ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 38

**Settore Concorsuale:** 09/F2 - TELECOMUNICAZIONI

**Settore Scientifico Disciplinare:** ING-INF/03 - TELECOMUNICAZIONI

# DESIGN, DECODING, AND FAULT-TOLERANT IMPLEMENTATION OF QUANTUM ERROR-CORRECTING CODES

**Presentata da:** Diego Forlivesi

**Coordinatore Dottorato**

Davide Dardari

**Supervisore**

Marco Chiani

**Co-supervisore**

Enrico Paolini

Esame finale anno 2026

# Abstract

Quantum computation promises to revolutionize information processing by harnessing the unique properties of quantum systems, such as superposition and entanglement. It offers unprecedented potential in areas such as simulating quantum systems, solving combinatorial optimization problems, and enabling secure quantum communication. Beyond individual processors, the quantum internet aims to connect quantum nodes through entanglement distribution, enabling distributed computation. Yet, quantum states are exceedingly fragile: interactions with the environment, imperfections in gate operations, and measurement errors can quickly degrade the encoded information. Ensuring reliable quantum computation therefore demands both carefully designed quantum codes to protect information and robust protocols that prevent the propagation of errors during computation. Unlike classical redundancy schemes, quantum error correction must preserve delicate phase relationships and rely on indirect error detection strategies, such as syndrome extraction with ancillary qubits, which enable error identification without collapsing the encoded states.

Topological codes, in particular, emerged early as the preferred choice for many architectures due to their planar structure, local interactions, and natural support for fault-tolerant logical operations. Color codes, for instance, allow transversal implementation of all Clifford gates, whereas non-Clifford operations require magic-state injection. More recent constructions, including quantum low density parity check codes, offer higher encoding rates and reduced resource overhead, but require long-range qubit connectivity and more complex protocols for implementing logical gates, such as lattice surgery and measurement-based procedures. Over the years, a rich ecosystem of fault-tolerant strategies has developed, from flag-based syndrome extraction to advanced distillation and code-switching

schemes, reflecting the deep interplay between code structure, error models, and operational requirements. Experimental progress has kept pace: repeated error-correction cycles of surface and Steane codes, preparation of logical Bell and GHZ states, and small-scale demonstrations of non-Clifford operations have been realized on various quantum computing platforms.

Despite these advances, several challenges remain and motivate further investigation. Experimentally, many platforms exhibit biased noise underscoring the need for codes that exploit channel asymmetry to reduce logical error rates. Real-time decoding continues to be a critical bottleneck: although post-processing can validate error correction, fault-tolerant computation requires low-latency decoders capable of keeping pace with microsecond-scale syndrome data. Additionally, fault-tolerant protocols for the initialization of logical states remain resource-intensive. Current approaches, based on repeated stabilizer measurements, scale poorly with code distance and often introduce significant overhead, limiting practical implementations.

Addressing these gaps, this thesis develops a unified framework for advancing quantum error correction from theory to practice. From a theoretical perspective, precise characterization of error-correcting capabilities remains essential, particularly under low physical error rates where Monte Carlo simulations are inefficient. Building on this foundation, it introduces quantum codes specifically designed for biased channels, exploiting noise asymmetry to improve logical error rates. Next, fast decoding algorithms with linear complexity are developed for real-time syndrome processing, ensuring that logical errors can be corrected within microsecond timescales. Finally, the thesis presents modular and efficient strategies for fault-tolerant state preparation of CSS codes, significantly reducing gate overhead. In particular, a logical zero state for the Golay code was implemented on the Quantinuum H2 trapped-ion quantum computer, providing a concrete demonstration of scalable fault-tolerant state preparation.

# Contents

# Acronyms

**BC** bubble clustering

**BD** bounded distance

**BP+OSD** belief propagation plus ordered statistics decoding

**CC** code-capacity

**CSS** Calderbank, Shor, and Steane

**DD** dynamical decoupling

**FT** fault-tolerant

**i.i.d.** independent identically distributed

**LEMON** library for efficient modeling and optimization in networks

**LUT** look-up table

**MST** minimum spanning tree

**MW** minimum weight

**MWPM** minimum weight perfect matching

**QECC** quantum error-correcting code

**QEC** quantum error correction

**QLDPC** quantum low density parity check

**RFire** Rapid-Fire

**STM** spanning tree matching

**UF** union-find

**WE** weight enumerator

# Chapter 1

# Introduction

Quantum computation leverages the phenomena of superposition and entanglement to process information in ways that are fundamentally different from classical devices. A single $n$-qubit quantum state lives in a $2^n$-dimensional complex vector space, allowing certain algorithms to explore exponentially large solution spaces more efficiently than classical algorithms. This leads to compelling potential applications across a variety of domains. One prominent example is the simulation of quantum systems in chemistry, materials science, and condensed matter physics, where classical computational methods rapidly become intractable [1, 2]. Another important area is combinatorial optimization, where approaches such as the quantum approximate optimization algorithm and quantum annealing offer possible speedups for solving NP-hard problems [3, 4]. In cryptanalysis, Shor's algorithm shows that large-scale quantum computers could break widely used public-key cryptosystems [5]. At the same time, quantum technologies enable novel forms of secure communication, most notably quantum key distribution, providing information-theoretic security guaranteed by the laws of physics [6]. Further opportunities are emerging in quantum machine learning and data science, where quantum subroutines for linear algebra may offer computational advantages over their classical counterparts [7].

