \end{array} \rightarrow T^{(\text{subs})} = \left\langle YYZ \left\| \begin{array}{l} \frac{Z_g = IZX}{Y_g = XIY} \end{array} \right\| \begin{array}{l} \frac{Z'_\ell = XZI}{X'_\ell = ZIX} \end{array} \right\rangle. \quad (4.15)$$

One can check the stabilizer tableau built from the same graph without adjusting the vertex stabilizers due to the presence of the $\sqrt{Z}$ on the logical and physical qubits will not be equivalent to $T^{(2)}$ up to the stabilizers. Similarly in the ZX-calculus perspective, we know removing local Clifford gates on the input or output wires of a ZX-encoding graph will not generate a graph that has the same normal form as the past one. Moreover, the resulting ZX-encoding graph represents a different encoding of the same stabilizer group. So the local Clifford gates generated by the local complementation during gauge fixing via ZX-calculus cannot be erased by updating the qubit basis if one cares to differentiate between encodings of a stabilizer code.

An advantage of the ZX-encoding graph perspective becomes noticeable in this example: one gets a more intuitive understanding of the difference between two stabilizer codes by comparing their encoding graphs in normal form than by looking at lists of generators of the stabilizer group. Therefore, the effect of the gauge fixing on the stabilizer code is more digestible using ZX-encoding graphs. Another benefit comes in when building subsystem codes from stabilizer codes. By extending the ZX-encoding graph of the stabilizer code, the bookkeeping updates of the logical operators come for free. Finally, we point out that graphs can be stored very compactly using the Nauty package [154], allowing efficient graph manipulations through bitwise computations. This package also computes canonical forms for graphs, making it easier to find the normal form of a ZX-encoding graph.

One can also directly see if the gauge fixing measurement itself performs a logical action because a $\sqrt{Z}$ or an H will appear on the logical input wire. For the moment, this doesn't represent a true logical gate because the encoding graph has also changed. Nevertheless, one can imagine that by performing a series of gauge fixings and local Clifford gates on the physical qubits, the encoding graph can return to its original self, with the exception of the accumulated gates on the logical input wire. As such, this whole procedure would generate a logical gate on the stabilizer code. This idea will be explored in section 4.2.

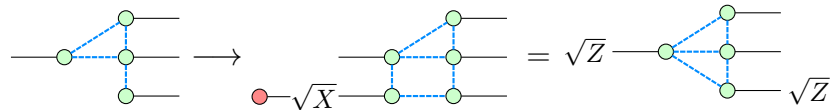
4.2 Dynamical Automorphisms and ZX-encoding graphs

As seen in the previous section, gauge fixing can be leveraged to transfer the logical information from one stabilizer code to another. In fact, logical gates such as lattice surgery and code switching can be understood as gauge fixing [99, 155]. As a reminder, dynamical automorphism gates [38] can be understood as a generalization of code switching where we perform local Clifford gates, qubit permutations and gauge fixings from one stabilizer to another until we return to the original code, up to a logical gate. In other words, dynamical automorphisms are produced by nontrivial loops in the space of $[[n, k]]$ stabilizer codes. Since we know that any two codes can be connected with high probability in a distance-preserving way via code switching [156, theorem 1] (see also [145, section 2.6] and [155]), this space is rich and can be used as a resource.

As an introduction, we begin with a motivating example using a $[[3, 1]]$ code. This example illustrates how one can achieve a logical action using a single gauge fixing. Then, we look at a gauge fixing that entangles logical qubits in the $[[4, 2, 2]]$ code. We move on to a more meaningful example with the $[[7, 1, 3]]$ code, where the gauge fixings are distance-preserving and the logical gate is not generated by its code automorphism group. We end this section by generalizing the idea to logical 2-qubit Clifford gates, and suggesting a way to find a gauge fixing procedure generating a given Clifford logical gate.

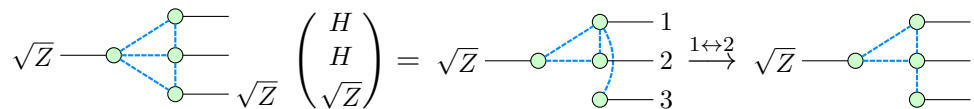
4.2.1 A Simple Nontrivial Logical Action

In section 4.1, we concluded with the intuition that the gates generated by the gauge fixing on the logical input wire could lead to a true logical gate by looping back to the original encoding. Let's finish that thought. Recall our recurring ZX-encoding graph and the Y gauge fixing.



$$(4.17)$$

The resulting stabilizer code can be transformed back into the original ZX-encoding graph through LC gates and qubit permutations.



$$(4.18)$$

The last encoding is the same as the original, but with an additional $\sqrt{Z}$ gate on the logical input. From this observation, we conclude the Y gauge fixing, the LC and the final SWAP perform logical $\sqrt{Z}$.

To get a nontrivial action, one has to do a nontrivial loop through the space of stabilizer codes using gauge fixing. Hence, it may be surprising only one gauge fixing was necessary to achieve a logical gate. While it is true gauge fixing from a given subsystem code projects you into three stabilizer codes, it may happen two of those are the same up to some local Clifford gates and some qubit permutations, as in the example above. One calls such stabilizer codes LC equivalent, a term borrowed from graph states (see subsection 2.2.4). Strictly speaking, LC equivalence does not include qubit permutation. However, in the following discussions, it will be assumed that permutations are freely available.

4.2.2 LC Orbits

Gauge fixing can be used to navigate through the space of LC equivalent stabilizer codes, but one can also simply use LC gates and SWAPs to generate code automorphisms, as seen in section 3.4. In order to compare the latter to the dynamical code automorphisms, we build the equivalence class of the encodings of a stabilizer code, which are called LC orbits. The LC orbit of a stabilizer code spans through all the different conjugations by LC gates of the stabilizers and logical operators. All the resulting encodings still represent the same stabilizer code, but with a different choice of Pauli basis for the qubits.

Here, we represent the LC orbit as a map of ZX-encoding graphs where neighbours are equivalent up to a single LC gate on either physical or logical qubits. The edges of the map are equality signs to represent the LC equivalence between graphs. The logical gates induced by LC gates for a step in the LC orbit are indicated above the edges $\stackrel{U}{=}$. Because the LC gates are self-inverse, their logical effect is the same when applying the LC gate on either connected ZX-encoding graphs. If there is more than one LC gate that connects the same two ZX-encoding graphs, we write the logical action of each U_i as $\stackrel{U_1/U_2}{=}$. It may also happen an LC gate brings a graph back to itself up to some logical operator, and these instances are illustrated as loops. Finally, we add red arrows to represent the Y and X gauge fixings from the subsystem discussed earlier at the beginning of section 4.1. We do not illustrate the Z gauge fixing, since it brings the subsystem code back to the original encoding. Note that the connections based on gauge fixing are specific to the subsystem we chose to look at, whereas the LC orbit itself is independent of this choice. As a reminder, the Y gauge fixing spawns a $\sqrt{Z}$, whereas the X gauge fixing spawns either a H if the edge

we choose to perform the edge complementation around is shared with the logical or I if not. So, the LC orbit of the 3-qubit code is found in figure 4.2.

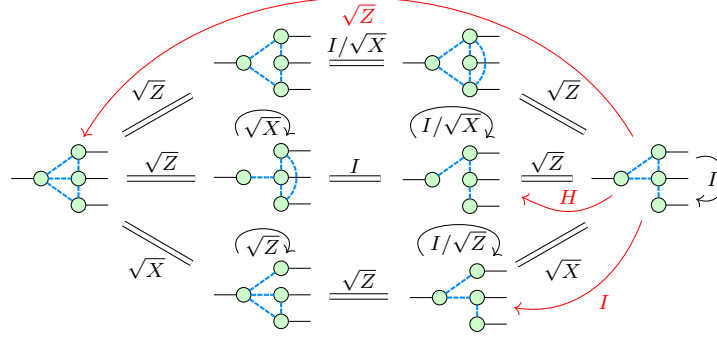


Figure 4.2: Local Clifford (LC) orbit of the 3-qubit motivating code from subsection 4.2.1, whose ZX-encoding graph is on the far right side of the diagram. Here, the equalities between ZX-encoding graphs refer to LC equivalence, whereas the looping arrows represent LC gates that bring the graph to itself. The gates above them keep track of the logical gate induced by the given LC, and we do not keep track of the physical gates they induce. The red arrows correspond to the gauge fixings seen in equations (4.7) & (4.8) for the Y gauge fixing and (4.9) for the X gauge fixing.

We start by looking at the static picture, i.e. the LC orbit without the red arrows representing gauge fixing. One can compute possible static logical gates by taking the composition of all logical gates in a path using only LC equivalences, and starting and ending at a given encoding. In our case, the base ZX-encoding graph is the graph on the far right of the LC orbit. There are three loops that pass by the left extremity of the orbit. Each branch connecting the right most graph to the left most one can either generate a I or an H . Thus, all loops using those branches can also generate either one of these two logical gates. Other ways to generate loops are to use the operations that bring a ZX-encoding graph to itself up to a logical operator or use the two different LC gates $\underline{\underline{U_1/U_2}}$ while going to another graph than while coming back to the original. All such loops represent code automorphisms. One can check that they also only generate I or H . So we conclude that $\{I, H\}$ are the set of logical gates generated by code automorphisms for the 3-qubit motivating code, choosing this ZX-encoding graph as a base encoding.

Instead, if one were to start with a ZX-encoding graph that already had an LC gate bringing the code to itself up to a logical $\sqrt{Z}$ or $\sqrt{X}$, then the three big loops would generate that same logical gate. We invite the reader to check this by picking another a ZX-encoding graph with a nontrivial self-loop as a starting point to compute the big loops.

However, this is to be expected because different encodings will only change the basis of the stabilizers. As such, the order of the group of code automorphisms, its number of generators, and their period are invariant with the choice of encoding.

Now we turn to navigating the LC orbit using the gauge fixings illustrated in red. The Y gauge fixing sends the right most graph code to the left most one. It creates a *shortcut* through the space of encodings of the stabilizer code. The measurement may be followed by LC gates to go back to the right most graph. This path would produce a logical $\sqrt{Z}$ as in subsection 4.2.1 or $\sqrt{Z}H$ depending on the choice of LC gates. These two logical gates expand the automorphism group such that together they generate all LC gates $\mathcal{C}_2^{(1)}$, showing that the ZX-calculus gauge-fixing formalism can generate additional gates beyond generated by the code's automorphism group.

For the X gauge fixing, there seem to be two different graphs into which the same graph code is transformed, thus two different paths. This wouldn't make much sense, since the same gauge operator is being measured in both cases. However, proposition 15 tells us the two ZX-encoding graphs have the same normal form. Therefore they represent the same isometry. Indeed, one can map the ZX-encoding graph above to the lower one by transferring the H gate from the logical qubit onto the first physical qubit via pivoting. A consequence of the equivalence between the two encodings is that both choices can generate the same logical gates: $\sqrt{X} = H\sqrt{X}\sqrt{Z}$ or $\sqrt{Z}\sqrt{X} = H\sqrt{Z}$. It becomes clear the graph codes in the LC orbit are not necessarily unique, implying there are some redundant paths within the LC orbit.

4.2.3 Entangling through Gauge Fixing

The example in subsection 4.2.1 gave a clear example of how one could produce gates $O \in Cl(1)$ by extending a ZX-encoding graph and gauge fixing. We now move on to a $CZ \in Cl(2)$ for the smallest 2-qubit code, the $[[4, 2, 2]]$ code, which has stabilizers $XXXX$ and $ZZZZ$ as seen in the gauge fixing example of subsection 2.3.5. In fact, the DA presented in figure here uses the exact gauge fixing presented there. Its ZX-encoding graph is found in figure 4.3a, while the DA procedure constructed by gauge fixing and LC gates is presented in figure 4.3b.

In ZX-calculus, a CZ gate is a vertical Hadamard edge between two Z spiders (see equation (2.71) in section 2.4). So, by comparing the original ZX-encoding graph in figure 4.3a and the final ZX-encoding graph in figure 4.3b, we may notice the extra edge between the two logical qubits was pulled out the graph. So this procedure does generate a logical CZ gate.

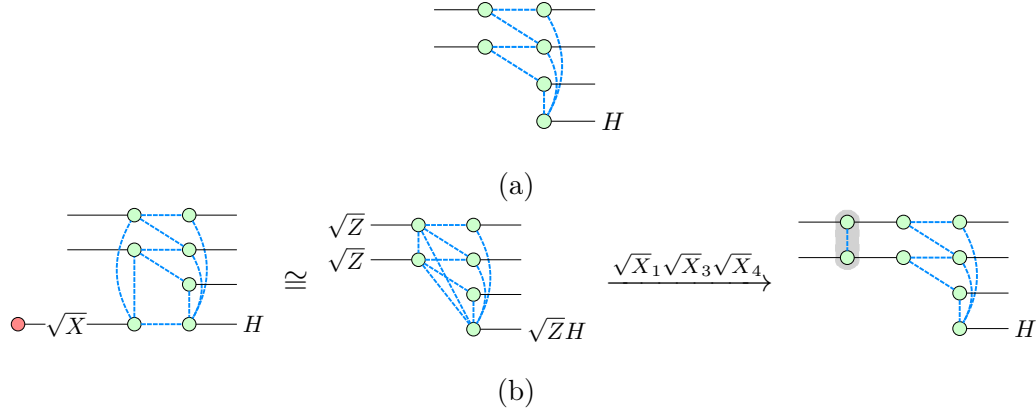


Figure 4.3: Dynamical automorphism of the $[[4, 2, 2]]$ code. We extend the original ZX-encoding graph (a) to construct a subsystem code (b) that Y gauge fixes into the $[[4, 2, 2]]$ code up to some local Clifford gates and a logical CZ gate (highlighted in gray).

Because LC gates allowed us to complete the loop back to the original encoding modulo a gate, we know the ZX-encoding after the gauge fixing belongs in the LC orbit of the $[[4, 2, 2]]$. Thus this DA produces a logical CZ in a distance-preserving way. We point out that, technically, the previous example with the 3-qubit code was also distance-preserving. However, that code had $d = 1$, so it wasn't worth mentioning until now that we have an error-detecting code.

In this particular example, the gauge fixing step may be replaced by a $\sqrt{X_2}$. The CZ gate is in fact transversal for this code, so while not necessary here, this example illustrates how gauge fixing induces entangling operations on logical qubits encoded in the same code block.

4.2.4 Bare Code Example

Lastly, we provide an example of a dynamical automorphism generating a logical S gate for the 7-qubit code known as the bare code [57, 157]. This dynamical automorphism differentiates itself from the ones presented previously because it changes code space. In addition, each code in the sequence has code distance $d = 3$, bringing us one step closer to fault-tolerance. We also computed the code automorphisms of the bare code and found that it has trivial action on the codespace [1]. So the phase gate generated by the distance-preserving dynamical automorphism below wasn't previously accessible to our knowledge. The full dynamical automorphism is shown in figure 4.4.

The procedure was conceived using Cross and Vandeth's map of $[[7, 1, 3]]$ codes, where