Beyond stand-alone quantum processors, there is a broader vision of a quantum internet: a network of quantum nodes linked by entanglement distribution and quantum communication protocols [8]. Such a network could enable distributed quantum computation, where separate processors collaborate on large-scale tasks, and quantum-secure communication via entanglement-based key distribution [9]. Realizing this vision is technically challenging, requiring long-distance preservation of entanglement, high-fidelity quantum memories, and communication protocols designed specifically for quantum information [10, 11]. Nonetheless, recent experimental progress, such as entanglement distribution over metropolitan-scale fiber networks, demonstrates that the foundations of such an infrastructure are already being established [12]. Taken together, these advances suggest that the future of quantum technologies will not be limited to isolated devices but will instead involve an ecosystem of networked quantum machines. This perspective highlights the central importance of quantum error correction (QEC) and fault-tolerance, which provide the foundation for both scalable quantum computation and the reliable operation of quantum networks.

Several physical platforms are currently being pursued for the realization of quantum processors, each offering distinct advantages and challenges. Superconducting qubits, based on Josephson junction circuits, enable fast gate operations and benefit from mature microfabrication techniques, but typically exhibit limited qubit connectivity, comparatively short coherence times, and the need for cryogenic operation [13]. Trapped-ion systems offer excellent coherence properties, high-fidelity gates, and effectively all-to-all connectivity within ion chains, at the expense of slower gate speeds and scalability challenges associated with ion control and transport [14]. Neutral atom platforms, implemented using optical tweezers and Rydberg interactions, provide long coherence times and reconfigurable, long-range connectivity across large two-dimensional arrays, although gate fidelities and interaction uniformity remain active areas of research [15]. Finally, photonic architectures naturally support long-distance quantum communication and room-temperature operation, making them well suited for quantum networking, but face fundamental challenges in realizing deterministic two-qubit gates and scalable quantum memories [16].

## 1.1 Protecting Quantum Information: Principles and Codes

In classical computation, information is encoded in bits, each of which assumes one of two discrete values, $0$ or $1$. At every step of a computation, the system can reliably correct deviations toward the nearest of these two stable states, thereby ensuring robustness against small errors. By contrast, quantum computation relies on qubits, whose states are not restricted to discrete values but instead can occupy any superposition of the computational basis states $|0\rangle$ and $|1\rangle$. This continuous structure makes quantum information intrinsically more fragile: even in the absence of active operations, interactions with the environment lead to a gradual loss of coherence. While each such deviation may be slight, these deviations accumulate over time and, without error correction, eventually compromise the fidelity of the computation.

There exists a well-developed theory of classical error-correcting codes, but it cannot be applied to the quantum setting in a straightforward way. Indeed, whereas classical codes only need to correct for bit flips, quantum codes must also preserve the relative phase of the state, which is equally crucial for maintaining quantum information. Moreover, many classical strategies rely on direct duplication of information. For instance, the simplest classical coding scheme, the repetition code, stores a logical zero as three physical zeros and a logical one as three physical ones. Such a scheme can detect and correct errors by majority voting: if one bit is flipped, the system can recover the original logical value. One might imagine a quantum analogue in which an unknown qubit state is copied directly onto multiple qubits. However, this is impossible due to the No-Cloning theorem, which forbids making a perfect copy of an unknown quantum state [17, 18]. Even attempting to verify the stored information by measurement is problematic, because measurement collapses the quantum state: a superposition of $|0\rangle$ and $|1\rangle$ will collapse to either $|0\rangle$ or $|1\rangle$ with probabilities determined by the amplitudes, destroying the original quantum information. As a result, QEC requires fundamentally different strategies. Remarkably, although qubit errors may appear continuous in nature, it suffices to correct a discrete set of representative errors to protect quantum information [19]. Redundancy is introduced not

through direct copying but by entangling the logical state with additional ancillary qubits [20]. Carefully designed protocols, such as syndrome extraction, allow errors to be detected and corrected indirectly, without disturbing the quantum information itself [19,21]. The resulting syndromes contain information about the physical errors that have occurred; a decoder processes it and attempts to restore the logical state.

Given the challenges outlined above and the fact that no quantum gate can be implemented perfectly, a central question arises: can a quantum computer perform extended computations without noise compromising their fidelity? The quantum fault-tolerance threshold theorem provides a resolution to this concern. It establishes that, if the error rate per qubit and per operation is below a certain critical threshold, then arbitrarily long quantum computations can be carried out with arbitrarily high fidelity. This is achieved through fault-tolerant (FT) protocols that combine quantum error-correcting codes with systematic error detection and correction at every stage of the computation. Although these protocols require additional qubits and operations, the resource overhead grows only polylogarithmically with the size of the ideal circuit and with the inverse of the target accuracy [19,22]. The threshold theorem therefore demonstrates that scalable quantum computation is possible, provided the physical error rate lies below the threshold.