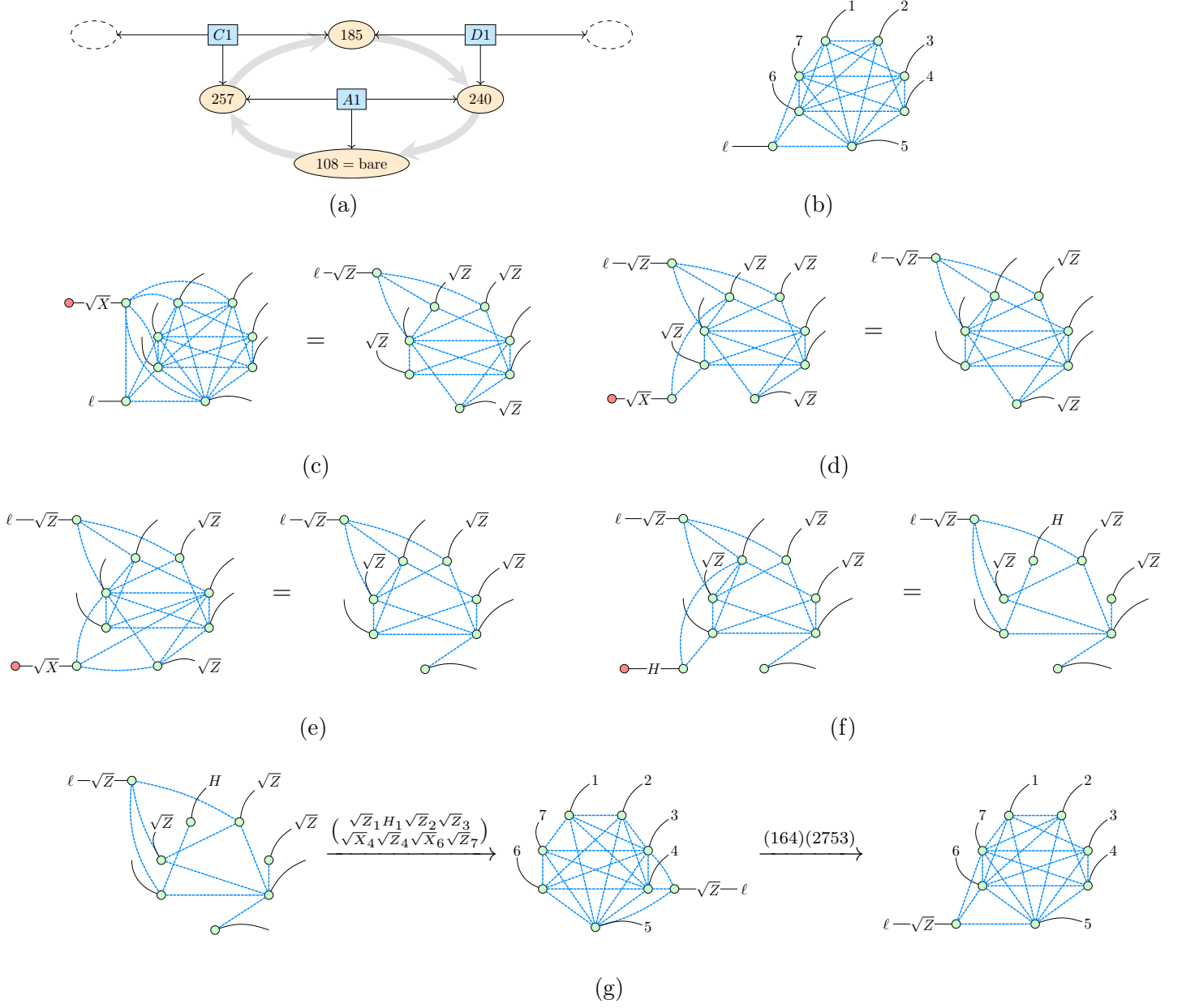
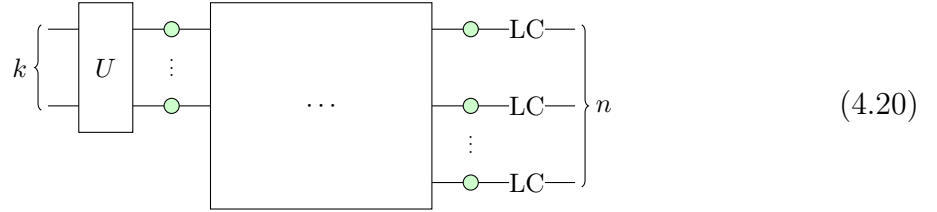


Figure 4.4: Dynamical automorphism for the bare code implementing a logical S gate. (a) Space of $[[7, 1, 3]]$ stabilizer codes, excerpt from [158, Fig. 12]. The yellow squares are subsystem codes S_i that Z/X/Y gauge fix into the stabilizer codes C_i (blue bubbles). (b) ZX-encoding graph of the 7-qubit bare code [157]. We label the logical qubit wire ℓ and the physical qubit wires $1 \dots 7$. (c) Y gauge fixing of S_1 into C_2 . (d) Y gauge fixing of S_2 into C_3 . (e) Y gauge fixing of S_3 into C_4 . (f) X gauge fixing of S_1 into the bare code. (g) Final local Clifford gates and qubit permutations.

4.2.5 Generalization to Logical 2-Qubit Cliffords

So far, we have explored the application of building a subsystem code that gauge fixes into the original stabilizer code. In general, subsystem codes will gauge fix into three different stabilizer codes. We would like to leverage these as bridges in the $[[n, k]]$ stabilizer code space in order to find a systematic method to look for 2-qubit Clifford gates. However, the search space can be immense, especially when adding more qubits, i.e. more vertices in the ZX-encoding graphs, more possible LC gates and more subsystem codes. Here, we suggest a method to optimize the hunt for a specific logical gate and to find the shortest nontrivial loop.

Any k -qubit Clifford $U \in Cl(k)$ can be written as a ZX-diagram in the Clifford fragment [41]. In fact, unitary Cliffords are a subset of ZX-encoding graphs where $n = k$, so one can use the normal form to characterize all $U \in Cl(k)$. We may then append this diagram to the input of a ZX-encoding graph G_C of a code C to find the graph where the gate U has been implemented, which we call $U \circ G_C$. We illustrate the result below, where the blank box represents the ZX-encoding graph connectivity.



If U is a 2-qubit Clifford gate, this graph will now have inner vertices, which cannot be directly fused to other vertices. Graph simplification procedures [107] can remove those in order to find the ZX-encoding graph in normal form. This is exactly the encoding we want to reach through gauge fixing, LC gates and SWAPs. So, to find the path from G_C to $U \circ G_C$, we can conduct a breadth-first search, starting from G_C and $U \circ G_C$ until we find two ZX-encoding graphs that are in the same LC orbit, say the LC orbit of the stabilizer code $\mathcal{S}$. We illustrate this method in figure 4.5.

The diagram illustrates this breadth-first approach, where all the grey bubbles represent the LC orbit of a stabilizer code, and all the arrows represent different gauge fixings using different subsystem codes found by extending the ZX-encoding graphs at each given step. In order to find the meeting point $\mathcal{S}$, we must extend the diagrams in every possible way in order to perform all possible gauge fixings, and thus find all possible routes through the $[[n, k]]$ stabilizer codes space. We point out that we need to concern ourselves only with one ZX-encoding graph for each stabilizer code. The LC gates that map different encodings

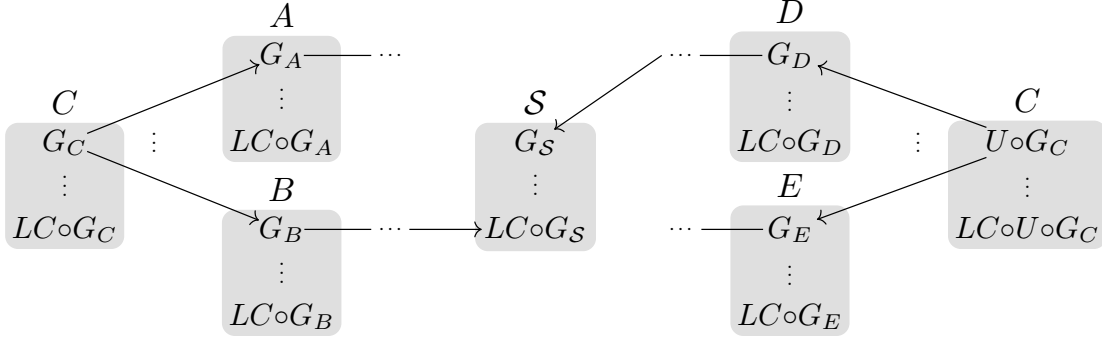


Figure 4.5: Breadth-first search to implement a 2-qubit Clifford gate U . The gray boxes represent the LC orbits of each stabilizer code A , B , C , D , E , & S . On the far left, there is the initial ZX-encoding graph G_C associated with the code C , and on the far right there is the goal ZX-encoding graph $U \circ G_C$. The arrows represent different gauge fixings, performed from both the initial and final ZX-encoding graphs. They eventually meet in the LC orbit of the code S .

to one another may be pushed through the subsystem codes, as explained in subsection 4.2.4. Theorem 17 ensures we can find all the relevant subsystems by extending a single ZX-encoding graph of a given stabilizer code.

Because any gauge fixing is self-inverse, we may use the path from G_C to G_S and the reverse of path $U \circ G_C$ to G_S to complete the desired dynamical code automorphism. However, we must not forget the LC gates that bring $LC \circ G_S$ to G_S in the middle of the gauge fixings. In an experimental setting, it could be more practical to have flexibility as to when we would want to perform LC gates in order to parallelize the procedure. Indeed, this is possible, but one has to update the subsystem codes' gauge fixings, as was explained in subsection 4.2.4. This step finalizes the multi-step gauge fixing procedure, and thus the dynamical automorphism via the ZX-calculus, by bringing the codespace back to itself modulo a logical Clifford gate.

Chapter 5

Discussion and Conclusion

We started this thesis by introducing in chapter 2 some quantum information basics, the stabilizer formalism and the ZX-calculus. First, we defined quantum states, quantum channels and two representations, the Choi representation and the Kraus representation. This then enabled us to lay the bases of the stabilizer formalism. In particular, we introduced Pauli operators, Clifford gates, non-Clifford gates and the symplectic representation, as well as stabilizer states and graph states. The next step was to discuss stabilizer codes. We built up to subsystem codes, gauge fixing and dynamical codes, which are at the centre of this manuscript. However, there was still a big chunk missing: the ZX-calculus. We first defined this graphical language and its connection to graph states before representing stabilizer codes in it using ZX-encoding graphs. We also explored different properties of these objects that were necessary to understand the results like the LC equivalence, their associated stabilizer tableau, and their normal form.

Before delving into the results, we first reviewed some fault-tolerant methods to build logic in stabilizer codes in chapter 3. First, we defined fault-tolerant gates as mapping correctable errors to each other. We also defined transversal gates. Next, we introduced the first FT protocol, magic state distillation. It is a protocol to generate high-fidelity non-Clifford gates using many noisy states to make good magic states. Then, we looked at lattice surgery, a technique to realize fault-tolerant CNOTs in the surface code and QLDPC codes. After two complex protocols, we fell back on code automorphisms, which are a type transversal gates based on LC gates and qubit permutation. Finally, we considered adding measurements as a tool for logic in dynamical automorphisms. In DAs, a gauge fixing sequence projects logical information onto different codespaces until it gets back to the original encoding, generating a logical Clifford gate along the way.

In chapter 4, we discussed the results of this work: dynamical automorphisms via the ZX-calculus. We first introduce gauge fixing in the ZX-calculus. In particular, we define a gauge qubit in ZX-encoding graphs as an input of the graph. Thus, gauge fixing becomes the act of closing off the input with a Z , X or Y eigenstate, represented by a red spider with a different LC gate attached. We then explained how the action of gauge fixing projects the subsystem code into different codespaces, depending on the choice of gauge fixing. After, it became natural to consider building the ZX-encoding graph of a subsystem code from the ZX-encoding graph of the stabilizer code it gauge fixes. This may be achieved by carefully adding a gauge input, and we define it as extending the ZX-encoding graph. We go on to prove that it is in fact possible to build any and all ZX-encoding graphs of subsystem codes which gauge fix into a given ZX-encoding graph by extending it. Our proof relies on the fact the normal form of a ZX-encoding graph is unique. We also give an alternate proof using a counting argument. Finally, we compare gauge fixing via the ZX-calculus to the familiar stabilizer tableau formalism. We thus confirm with an example that these two formalisms for gauge fixing are equivalent.

Having defined gauge fixing via the ZX-calculus, we then used it as a tool to generate logic through dynamical automorphisms. We first explored a simple example with a 3-qubit motivating code, where the logical gate realized was outside the set of gates generated by code automorphisms. This showed that dynamical automorphisms do have an edge over code automorphisms. However, it was a bit peculiar that a single gauge fixing step was sufficient to generate a nontrivial action, considering that dynamical automorphisms are nontrivial loops in the space of stabilizer codes. To understand this, we looked at the LC orbit of the code, and found that the gauge fixing was acting like chutes and ladders between different encodings of the same stabilizer code. So the loop was hidden inside the LC orbit. Then, we moved on to 2-qubit Clifford gates. We gave an example of a dynamical automorphism implementing a logical CZ gate for the $[[4, 2, 2]]$ code. Again, a single gauge fixing was sufficient in this protocol, the making the procedure distance-preserving. Finally, we looked at distance-preserving dynamical automorphism for the $[[7, 1, 3]]$ bare code, where the logical qubit was projected onto different stabilizer codes before coming back to the original space up to a phase gate. Interestingly, this gate was also cannot be generated via the code automorphism group.

Now that we understand better our formalism, we can compare it to the other FT protocols reviewed in chapter 3. First, since our formalism is based on the Clifford fragment of the ZX-calculus, it cannot describe any non-Clifford gates. As such, it is complementary to magic state distillation. However, if we choose our base code to be a triorthogonal code [25] which has a transversal non-Clifford gate, then we would bypass the need for magic state distillation.

Second, we point out that lattice surgery, presented in Section 3.3, and code deformation can be thought of as gauge fixing [99]. So our gauge fixing formalism could be used to describe lattice surgery as well as dynamical automorphisms. Also, we point out that latticed surgery has been described in the ZX-calculus [127]. The results presented in Chapter 4 are thus close in flavour to this prior work. Our work differentiates itself because they look at the procedures on the logical level, whereas we create a gauge fixing formalism, giving measurement schedules for the physical qubits.

Finally, the examples of dynamical automorphisms for the 3-qubit code and the bare code presented in section 4.2 are clear examples that code automorphisms are weaker than dynamical automorphisms. However, the latter require more complex protocols because of the gauge operator measurements. Because our formalism uses compact graph-like ZX-diagrams, it loses information about the physical implementation of these gauge fixings. One could imagine using ZX-calculus rules to extract measurement circuits from the ZX-encoding graphs. It would be interesting to take inspiration from distance-preserving rewrites in the ZX-calculus [54] to obtain distance-preserving measurement circuits.

An important extension of our work is to compute the effect of the gauge fixing steps on the code distance, which may decrease in some cases [99]. To accurately benchmark the performance of our dynamical automorphism gates, we would also need to construct circuits for measuring the gauge operators and include rounds of stabilizer measurement. Before reaching these next steps, we do not have enough information to compare FT protocols based on dynamical automorphisms to magic state distillation or lattice surgery.

One of the strong suits of the formalism is that ZX-encoding graphs can be stored bit wise using the Nauty package for graphs [154]. Then, local complementations and thus LC gates correspond to bit wise operations, which are very efficient. This would make the breadth-first scheme discussed in subsection 4.2.5 also potentially efficient. One issue that might arise is the data processing. As the number of physical and logical qubits grows, the bigger the space of $[[n, k]]$ stabilizer codes get. So managing the data from all the LC orbits of the codes and all the subsystems may be hard in practice.

Also, the formalism is very intuitive: it is straightforward to see when a logical gate appears after some gauge fixings. So when considering bigger codes, it would be interesting to see if an AI model could also understand these diagrams intuitively [161]. It also may be able detect patterns to predict when gauge fixings are distance-preserving, helping to direct the search towards fault-tolerant dynamical automorphisms.

References

1. **Frei, A., Zakaib-Bernier, S., Mann, Z., Vasmer, M. & Albert, V. V.** *Chutes and Ladders: Dynamical Automorphisms via the ZX-Calculus* 2026. arXiv: [2606.02542](https://arxiv.org/abs/2606.02542) [quant-ph]. <https://arxiv.org/abs/2606.02542>.
2. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information* 10th anniversary ed. ISBN: 978-1-107-00217-3 (Cambridge University Press, Cambridge ; New York, 2010).
3. Ofek, N. *et al.* Extending the lifetime of a quantum bit with error correction in superconducting circuits. *Nature* **536**, 441–445. ISSN: 1476-4687. <https://doi.org/10.1038/nature18949> (Aug. 2016).
4. Krinner, S. *et al.* Realizing repeated quantum error correction in a distance-three surface code. *Nature* **605**, 669–674. ISSN: 1476-4687. <https://doi.org/10.1038/s41586-022-04566-8> (May 2022).
5. Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* **614**, 676–681. ISSN: 1476-4687. <http://dx.doi.org/10.1038/s41586-022-05434-1> (Feb. 2023).
6. Google Quantum AI and Collaborators. Quantum Error Correction below the Surface Code Threshold. *Nature* **638**, 920–926. ISSN: 1476-4687 (Feb. 2025).
7. Lacroix, N., Bourassa, A., Heras, F. J. H., *et al.* Scaling and Logic in the Colour Code on a Superconducting Quantum Processor. *Nature* **645**, 614–619. ISSN: 1476-4687 (May 2025).
8. Ryan-Anderson, C. *et al.* Realization of Real-Time Fault-Tolerant Quantum Error Correction. *Physical Review X* **11**, 041058. ISSN: 2160-3308 (Dec. 2021).
9. Postler, L. *et al.* Demonstration of Fault-Tolerant Steane Quantum Error Correction. *PRX Quantum* **5**, 030326. <https://link.aps.org/doi/10.1103/PRXQuantum.5.030326> (3 Aug. 2024).