The first explicit constructions of QEC were provided by codes such as the Shor code and the Steane code. Shor's nine-qubit code, inspired by the classical repetition code, was the first to show explicitly that arbitrary single-qubit errors, whether bit flips, phase flips, or their combinations, could be corrected in principle [20]. Soon after, the Steane seven-qubit code demonstrated a more efficient construction, encoding one logical qubit into seven physical qubits [23]. A decisive step forward came with the development of the stabilizer formalism, which provided a unifying mathematical framework for describing and analyzing large families of quantum codes. In this approach, codes are identified by the operators that leave their logical states invariant, offering a compact and systematic way to specify complex code spaces [19]. This perspective not only clarified the structure underlying many existing codes but also enabled the general formulation of methods for encoding, detecting, and correcting errors. A particularly important class of stabilizer codes is the Calderbank, Shor, and Steane (CSS) codes, whose stabi-

lizers are composed exclusively of either $\boldsymbol{X}$-type or $\boldsymbol{Z}$-type generators [24, 25]. By organizing the stabilizers in this way, CSS codes allow for independent detection and correction of bit-flip and phase-flip errors. Additionally, this structure allows for a simplified implementation of certain encoded logical operations.

Among the most influential families of codes to emerge are the topological codes, which exploit the geometry of qubits arranged on a lattice [26–30]. The surface code has become the most prominent representative of this class. Defined on a two-dimensional grid with only local stabilizer measurements, it is particularly well suited for implementation in hardware platforms with nearest-neighbor interactions. In addition, the surface code exhibits one of the highest known threshold error rates, making it a leading candidate for large-scale FT quantum computation. Closely related are the color codes, which share the advantages of locality but also possess an additional feature: the full Clifford group can be generated transversally, meaning that each physical qubit in the code block is acted upon independently. This property is valuable because it prevents errors from spreading uncontrollably across qubits, thereby simplifying the realization of FT logical operations. However, neither surface nor planar color codes alone enable a universal set of gates, as non-Clifford operations cannot be realized transversally [31]. Achieving full universality therefore requires complementary FT techniques, such as magic state distillation [32] or code switching [33], which supplement these topological architectures with the necessary non-Clifford resources.

Despite the robustness and high thresholds of topological codes, they require a large number of physical qubits to encode a single logical qubit. This challenge has motivated the more recent development of quantum low density parity check (QLDPC) codes, which aim to reduce resource overhead while ensuring that each stabilizer involves only a limited number of qubits. This property is important both for technological feasibility and for fault-tolerance, as it simplifies implementation and enables better control of error propagation during syndrome extraction. However, this comes at the cost of requiring long-range interactions between qubits, since the stabilizer checks are generally non-local and may involve qubits separated by large distances on the physical layout. Moreover, performing encoded operations on QLDPC codes often requires more complex protocols, such as measurement-based procedures [34] and lattice surgery [35–37], to

implement logical gates fault-tolerantly. The first explicit QLDPC constructions were the bicycle codes, a class of CSS codes whose stabilizer generators are constructed from a binary circulant matrix in a way that automatically satisfies the commutation conditions [38]. Generalized bicycle codes extended this scheme by allowing pairs of commuting matrices to define the stabilizers, providing greater flexibility in code design [39]. Building on this, bivariate bicycle codes fall into the family of tensor-product generalized bicycle codes. The resulting structure can be visualized on a two-dimensional grid, where some of the check operators are non-local and obtained by systematic shifts along the grid [40]. In parallel, homological product codes and related constructions were introduced, taking the tensor product of two classical or quantum codes, producing larger codes with increased encoding capacity and improved error-correcting performance [41, 42]. Variants such as fiber bundle codes introduce controlled "twists" in the stabilizers to enhance code properties [43]. Lifted product codes generalize the product construction further by applying it to a pair of protographs, which are matrices constructed from linear combinations of circulant permutation matrices [44]. Separately, tri-orthogonal codes were developed not as QLDPC codes but as a class of stabilizer codes designed for FT magic-state distillation. Their defining feature is a triorthogonality condition on the stabilizer generators: any pair of rows overlaps on an even number of positions, and any triple of rows also overlaps on an even number of positions [45]. This property enables the transversal implementation of a non-Clifford logical gate across the code block, a crucial ingredient for achieving a universal set of FT logical operations.

## 1.2 From Theory to Hardware: Realizing Quantum Codes

The transition from theoretical constructions of QEC to practical demonstrations on real hardware marks a critical step toward scalable quantum computing. In very recent years, this area has witnessed intense experimental activity, reflecting rapid progress in the development and testing of error-correcting codes on physical devices. Surface codes naturally emerged as the leading candidates for early im-