10. Paetznick, A. *et al.* *Demonstration of logical qubits and repeated error correction with better-than-physical error rates* 2024. arXiv: [2404.02280](https://arxiv.org/abs/2404.02280) [quant-ph]. <https://arxiv.org/abs/2404.02280>.
11. Reichardt, B. W. *et al.* *Demonstration of Quantum Computation and Error Correction with a Tesseract Code* 2024. arXiv: [2409.04628](https://arxiv.org/abs/2409.04628) [quant-ph]. <https://arxiv.org/abs/2409.04628>.
12. Ryan-Anderson, C. *et al.* *High-Fidelity and Fault-tolerant Teleportation of a Logical Qubit Using Transversal Gates and Lattice Surgery on a Trapped-ion Quantum Computer* 2024. arXiv: [2404.16728](https://arxiv.org/abs/2404.16728) [quant-ph]. <https://arxiv.org/abs/2404.16728>.
13. Muniz, J. A. *et al.* Repeated Ancilla Reuse for Logical Computation on a Neutral Atom Quantum Computer. *Phys. Rev. X* **15**, 041040. <https://link.aps.org/doi/10.1103/v7ny-fg31> (4 Dec. 2025).
14. Bluvstein, D. *et al.* *Architectural mechanisms of a universal fault-tolerant quantum computer* 2025. arXiv: [2506.20661](https://arxiv.org/abs/2506.20661) [quant-ph]. <https://arxiv.org/abs/2506.20661>.
15. Gottesman, D. *Stabilizer Codes and Quantum Error Correction* 1997. arXiv: [quant-ph/9705052](https://arxiv.org/abs/quant-ph/9705052) [quant-ph]. <https://arxiv.org/abs/quant-ph/9705052>.
16. Eastin, B. & Knill, E. Restrictions on Transversal Encoded Quantum Gate Sets. *Physical Review Letters* **102**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.102.110502> (Mar. 2009).
17. Horsman, D., Fowler, A. G., Devitt, S. & Meter, R. V. Surface code quantum computing by lattice surgery. *New Journal of Physics* **14**, 123011. ISSN: 1367-2630. <http://dx.doi.org/10.1088/1367-2630/14/12/123011> (Dec. 2012).
18. Litinski, D. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum* **3**, 128. ISSN: 2521-327X. <https://doi.org/10.22331/q-2019-03-05-128> (Mar. 2019).
19. Cohen, L. Z., Kim, I. H., Bartlett, S. D. & Brown, B. J. Low-Overhead Fault-Tolerant Quantum Computing Using Long-Range Connectivity. *Science Advances* **8**, eabn1717. ISSN: 2375-2548 (2022).
20. Ide, B., Gowda, M. G., Nadkarni, P. J. & Dauphinais, G. Fault-Tolerant Logical Measurements via Homological Measurement. *Physical Review X* **15**, 021088. ISSN: 2160-3308 (June 2025).
21. Swaroop, E., Jochym-O'Connor, T. & Yoder, T. J. Universal Adapters between Quantum LDPC Codes. *arXiv preprint* (Oct. 2024).

22. He, Z., Cowtan, A., Williamson, D. J. & Yoder, T. J. Extractors: QLDPC Architectures for Efficient Pauli-Based Computation. *arXiv preprint* (Mar. 2025).
23. Knill, E. Fault-Tolerant Postselected Quantum Computation: Schemes. *arXiv preprint* (Feb. 2004).
24. Bravyi, S. & Kitaev, A. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A* **71**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.71.022316> (Feb. 2005).
25. Bravyi, S. & Haah, J. Magic-state distillation with low overhead. *Physical Review A* **86**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.86.052329> (Nov. 2012).
26. Haah, J. & Hastings, M. B. Codes and Protocols for Distilling T , controlled- S , and Toffoli Gates. *Quantum* **2**, 71. ISSN: 2521-327X. <https://doi.org/10.22331/q-2018-06-07-71> (June 2018).
27. Litinski, D. Magic State Distillation: Not as Costly as You Think. *Quantum* **3**, 205. ISSN: 2521-327X. <https://doi.org/10.22331/q-2019-12-02-205> (Dec. 2019).
28. Paetzniak, A. & Reichardt, B. W. Universal Fault-Tolerant Quantum Computation with Only Transversal Gates and Error Correction. *Physical Review Letters* **111**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.111.090505> (Aug. 2013).
29. Bombin, H. *Gauge Color Codes: Optimal Transversal Gates and Gauge Fixing in Topological Stabilizer Codes* 2015. arXiv: [1311.0879](https://arxiv.org/abs/1311.0879) [quant-ph]. <https://arxiv.org/abs/1311.0879>.
30. Kubica, A. & Beverland, M. E. Universal transversal gates with color codes: A simplified approach. *Physical Review A* **91**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.91.032330> (Mar. 2015).
31. Bombin, H. *Dimensional Jump in Quantum Error Correction* 2016. arXiv: [1412.5079](https://arxiv.org/abs/1412.5079) [quant-ph]. <https://arxiv.org/abs/1412.5079>.
32. Beverland, M. E., Kubica, A. & Svore, K. M. Cost of Universality: A Comparative Study of the Overhead of State Distillation and Code Switching with Color Codes. *PRX Quantum* **2**, 020341. <https://link.aps.org/doi/10.1103/PRXQuantum.2.020341> (2 June 2021).
33. Hastings, M. B. & Haah, J. Dynamically Generated Logical Qubits. *Quantum* **5**, 564. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2021-10-19-564> (Oct. 2021).

34. Davydova, M., Tantivasadakarn, N. & Balasubramanian, S. Floquet Codes without Parent Subsystem Codes. *PRX Quantum* **4**. ISSN: 2691-3399. <http://dx.doi.org/10.1103/PRXQuantum.4.020341> (June 2023).
35. Kesselring, M. S. *et al.* Anyon Condensation and the Color Code. *PRX Quantum* **5**, 010342. <https://link.aps.org/doi/10.1103/PRXQuantum.5.010342> (1 Mar. 2024).
36. Higgott, O. & Breuckmann, N. P. Constructions and Performance of Hyperbolic and Semi-Hyperbolic Floquet Codes. *PRX Quantum* **5**, 040327. <https://link.aps.org/doi/10.1103/PRXQuantum.5.040327> (4 Nov. 2024).
37. Bombin, H., Litinski, D., Nickerson, N., Pastawski, F. & Roberts, S. Unifying flavors of fault tolerance with the ZX calculus. *Quantum* **8**, 1379. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2024-06-18-1379> (June 2024).
38. Davydova, M., Tantivasadakarn, N., Balasubramanian, S. & Aasen, D. Quantum computation from dynamic automorphism codes. *Quantum* **8**, 1448. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2024-08-27-1448> (Aug. 2024).
39. Coecke, B. & Duncan, R. *Interacting Quantum Observables in Automata, Languages and Programming* (eds Aceto, L. *et al.*) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), 298–310. ISBN: 978-3-540-70583-3.
40. Coecke, B. & Duncan, R. Interacting Quantum Observables: Categorical Algebra and Diagrammatics. *New Journal of Physics* **13**, 043016 (2011).
41. Kissinger, A. & van de Wetering, J. *Picturing Quantum Software: An Introduction to the ZX-Calculus and Quantum Compilation* <https://github.com/zxcalc/book> (Book preprint, 2024).
42. Kissinger, A. Phase-free ZX diagrams are CSS codes (...or how to graphically grok the surface code). *arXiv preprint* (2022).
43. Khesin, A. B. & Li, A. *Equivalence Classes of Quantum Error-Correcting Codes* 2024. arXiv: [2406.12083](https://arxiv.org/abs/2406.12083) [quant-ph]. <https://arxiv.org/abs/2406.12083>.
44. Khesin, A. B., Lu, J. Z. & Shor, P. W. Universal graph representation of stabilizer codes. *arXiv preprint* (2025).
45. Wu, Z., Cheng, S. & Zeng, B. A ZX-Calculus Approach for the Construction of Graph Codes. *arXiv preprint* (2024).
46. De Beaudrap, N. & Horsman, D. The ZX calculus is a language for surface code lattice surgery. *Quantum* **4**, 218. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2020-01-09-218> (Jan. 2020).