plementations, owing to their planar structure and local stabilizer measurements, which align well with the connectivity of superconducting qubit platforms. The first realizations focused on the distance three surface code [46, 47]. In these experiments, error syndromes were extracted and decoded in post-processing using the minimum weight perfect matching (MWPM) algorithm [48], which pairs nontrivial syndrome bits from ancilla measurements to infer likely data qubit errors, selecting connections that minimize the overall probability of the inferred error configuration. This setting established the feasibility of stabilizer-based error correction on real devices and demonstrated repeated error correction cycles. Building on these advances, later work investigated how logical performance scales with increasing code size. While larger codes in principle provide stronger error protection, they also require more qubits and operations, introducing additional errors. In [49], this trade-off was examined by implementing a distance five surface code and comparing its performance to the distance three surface code. Using belief-matching decoding [50], a hybrid of belief propagation and MWPM, the larger code achieved modest but consistent improvements in logical error rates, offering the first experimental evidence that increasing code distance can yield gains in fault-tolerance. A decisive advance came with the first demonstration of surface code operation below threshold on superconducting processors [51]. Distance five and distance seven surface codes were realized, respectively, and it was shown that increasing the code distance reduced the logical error rate by more than a factor of two, in line with theoretical expectations. Importantly, these experiments also implemented real-time decoding, based on a correlation-augmented MWPM algorithm that used Sparse Blossom as its main subroutine [52]. This represented a major step forward, showing that low-latency decoding can be integrated into experimental cycles while still maintaining operation below threshold. Beyond superconducting platforms, recent progress has also been achieved with neutral atom arrays, which offer long coherence times and reconfigurable connectivity. In [53], a logical processor was realized that demonstrated key building blocks of QEC. Using distance three to distance seven surface codes, logical Bell states were prepared and entangling performance was shown to improve with increasing code size. Complementarily, the Steane code was employed to implement FT Clifford circuits, including the preparation of a logical GHZ state.

While error correction with surface and color codes provides the foundation for protecting quantum information, achieving universal quantum computation further requires the ability to implement non-Clifford gates. A standard approach is magic state distillation, where multiple noisy magic states are purified into a smaller number of high-fidelity ones using error-correcting codes [32]. In a recent neutral atom experiment [54], a magic state factory based on the $[[5, 1, 3]]$ perfect code was demonstrated. The protocol consumed five noisy logical magic states, applied an optimized un-encoding circuit made of three layers of entangling gates, and produced a distilled output by measuring four of the logical qubits and postselecting on favorable outcomes. This procedure achieved quadratic suppression of logical error rates, validating the principles of magic state distillation in a hardware setting. Moreover, the experiment realized distillation of states encoded in color codes of distances three and five, directly observing an improvement in output fidelity compared to the noisy inputs. On trapped-ion processors, further advances have been achieved in the direct preparation of encoded magic states. In [55], a protocol was implemented to fault-tolerantly prepare a pair of logical magic states within the $[[6, 2, 2]]$ error-detecting code using only eight physical qubits. The scheme begins by encoding two candidate states through an arbitrary state encoder circuit, such that any single gate fault can affect at most one logical qubit. Potential single-qubit logical errors are then detected by fault-tolerantly measuring joint logical Hadamard operators with the aid of an ancilla. By discarding runs with detected faults, the protocol produces logical magic states whose failure probability scales quadratically with the physical error rate. These states were subsequently used to implement a logical controlled-Hadamard gate, which outperformed its best physical counterpart and provided a clear demonstration of the benefits of FT encoding. On the same trapped-ion platform, a complementary strategy to magic state distillation was demonstrated through code switching [56]. In this approach, a magic state is first prepared in a code that admits a transversal non-Clifford gate, and then transferred into a different code supporting a complementary FT gate set. Specifically, the experiment realized the first code switching protocol between color codes, preparing a magic state in the $[[15, 1, 3]]$ quantum Reed–Muller code, which supports a transversal non-Clifford gate, and then teleporting it into the Steane code, which admits the full Clifford group transver-

sally. The procedure combined verified initialization, transversal logical gates, and a logical teleportation step to transfer the encoded state. The resulting logical magic state reached state-of-the-art fidelity for encoded non-Clifford resources, demonstrating code switching as a practical alternative pathway toward universal FT computation.

Moving beyond topological codes, one of the first experimental realizations of a QLDPC code with long-range interactions was recently demonstrated on a trapped-ion platform [57]. This proof-of-principle experiment implemented a $[[25, 4, 3]]$ QLDPC code, encoding four logical qubits into 25 physical qubits. The platform's all-to-all connectivity enabled nonlocal stabilizer measurements and transversal initialization of logical states. Logical GHZ states were prepared and verified with high fidelity by implementing logical controlled-NOT gates through software-level qubit relabeling, with only mild postselection. All error correction was performed in post-processing using belief propagation plus ordered statistics decoding (BP+OSD) [58], a decoder that combines probabilistic inference with a subsequent processing step involving a matrix inversion to converge on a likely error configuration for the data qubits. Building on the realization of QLDPC codes, a recent experiment on a trapped-ion quantum computer demonstrated the $[[33, 1, 4]]$ 4D surface code [59], marking the first hardware implementation of single-shot QEC. This code benefits from linear dependencies in its stabilizer checks, which provide inherent robustness against syndrome errors and eliminate the need for repeated measurements. Error correction was performed using BP+OSD, demonstrating that single-shot codes can achieve high-fidelity logical operations while minimizing measurement overhead. Finally, preliminary experiments on superconducting platforms have also begun exploring QLDPC codes, despite the challenges posed by limited connectivity [60]. Using a processor with long-range-coupled transmons, the $[[18, 4, 4]]$ bivariate bicycle code was implemented. All nonlocal stabilizers were measured simultaneously via an efficient syndrome extraction circuit, demonstrating the feasibility of executing QLDPC protocols on superconducting hardware. While logical error rates remain above the physical error rates, these results represent an important first step toward realizing low-overhead QEC across different quantum computing platforms.

## 1.3 Motivation and Scope of this Thesis

Motivated by the rapid progress in experimental QEC, this thesis focuses on understanding and optimizing both the theoretical performance and practical implementation of quantum codes. A first key aspect is the performance analysis of stabilizer codes, which is particularly relevant given the variety of physical platforms and the need to understand how logical error rates scale with code parameters under realistic noise. Second, the design of topological codes for asymmetric channels addresses the fact that actual hardware rarely experiences uniform noise: in trapped-ions, quantum dots, and certain superconducting qubits, dephasing errors typically dominate over bit-flip and other errors, requiring codes optimized for biased noise [61, 62]. Third, the development of fast, low-latency decoders is crucial for realizing FT operations in practice, as real-time syndrome processing is an active challenge even for surface codes. Finally, the construction of robust and efficient FT preparation protocols for CSS states is essential, since all practical QEC schemes rely on accurately initialized logical states. These considerations together form the foundation and motivation for the research presented in this thesis.

## 1.4 Thesis Organization

In the next chapters, we explore both the theoretical and practical aspects of QEC, from the performance analysis of stabilizer codes to the implementation of FT state preparation circuits on real hardware.

In Chapter 2, we examine the performance of quantum stabilizer codes, introducing the stabilizer formalism, noise channel models, and the weight enumerator of undetectable errors. This chapter establishes theoretical benchmarks for logical error rates and presents numerical results that guide code evaluation under varying noise conditions [63–65].

Chapter 3 addresses the design of quantum codes tailored for asymmetric channels, where dephasing errors dominate over bit-flip errors. We present constructions such as ZZZY codes, obtained by modifying the structure of surface codes. Moreover, we introduce cylindrical codes and Möbius codes, belonging to

the families of homological product and fiber-bundle codes, and show how channel bias can be exploited to enhance error-correction capability [66, 67].

In Chapter 4, we investigate fast decoders for quantum topological codes, including the spanning tree matching (STM), Rapid-Fire (RFire), and bubble clustering (BC). Each decoder is analyzed in terms of logical error rate, computational complexity, and average execution time under different physical error rates, highlighting their potential for real-time syndrome processing [68–70].

Chapter 5 presents protocols for modular and efficient FT initialization of CSS logical states. We introduce a bipartite preparation circuit for CSS states, FT flag gadgets, and decoding strategies, concluding with the implementation of a logical zero state for the $[[23, 1, 7]]$ Golay code using 237 controlled-NOT gates on the Quantinuum H2 trapped-ion quantum computer [71].

# Chapter 2

# Performance Analysis of Quantum Stabilizer Codes

Estimating the logical error rates is a central task in quantum error correction, as it directly determines the effective reliability of an encoded computation. While Monte Carlo simulations are often employed for this purpose, they become increasingly impractical as the physical error rate decreases: the probability of observing a logical failure diminishes so rapidly that an infeasibly large number of trials would be required to obtain statistically meaningful estimates.

In this chapter, we introduce a framework for analyzing the performance of stabilizer codes using the quantum MacWilliams identities. Our approach overcomes the limitations of simulation by providing rigorous upper bounds on the logical error rate of quantum CSS codes, valid across all physical error rates yet especially informative in the low-noise regime. These bounds are derived from the undetectable errors weight enumerator. We further develop a detailed characterization of logical operators, leading to exact expressions for the asymptotic logical error rate under the assumption of complete decoding, where the asymptotic regime corresponds to sufficiently low physical error rates.

## 2.1 Background on Quantum Error Correction

In this section, we review the fundamental concepts of QEC, focusing first on the stabilizer formalism, which provides a systematic way to describe quantum codes and their error syndromes, and then on the channel models that characterize the types of errors affecting physical qubits.