47. Kupper, M., Horsman, D., Heunen, C. & de Beaudrap, N. String Diagrams for Defect-Based Surface Code Computing. *Electronic Proceedings in Theoretical Computer Science* **426**, 159–196. ISSN: 2075-2180. <http://dx.doi.org/10.4204/EPTCS.426.6> (Aug. 2025).
48. Bauer, A. & de la Fuente, J. C. M. *Planar fault-tolerant circuits for non-Clifford gates on the 2D color code* 2025. arXiv: [2505.05175](https://arxiv.org/abs/2505.05175) [quant-ph]. <https://arxiv.org/abs/2505.05175>.
49. Magdalena de la Fuente, J. C. *et al.* XYZ Ruby Code: Making a Case for a Three-Colored Graphical Calculus for Quantum Error Correction in Spacetime. *PRX Quantum* **6**. ISSN: 2691-3399. <http://dx.doi.org/10.1103/PRXQuantum.6.010360> (Mar. 2025).
50. Derks, P.-J. H. S. *et al.* *Dynamical codes for hardware with noisy readouts* 2025. arXiv: [2505.07658](https://arxiv.org/abs/2505.07658) [quant-ph]. <https://arxiv.org/abs/2505.07658>.
51. Xu, Y. & Dua, A. *Fault-tolerant protocols through spacetime concatenation* 2025. arXiv: [2504.08918](https://arxiv.org/abs/2504.08918) [quant-ph]. <https://arxiv.org/abs/2504.08918>.
52. De la Fuente, J. C. M. *Dynamical weight reduction of Pauli measurements* 2024. arXiv: [2410.12527](https://arxiv.org/abs/2410.12527) [quant-ph]. <https://arxiv.org/abs/2410.12527>.
53. Townsend-Teague, A., Magdalena de la Fuente, J. & Kesselring, M. Floquetifying the Colour Code. *Electronic Proceedings in Theoretical Computer Science* **384**, 265–303. ISSN: 2075-2180. <http://dx.doi.org/10.4204/EPTCS.384.14> (Aug. 2023).
54. Rodatz, B., Poór, B. & Kissinger, A. Floquetifying stabiliser codes with distance-preserving rewrites. *arXiv preprint* **2410.17240** (2024).
55. Jacoby, S., Retzker, A. & Pastawski, F. Stairway Codes: Floquetifying Bivariate Bicycle Codes and Beyond. *arXiv preprint* (Feb. 2026).
56. Poulin, D. Stabilizer Formalism for Operator Quantum Error Correction. *Physical Review Letters* **95**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.95.230504> (Dec. 2005).
57. Li, M., Gutiérrez, M., David, S. E., Hernandez, A. & Brown, K. R. Fault tolerance with bare ancillary qubits for a $[[7,1,3]]$ code. *Phys. Rev. A* **96**, 032341. <https://link.aps.org/doi/10.1103/PhysRevA.96.032341> (3 2017).
58. Watrous, J. *The Theory of Quantum Information* <https://cs.uwaterloo.ca/~watrous/TQI/> (Cambridge University Press, 2018).
59. Emerson, J. *Theory of Quantum Systems* (Personal communication, 2025).

60. Gottesman, D. *An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation* 2009. arXiv: [0904.2557](https://arxiv.org/abs/0904.2557) [quant-ph]. <https://arxiv.org/abs/0904.2557>.
61. Gottesman, D. A Theory of Fault-Tolerant Quantum Computation. *Physical Review A* **57**, 127–137. ISSN: 1050-2947, 1094-1622. arXiv: [quant-ph/9702029](https://arxiv.org/abs/quant-ph/9702029) (Jan. 1998).
62. Mackenzie, R. *Introduction to group theory in quantum mechanics*, Université de Montréal (Personal communication, 2024).
63. Shi, Y. *Both Toffoli and Controlled-NOT need little help to do universal quantum computation* 2002. arXiv: [quant-ph/0205115](https://arxiv.org/abs/quant-ph/0205115) [quant-ph]. <https://arxiv.org/abs/quant-ph/0205115>.
64. Gottesman, D. & Chuang, I. L. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature* **402**, 390–393. ISSN: 1476-4687. <http://dx.doi.org/10.1038/46503> (Nov. 1999).
65. Calderbank, A. R., Rains, E. M., Shor, P. W. & Sloane, N. J. A. Quantum Error Correction and Orthogonal Geometry. *Physical Review Letters* **78**, 405–408. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.78.405> (Jan. 1997).
66. Dehaene, J. & De Moor, B. Clifford group, stabilizer states, and linear and quadratic operations over GF(2). *Physical Review A* **68**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.68.042318> (Oct. 2003).
67. Harrow, A. *Quantum Information Science II: Lecture 5 - Stabilizers* Feb. 2018.
68. Harrow, A. *Quantum Information Science II: Lecture 6 - Clifford Group* Feb. 2018.
69. Gottesman, D. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Physical Review A* **54**, 1862–1868. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.54.1862> (Sept. 1996).
70. Hein, M., Eisert, J. & Briegel, H. J. Multiparty Entanglement in Graph States. *Physical Review A* **69**, 062311. ISSN: 1050-2947, 1094-1622 (June 2004).
71. Van den Nest, M., Dehaene, J. & De Moor, B. Graphical description of the action of local Clifford transformations on graph states. *Physical Review A* **69**, 022316 (2004).
72. Backens, M. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics* **16**, 093021 (2014).
73. Schlingemann, D. *Stabilizer Codes Can Be Realized as Graph Codes* Nov. 2001. eprint: [quant-ph/0111080](https://arxiv.org/abs/quant-ph/0111080).

74. Grassl, M., Klappenecker, A. & Roetteler, M. *Graphs, Quadratic Forms, and Quantum Codes* in *Proceedings IEEE International Symposium on Information Theory*, (2002), 45. eprint: [quant-ph/0703112](https://arxiv.org/abs/quant-ph/0703112).
75. Bouchet, A. Graphic presentations of isotropic systems. *Journal of Combinatorial Theory, Series B* **45**, 58–76 (1988).
76. Greenbaum, D. & Dutton, Z. Modeling coherent errors in quantum error correction. *Quantum Science and Technology* **3**, 015007. ISSN: 2058-9565. [http://dx.doi.org/10.1088/2058-9565/aa9a06](https://dx.doi.org/10.1088/2058-9565/aa9a06) (Dec. 2017).
77. Aliferis, P. & Preskill, J. Fault-tolerant quantum computation against biased noise. *Physical Review A* **78**. ISSN: 1094-1622. [http://dx.doi.org/10.1103/PhysRevA.78.052331](https://dx.doi.org/10.1103/PhysRevA.78.052331) (Nov. 2008).
78. Klesse, R. & Frank, S. Quantum Error Correction in Spatially Correlated Quantum Noise. *Physical Review Letters* **95**. ISSN: 1079-7114. [http://dx.doi.org/10.1103/PhysRevLett.95.230503](https://dx.doi.org/10.1103/PhysRevLett.95.230503) (Nov. 2005).
79. Breuer, H.-P., Laine, E.-M., Piilo, J. & Vacchini, B. *Colloquium*: Non-Markovian dynamics in open quantum systems. *Reviews of Modern Physics* **88**. ISSN: 1539-0756. [http://dx.doi.org/10.1103/RevModPhys.88.021002](https://dx.doi.org/10.1103/RevModPhys.88.021002) (Apr. 2016).
80. Knill, E., Laflamme, R. & Milburn, G. *Efficient Linear Optics Quantum Computation* 2000. arXiv: [quant-ph/0006088](https://arxiv.org/abs/quant-ph/0006088) [quant-ph]. <https://arxiv.org/abs/quant-ph/0006088>.
81. Ghosh, J., Fowler, A. G., Martinis, J. M. & Geller, M. R. Understanding the effects of leakage in superconducting quantum-error-detection circuits. *Physical Review A* **88**. ISSN: 1094-1622. [http://dx.doi.org/10.1103/PhysRevA.88.062329](https://dx.doi.org/10.1103/PhysRevA.88.062329) (Dec. 2013).
82. Fowler, A. G., Mariantoni, M., Martinis, J. M. & Cleland, A. N. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* **86**. ISSN: 1094-1622. [http://dx.doi.org/10.1103/PhysRevA.86.032324](https://dx.doi.org/10.1103/PhysRevA.86.032324) (Sept. 2012).
83. Knill, E., Laflamme, R., Martinez, R. & Negrevergne, C. Benchmarking Quantum Computers: The Five-Qubit Error Correcting Code. *Physical Review Letters* **86**, 5811–5814. ISSN: 1079-7114. [http://dx.doi.org/10.1103/PhysRevLett.86.5811](https://dx.doi.org/10.1103/PhysRevLett.86.5811) (June 2001).
84. Calderbank, A. R. & Shor, P. W. Good quantum error-correcting codes exist. *Phys. Rev. A* **54**, 1098–1105. <https://link.aps.org/doi/10.1103/PhysRevA.54.1098> (2 Aug. 1996).