### 2.1.1 Stabilizer Formalism

A qubit is an element of the two-dimensional Hilbert space $\mathcal{H}^2$, with basis $|0\rangle$ and $|1\rangle$ [22]. The Pauli operators $\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Z}$, and $\boldsymbol{Y}$, are defined by $\boldsymbol{I}|a\rangle = |a\rangle$, $\boldsymbol{X}|a\rangle = |a \oplus 1\rangle$, $\boldsymbol{Z}|a\rangle = (-1)^a|a\rangle$, and $\boldsymbol{Y}|a\rangle = i(-1)^a|a \oplus 1\rangle$ for $a \in \{0, 1\}$. These operators either commute (e.g. $\boldsymbol{IX} = \boldsymbol{XI}$) or anticommute (e.g. $\boldsymbol{XZ} = -\boldsymbol{ZX}$) with each other. Also, apart from an overall factor $\pm 1, \pm i$, the composition of two Pauli produces another Pauli (e.g. $\boldsymbol{XY} = i\boldsymbol{Z}$). Thus, all the Pauli operators, together with multiplicative factors $\pm 1, \pm i$ constitute a group, indicated as $\mathcal{G}_1$. Similarly, all Pauli operators on $n$ qubits together with multiplicative factors $\pm 1, \pm i$ form the $\mathcal{G}_n$ Pauli group [19, 22, 72]. We indicate with $[[n, k, d]]$ a quantum error-correcting code (QECC) with minimum distance $d$, that encodes $k$ information qubits $|\varphi\rangle$ (called logical qubits) into a codeword of $n$ qubits $|\psi\rangle$ (called data or physical qubits), allowing the decoder to correct all patterns up to $t = \lfloor (d-1)/2 \rfloor$ errors (and some patterns of more errors). The codewords will be assumed equiprobable in the following. Using the stabilizer formalism, we start by choosing $n - k$ independent and commuting operators $\boldsymbol{G}_i \in \mathcal{G}_n$, called stabilizer generators. The subgroup of $\mathcal{G}_n$ generated by all combinations of the $\boldsymbol{G}_i \in \mathcal{G}_n$ is a stabilizer, indicated as $\mathcal{S}$. The code $\mathcal{C}$ is the set of quantum states $|\psi\rangle$ stabilized by $\mathcal{S}$, i.e., satisfying $\boldsymbol{S}|\psi\rangle = |\psi\rangle \ \forall \boldsymbol{S} \in \mathcal{S}$, or, equivalently, $\boldsymbol{G}_i|\psi\rangle = |\psi\rangle, \ i = 1, 2, \ldots, n-k$. Operators that commute with every element of the stabilizer group form the normalizer of $\mathcal{S}$ in the Pauli group, denoted $N(\mathcal{S}) = \{P \in \mathcal{G}_n \mid PS = SP \ \forall S \in \mathcal{S}\}$. Logical operators are precisely those elements of $N(\mathcal{S})$ that are not contained in $\mathcal{S}$, i.e., $N(\mathcal{S}) \setminus \mathcal{S}$. They act non-trivially on the encoded logical qubits while preserving the code space. In what follows, logical Pauli operators are denoted by $\boldsymbol{X}_L$ and $\boldsymbol{Z}_L$, corresponding to the logical

bit- and phase-flip operators, respectively.

Assume a codeword $|\psi\rangle \in \mathcal{C}$ is affected by a channel error. Measuring the received state according to the generators $\boldsymbol{G}_i$ with the aid of ancilla qubits, the error is projected, up to logical operators, onto a discrete set of possibilities represented by the Pauli operators $\boldsymbol{E} \in \mathcal{G}_n$ [72]. We call this $\boldsymbol{E}$ a Pauli error. The weight of an error $\boldsymbol{E} \in \mathcal{G}_n$ is the number of single qubits Pauli operators which are not equal to the identity. For example, the error $\boldsymbol{E} = \boldsymbol{X}_2\boldsymbol{Y}_3$ has weight two, with $\boldsymbol{X}$ occurred on the second qubit and $\boldsymbol{Y}$ occurred on the third qubit (we implicitly mean that the others qubits see the Pauli identity operator). The measurement procedure over the ancilla qubits results in a quantum error syndrome $\boldsymbol{s}(\boldsymbol{E}) = (s_1, s_2, \ldots, s_{n-k})$, with each $s_i = 0$ or $1$ depending on $\boldsymbol{E}$ commuting or anticommuting with $\boldsymbol{G}_i$, respectively [72]. In the following, we will often refer to ancillas measuring $s_i = 1$ as *switched on ancillas*, or equivalently as *defects*. Note that an error $\boldsymbol{E} \in \mathcal{S}$ has no effect on a codeword since in this case $\boldsymbol{E}|\psi\rangle = |\psi\rangle$.

CSS codes form a subclass of stabilizer codes in which the stabilizer generators separate into two sets: one consisting only of tensor products of Pauli $\boldsymbol{X}$ operators and the other consisting only of tensor products of Pauli $\boldsymbol{Z}$ operators [23, 24]. Such a code is constructed from two classical binary linear codes $C_X$ and $C_Z$ satisfying the inclusion $C_Z^{\perp} \subseteq C_X$. If $\boldsymbol{H}_X$ and $\boldsymbol{H}_Z$ denote the parity-check matrices of $C_X$ and $C_Z$, respectively, then each row of $\boldsymbol{H}_X$ defines a stabilizer generator composed of $\boldsymbol{X}$ operators, and each row of $\boldsymbol{H}_Z$ defines a stabilizer generator composed of $\boldsymbol{Z}$ operators. This structure enables independent syndrome extraction and correction of bit-flip and phase-flip errors.

In the following, we will also adopt the notation $[[n, k, d_{\mathrm{X}}/d_{\mathrm{Z}}]]$ for asymmetric codes able to correct all patterns up to $t_{\mathrm{X}} = \lfloor (d_{\mathrm{X}} - 1)/2 \rfloor$ Pauli $\boldsymbol{X}$ errors and $t_{\mathrm{Z}} = \lfloor (d_{\mathrm{Z}} - 1)/2 \rfloor$ Pauli $\boldsymbol{Z}$ errors. Moreover, other single- and two-qubit gates considered in this thesis are the Hadamard $\boldsymbol{H}$, phase $\boldsymbol{S}$, and $\boldsymbol{T}$ gates, together with the controlled-NOT $\boldsymbol{CX}$ and controlled-Z $\boldsymbol{CZ}$ gates, defined by $\boldsymbol{H}|a\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + (-1)^a|1\rangle\big)$, $\boldsymbol{S}|a\rangle = i^a|a\rangle$, $\boldsymbol{T}|a\rangle = e^{i\pi a/4}|a\rangle$, $\boldsymbol{CX}|a,b\rangle = |a, a \oplus b\rangle$, and $\boldsymbol{CZ}|a,b\rangle = (-1)^{ab}|a,b\rangle$, for $a, b \in \{0, 1\}$.

### 2.1.2 Channel Models

A possible channel model is one characterized by errors occurring independently and with the same statistic on the individual qubits of each codeword. In this model, the error on each physical qubit can be $\boldsymbol{X}$, $\boldsymbol{Z}$ or $\boldsymbol{Y}$ with probabilities $p_{\mathrm{X}}$, $p_{\mathrm{Z}}$, and $p_{\mathrm{Y}}$, respectively. The *physical error probability*, i.e., the probability of a generic error occurring on a physical qubit, is $p = p_{\mathrm{X}} + p_{\mathrm{Z}} + p_{\mathrm{Y}}$. Two important models are the *depolarizing channel* where $p_{\mathrm{X}} = p_{\mathrm{Z}} = p_{\mathrm{Y}} = p/3$, and the *phase flip channel* where $p = p_{\mathrm{Z}}$, $p_{\mathrm{X}} = p_{\mathrm{Y}} = 0$. We will also consider more general asymmetric channels with the constraint $p_{\mathrm{X}} = p_{\mathrm{Y}}$, therefore completely characterized by the bias parameter $A = 2p_{\mathrm{Z}}/(p - p_{\mathrm{Z}})$. Note that for $A = 1$ we have the depolarizing channel, and for $A \to \infty$ we have the phase flip channel.

This corresponds to a *code-capacity* noise model, where errors affect only the physical qubits and syndrome measurements are assumed to be perfect. More realistic models include the quantum *phenomenological* noise model, in which both data qubits and syndrome measurements are subject to noise, and the quantum *circuit-level* noise model, where errors are associated with all elementary operations in the error-correction circuit, including gates, state preparation, and measurements.

In Chapter 2, a code-capacity noise model is first considered, and the results are then further generalized by including a noisy syndrome extraction circuit within a circuit-level noise model. In Chapter 3, the code-capacity noise model is adopted for the design of the codes, while the derivation of FT syndrome extraction schemes is left for future research. In Chapter 4, we focus on the code-capacity setting in order to benchmark the decoding performance of the proposed decoders independently of the specific FT circuit used for syndrome extraction. Finally, in Chapter 5, a circuit-level noise model is considered for the design of the FT state preparation protocol.

## 2.2 Asymptotic Performance of Quantum Codes

In the following, we will refer to the codeword error probability, $p_{\mathrm{L}}$, as *logical error probability*, defined as the probability that the decoder does not correct the

errors introduced by the quantum channel. Let us first assume an $[[n, k, d]]$ QECC together with a decoder which corrects up to $t = \lfloor (d-1)/2 \rfloor$ generic errors (i.e., $\boldsymbol{X}$, $\boldsymbol{Z}$, or $\boldsymbol{Y}$) per codeword, and no others. For this bounded distance (BD) decoder, the logical error probability is simply

$$p_{\mathrm{L}} = 1 - \sum_{j=0}^{t} \binom{n}{j} p^j \, (1-p)^{n-j} \tag{2.1}$$

that, for $p \ll 1$, can be approximated as

$$p_{\mathrm{L}} \simeq \binom{n}{t+1} p^{t+1} \, . \tag{2.2}$$

The error probability analysis can be generalized to asymmetric QECCs assuming a decoder able to correct up to $e_{\mathrm{g}}$ generic errors plus up to $e_{\mathrm{Z}}$ Pauli $\boldsymbol{Z}$ errors per codeword, and no others. In this case, weighting each pattern with the corresponding probability of occurrence, the BD decoding error rate (2.1) over an asymmetric quantum channel with arbitrary $p_{\mathrm{X}}$, $p_{\mathrm{Y}}$, and $p_{\mathrm{Z}}$ becomes [73]

$$p_{\mathrm{L}} = 1 - \sum_{j=0}^{e_{\mathrm{g}}+e_{\mathrm{Z}}} \binom{n}{j} (1-p)^{n-j} \sum_{i=(j-e_{\mathrm{g}})^+}^{j} \binom{j}{i} p_{\mathrm{Z}}^i \, (p-p_{\mathrm{Z}})^{j-i} \tag{2.3}$$

where $(x)^+ = \max\{x, 0\}$. For a channel with asymmetry parameter $A = 2p_{\mathrm{Z}}/(p - p_{\mathrm{Z}})$, noting that $(j - e_{\mathrm{g}})^+ = 0$ if $j \leq e_{\mathrm{g}}$ and using the binomial theorem, the expression in (2.3) can be rewritten

$$p_{\mathrm{L}} = 1 - \sum_{j=0}^{e_{\mathrm{g}}+e_{\mathrm{Z}}} \alpha_j \binom{n}{j} p^j (1-p)^{n-j} \tag{2.4}$$

where

$$\alpha_j = \begin{cases} 1 & \text{if } j \leq e_{\mathrm{g}} \\ \left(\dfrac{2}{A+2}\right)^j \displaystyle\sum_{i=j-e_{\mathrm{g}}}^{j} \binom{j}{i} \left(\frac{A}{2}\right)^i & \text{otherwise.} \end{cases} \tag{2.5}$$