85. Steane, A. Multiple Particle Interference and Quantum Error Correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **452**, 2551–2577. ISSN: 1471-2946. <http://dx.doi.org/10.1098/rspa.1996.0136> (Nov. 1996).
86. Kitaev, A. Y. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys* **52**, 1191. <https://doi.org/10.1070/RM1997v052n06ABEH002155> (Dec. 1997).
87. Bravyi, S. B. & Kitaev, A. Y. *Quantum codes on a lattice with boundary* 1998. arXiv: [quant-ph/9811052](https://arxiv.org/abs/quant-ph/9811052) [quant-ph]. <https://arxiv.org/abs/quant-ph/9811052>.
88. Bombin, H. & Martin-Delgado, M. A. Topological Quantum Distillation. *Physical Review Letters* **97**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.97.180501> (Oct. 2006).
89. Bombin, H. *An Introduction to Topological Quantum Codes* 2013. arXiv: [1311.0277](https://arxiv.org/abs/1311.0277) [quant-ph]. <https://arxiv.org/abs/1311.0277>.
90. Postol, M. S. *A Proposed Quantum Low Density Parity Check Code* 2001. arXiv: [quant-ph/0108131](https://arxiv.org/abs/quant-ph/0108131) [quant-ph]. <https://arxiv.org/abs/quant-ph/0108131>.
91. Breuckmann, N. P. & Eberhardt, J. N. Quantum Low-Density Parity-Check Codes. *PRX Quantum* **2**. ISSN: 2691-3399. <http://dx.doi.org/10.1103/PRXQuantum.2.040101> (Oct. 2021).
92. Schlingemann, D. & Werner, R. F. Quantum error-correcting codes associated with graphs. *Physical Review A* **65**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.65.012308> (Dec. 2001).
93. MacKay, D., Mitchison, G. & McFadden, P. Sparse-Graph Codes for Quantum Error Correction. *IEEE Transactions on Information Theory* **50**, 2315–2330. ISSN: 0018-9448. <http://dx.doi.org/10.1109/TIT.2004.834737> (Oct. 2004).
94. Aaronson, S. & Gottesman, D. Improved simulation of stabilizer circuits. *Physical Review A* **70**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.70.052328> (Nov. 2004).
95. Hsieh, M.-H. & Le Gall, F. NP-hardness of decoding quantum error-correction codes. *Physical Review A* **83**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.83.052331> (May 2011).
96. Iyer, P. & Poulin, D. *Hardness of decoding quantum stabilizer codes* 2013. arXiv: [1310.3235](https://arxiv.org/abs/1310.3235) [quant-ph]. <https://arxiv.org/abs/1310.3235>.

97. Dennis, E., Kitaev, A., Landahl, A. & Preskill, J. Topological quantum memory. *Journal of Mathematical Physics* **43**, 4452–4505. ISSN: 1089-7658. <http://dx.doi.org/10.1063/1.1499754> (Sept. 2002).
98. Bravyi, S., Suchara, M. & Vargo, A. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A* **90**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.90.032326> (Sept. 2014).
99. Vuillot, C. *et al.* Code deformation and lattice surgery are gauge fixing. *New Journal of Physics* **21**, 033028. ISSN: 1367-2630. <http://dx.doi.org/10.1088/1367-2630/ab0199> (Mar. 2019).
100. Bacon, D. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Physical Review A* **73**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.73.012340> (Jan. 2006).
101. Bravyi, S., Duclos-Cianci, G., Poulin, D. & Suchara, M. *Subsystem surface codes with three-qubit check operators* 2013. arXiv: [1207.1443](https://arxiv.org/abs/1207.1443) [quant-ph]. <https://arxiv.org/abs/1207.1443>.
102. Gottesman, D. *The Heisenberg Representation of Quantum Computers* 1998. arXiv: [quant-ph/9807006](https://arxiv.org/abs/quant-ph/9807006) [quant-ph]. <https://arxiv.org/abs/quant-ph/9807006>.
103. Vaidman, L., Goldenberg, L. & Wiesner, S. Error prevention scheme with four particles. *Physical Review A* **54**, R1745–R1748. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.54.R1745> (Sept. 1996).
104. Grassl, M., Beth, T. & Pellizzari, T. Codes for the quantum erasure channel. *Physical Review A* **56**, 33–38. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.56.33> (July 1997).
105. Bombin, H. Topological subsystem codes. *Physical Review A* **81**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.81.032301> (Mar. 2010).
106. Kitaev, A. Fault-tolerant quantum computation by anyons. *Annals of Physics* **303**, 2–30. ISSN: 0003-4916. [http://dx.doi.org/10.1016/S0003-4916\(02\)00018-0](http://dx.doi.org/10.1016/S0003-4916(02)00018-0) (Jan. 2003).
107. Duncan, R., Kissinger, A., Perdrix, S. & Wetering, J. v. d. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum* **4**, 279. <https://quantum-journal.org/papers/q-2020-06-04-279/> (2020).
108. Duncan, R. & Perdrix, S. *Graph States and the Necessity of Euler Decomposition in Mathematical Theory and Computational Practice* (Springer, 2009), 167–177.

109. Duncan, R. & Perdrix, S. Pivoting makes the ZX-calculus complete for real stabilizers. *Electronic Proceedings in Theoretical Computer Science* **171**, 50–62 (2014).
110. Newman, M. *Networks* ISBN: 978-0-19-880509-0 (Oxford University Press, July 2018).
111. Hu, A. T. & Khesin, A. B. Improved graph formalism for quantum circuit simulation. *Physical Review A* **105**, 022432 (2022).
112. Aharonov, D. & Ben-Or, M. *Fault-Tolerant Quantum Computation With Constant Error Rate* 1999. arXiv: [quant-ph/9906129](https://arxiv.org/abs/quant-ph/9906129) [quant-ph]. <https://arxiv.org/abs/quant-ph/9906129>.
113. Knill, E., Laflamme, R. & Zurek, W. H. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **454**, 365–384. ISSN: 1471-2946. <http://dx.doi.org/10.1098/rspa.1998.0166> (Jan. 1998).
114. Gottesman, D. *Surviving as a Quantum Computer in a Classical World* 2026. <https://www.cs.umd.edu/~dgottesm/QECCbook-2026.pdf>.
115. Nielsen, M. & Chuang, I. *Quantum Computation and Quantum Information* ISBN: 978-1-107-00217-3 (Cambridge, 2000).
116. Knill, E., Laflamme, R. & Zurek, W. *Threshold Accuracy for Quantum Computation* 1996. arXiv: [quant-ph/9610011](https://arxiv.org/abs/quant-ph/9610011) [quant-ph]. <https://arxiv.org/abs/quant-ph/9610011>.
117. Steane, A. *Quantum Reed-Muller Codes* 1996. arXiv: [quant-ph/9608026](https://arxiv.org/abs/quant-ph/9608026) [quant-ph]. <https://arxiv.org/abs/quant-ph/9608026>.
118. Bennett, C. H. *et al.* Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.* **70**, 1895–1899. <https://link.aps.org/doi/10.1103/PhysRevLett.70.1895> (13 Mar. 1993).
119. Shor, P. *Fault-tolerant quantum computation* in *Proceedings of 37th Conference on Foundations of Computer Science* (1996), 56–65. <https://arxiv.org/abs/quant-ph/9605011>.
120. Liou, P.-H. & Lai, C.-Y. Parallel syndrome extraction with shared flag qubits for Calderbank-Shor-Steane codes of distance three. *Physical Review A* **107**. ISSN: 2469-9934. <http://dx.doi.org/10.1103/PhysRevA.107.022614> (Feb. 2023).
121. Bravyi, S. *et al.* High-threshold and low-overhead fault-tolerant quantum memory. *Nature* **627**, 778–782. ISSN: 1476-4687. <http://dx.doi.org/10.1038/s41586-024-07107-7> (Mar. 2024).

122. Xu, S., Liu, K., Rall, P., He, Z. & Ding, Y. *Distilling Magic States in the Bicycle Architecture* 2026. arXiv: [2602.20546](https://arxiv.org/abs/2602.20546) [quant-ph]. <https://arxiv.org/abs/2602.20546>.
123. Gidney, C., Shu, N. & Jones, C. *Magic state cultivation: growing T states as cheap as CNOT gates* 2024. arXiv: [2409.17595](https://arxiv.org/abs/2409.17595) [quant-ph]. <https://arxiv.org/abs/2409.17595>.
124. Bombin, H. & Martin-Delgado, M. A. Topological Computation without Braiding. *Physical Review Letters* **98**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.98.160502> (Apr. 2007).
125. Gidney, C. & Jones, C. *New circuits and an open source decoder for the color code* 2023. arXiv: [2312.08813](https://arxiv.org/abs/2312.08813) [quant-ph]. <https://arxiv.org/abs/2312.08813>.
126. Chakraborty, A. & Gottesman, D. *No-Go Theorem on Fault Tolerant Gadgets for Multiple Logical Qubits* 2026. arXiv: [2602.13395](https://arxiv.org/abs/2602.13395) [quant-ph]. <https://arxiv.org/abs/2602.13395>.
127. De Beaudrap, N. & Horsman, D. The ZX calculus is a language for surface code lattice surgery. *Quantum* **4**, 218. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2020-01-09-218> (Jan. 2020).
128. Fowler, A. G. & Gidney, C. *Low overhead quantum computation using lattice surgery* 2019. arXiv: [1808.06709](https://arxiv.org/abs/1808.06709) [quant-ph]. <https://arxiv.org/abs/1808.06709>.
129. Kubica, A., Yoshida, B. & Pastawski, F. Unfolding the color code. *New Journal of Physics* **17**, 083026. ISSN: 1367-2630. <http://dx.doi.org/10.1088/1367-2630/17/8/083026> (Aug. 2015).
130. Moussa, J. E. Transversal Clifford gates on folded surface codes. *Physical Review A* **94**. ISSN: 2469-9934. <http://dx.doi.org/10.1103/PhysRevA.94.042316> (Oct. 2016).
131. Gottesman, D. Fault-Tolerant Quantum Computation with Higher-Dimensional Systems. *Chaos, Solitons & Fractals* **10**, 1749–1758. ISSN: 0960-0779. [http://dx.doi.org/10.1016/S0960-0779\(98\)00218-5](http://dx.doi.org/10.1016/S0960-0779(98)00218-5) (Sept. 1999).
132. Bombin, H. Topological Order with a Twist: Ising Anyons from an Abelian Model. *Phys. Rev. Lett.* **105**, 030403. <https://link.aps.org/doi/10.1103/PhysRevLett.105.030403> (3 July 2010).
133. Bombin, H. Clifford gates by code deformation. *New Journal of Physics* **13**, 043005. ISSN: 1367-2630. <http://dx.doi.org/10.1088/1367-2630/13/4/043005> (Apr. 2011).

134. Litinski, D. & Oppen, F. v. Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes. *Quantum* **2**, 62. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2018-05-04-62> (May 2018).
135. Cohen, L. Z., Kim, I. H., Bartlett, S. D. & Brown, B. J. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Science Advances* **8**. ISSN: 2375-2548. <http://dx.doi.org/10.1126/sciadv.abn1717> (2022).
136. Cross, A. W., He, Z., Rall, P. J. & Yoder, T. J. *Improved QLDPC Surgery: Logical Measurements and Bridging Codes* 2025. arXiv: [2407.18393](https://arxiv.org/abs/2407.18393) [quant-ph]. <https://arxiv.org/abs/2407.18393>.
137. Gottesman, D. *Quantum fault tolerance in small experiments* 2016. arXiv: [1610.03507](https://arxiv.org/abs/1610.03507) [quant-ph]. <https://arxiv.org/abs/1610.03507>.
138. Koh, J. M. *et al. Entangling logical qubits without physical operations* 2026. arXiv: [2601.20927](https://arxiv.org/abs/2601.20927) [quant-ph]. <https://arxiv.org/abs/2601.20927>.
139. Sayginel, H., Koutsoumpas, S., Webster, M., Rajput, A. & Browne, D. E. *Fault-Tolerant Logical Clifford Gates from Code Automorphisms* 2025. arXiv: [2409.18175](https://arxiv.org/abs/2409.18175) [quant-ph]. <https://arxiv.org/abs/2409.18175>.
140. Breuckmann, N. P. & Burton, S. Fold-Transversal Clifford Gates for Quantum Codes. *Quantum* **8**, 1372. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2024-06-13-1372> (June 2024).
141. Aharonov, D. *A Simple Proof that Toffoli and Hadamard are Quantum Universal* 2003. arXiv: [quant-ph/0301040](https://arxiv.org/abs/quant-ph/0301040) [quant-ph]. <https://arxiv.org/abs/quant-ph/0301040>.
142. Anderson, J. T., Duclos-Cianci, G. & Poulin, D. Fault-Tolerant Conversion between the Steane and Reed-Muller Quantum Codes. *Physical Review Letters* **113**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.113.080501> (Aug. 2014).
143. Zeng, B., Cross, A. & Chuang, I. L. *Transversality versus Universality for Additive Quantum Codes* 2007. arXiv: [0706.1382](https://arxiv.org/abs/0706.1382) [quant-ph]. <https://arxiv.org/abs/0706.1382>.
144. Hill, C. D., Fowler, A. G., Wang, D. S. & Hollenberg, L. C. L. *Fault-tolerant quantum error correction code conversion* 2011. arXiv: [1112.2417](https://arxiv.org/abs/1112.2417) [quant-ph]. <https://arxiv.org/abs/1112.2417>.
145. Colladay, K. R. & Mueller, E. J. Rewiring stabilizer codes. *New Journal of Physics* **20**, 083030. ISSN: 1367-2630. <http://dx.doi.org/10.1088/1367-2630/aad8dd> (Aug. 2018).

146. DiVincenzo, D. P. & Aliferis, P. Effective Fault-Tolerant Quantum Computation with Slow Measurements. *Physical Review Letters* **98**. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.98.020501> (Jan. 2007).
147. Steane, A. M. Active Stabilization, Quantum Computation, and Quantum State Synthesis. *Physical Review Letters* **78**, 2252–2255. ISSN: 1079-7114. <http://dx.doi.org/10.1103/PhysRevLett.78.2252> (Mar. 1997).
148. Knill, E. Quantum computing with realistically noisy devices. *Nature* **434**, 39–44. ISSN: 1476-4687. <http://dx.doi.org/10.1038/nature03350> (Mar. 2005).
149. Chamberland, C. & Beverland, M. E. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum* **2**, 53. ISSN: 2521-327X. <http://dx.doi.org/10.22331/q-2018-02-08-53> (Feb. 2018).
150. Chao, R. & Reichardt, B. W. Flag Fault-Tolerant Error Correction for any Stabilizer Code. *PRX Quantum* **1**. ISSN: 2691-3399. <http://dx.doi.org/10.1103/PRXQuantum.1.010302> (2020).
151. Grans-Samuelsson, L., Aasen, D. & Bonderson, P. Fault-tolerant pairwise measurement-based code on eight qubits. *Phys. Rev. A* **112**, 042413. <https://link.aps.org/doi/10.1103/c54n-y9vw> (4 Oct. 2025).
152. Raussendorf, R., Browne, D. E. & Briegel, H. J. Measurement-based quantum computation on cluster states. *Physical Review A* **68**. ISSN: 1094-1622. <http://dx.doi.org/10.1103/PhysRevA.68.022312> (Aug. 2003).
153. Banfield, D., Leitch, H. & Kay, A. Implementing Clifford Gates on Stabilizer Codes via Measurement. *arXiv preprint* **2210.14074** (2025).
154. McKay, B. D. & Piperno, A. Practical graph isomorphism, II. *Journal of Symbolic Computation* **60**, 94–112. <https://www.sciencedirect.com/science/article/pii/S0747717113001193> (2014).
155. Aasen, D., Haah, J., Li, Z. & Mong, R. S. K. *Measurement Quantum Cellular Automata and Anomalies in Floquet Codes* 2023. arXiv: [2304.01277](https://arxiv.org/abs/2304.01277) [quant-ph]. <https://arxiv.org/abs/2304.01277>.
156. Huang, C. & Newman, M. *Transversal switching between generic stabilizer codes* 2018. arXiv: [1709.09282](https://arxiv.org/abs/1709.09282) [quant-ph]. <https://arxiv.org/abs/1709.09282>.
157. Yu, S., Chen, Q. & Oh, C. *Graphical Quantum Error-Correcting Codes* 2007. arXiv: [0709.1780](https://arxiv.org/abs/0709.1780) [quant-ph]. <https://arxiv.org/abs/0709.1780>.
158. Cross, A. & Vandeth, D. *Small Binary Stabilizer Subsystem Codes* 2025. arXiv: [2501.17447](https://arxiv.org/abs/2501.17447) [quant-ph]. <https://arxiv.org/abs/2501.17447>.

- 159. Morley-Short, S. *gsc (graph-state compass)* <https://github.com/sammorley-short/gsc>. 2019.
- 160. Adcock, J. C., Morley-Short, S., Dahlberg, A. & Silverstone, J. W. Mapping graph state orbits under local complementation. *Quantum* **4**, 305 (2020).
- 161. Zhou, C. Personal communication, Canadian Workshop on Quantum Algorithms 2026.