Similarly to the approximation done in (2.2) derived from (2.1) when $p \ll 1$, considering the most significant terms in (2.4), the asymptotic slope analysis can be extended to asymmetric codes with $e_{\mathrm{Z}} \geqslant 1$ as

$$p_{\mathrm{L}} \simeq \begin{cases} \binom{n}{e_{\mathrm{g}}+1} \left(\frac{2p}{A+2}\right)^{e_{\mathrm{g}}+1} & 1 \leq A < \infty \\ \binom{n}{e_{\mathrm{g}}+e_{\mathrm{Z}}+1} p^{e_{\mathrm{g}}+e_{\mathrm{Z}}+1} & A \to \infty\,. \end{cases} \tag{2.6}$$

Let us now assume to have a decoder that always outputs a codeword (complete decoder). This could allow correcting also some error patterns which are not correctable with BD decoding. Specifically, we analyse the minimum weight (MW) decoder, which can be implemented as the MWPM decoder for surface codes.

**Definition 2.1.** We indicate with $\beta_j$ the fraction of errors of weight $j$ that can be corrected by a complete decoder.

Note that $\beta_j$ depends in general on the code structure, on the decoder, and on the channel asymmetry parameter $A$.

**Definition 2.2** (Error class)**.** We state that two error patterns are in the same class if they have the same Pauli weight with respect to each Pauli operator, i.e., they contain the same number of $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ errors, respectively.

In general, the logical error rate of an error-correcting code of length $n$ is

$$p_{\mathrm{L}} = \sum_{\boldsymbol{E} \in \mathcal{C}_n} \mathbb{P}\{\text{error}|\boldsymbol{E}\}\, \mathbb{P}\{\boldsymbol{E}\} \tag{2.7}$$

where $\mathcal{C}_n$ is the set of all possible error classes over $n$ qubits, $\mathbb{P}\{\text{error}|\boldsymbol{E}\}$ is the probability to have an error given the particular error class $\boldsymbol{E}$, and $\mathbb{P}\{\boldsymbol{E}\}$ is the occurrence probability of $\boldsymbol{E}$. Since the probability $\mathbb{P}\{\text{error}|\boldsymbol{E}\}$ depends only on how many $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ the error $\boldsymbol{E}$ contains, we define $f_j(i,\ell)$ as the fraction of errors of weight $j$ with $i$ Pauli $\boldsymbol{Z}$ and $\ell$ Pauli $\boldsymbol{X}$ errors that are not corrected.

Then, we can write

$$p_{\mathrm{L}} = \sum_{j=1}^{n} \binom{n}{j} (1-p)^{n-j} p^j (1-\beta_j) \tag{2.8}$$

where

$$1-\beta_j = \frac{1}{p^j} \sum_{i=0}^{j} \binom{j}{i} p_{\mathrm{Z}}^i \sum_{\ell=0}^{j-i} \binom{j-i}{\ell} p_{\mathrm{X}}^{\ell} p_{\mathrm{Y}}^{j-i-\ell} f_j(i,\ell)\,. \tag{2.9}$$

Considering a symmetric error correcting code with bounded distance decoding which corrects up to $e_{\mathrm{g}}$ errors, we have that $f_j(i,\ell) = 0$ for $j \leq e_{\mathrm{g}}$ and $f_j(i,\ell) = 1$ otherwise. In the case of an asymmetric code able to correct $e_{\mathrm{g}}$ generic errors plus $e_{\mathrm{Z}}$ Pauli $\boldsymbol{Z}$ errors, $f_j(i,\ell) = 0$ for $j \leq e_{\mathrm{g}}$, $f_j(i,\ell) = 0$ for $e_{\mathrm{g}} < j \leq e_{\mathrm{g}} + e_{\mathrm{z}}$ and $i \geqslant e_{\mathrm{z}}$, and $f_j(i,\ell) = 1$ otherwise. For channels with $p_{\mathrm{X}} = p_{\mathrm{Y}}$ (e.g., depolarizing and asymmetric) we have that

$$1-\beta_j = \frac{1}{(A+2)^j} \sum_{i=0}^{j} A^i \sum_{\ell=0}^{j-i} \binom{j}{i} \binom{j-i}{\ell} f_j(i,\ell) \tag{2.10}$$

and $\beta_j = \beta_j(A)$. The dependence on $A$ will be indicated only when necessary. In the following we will assume $p_{\mathrm{X}} = p_{\mathrm{Y}}$.

For a symmetric code, starting from (2.8) we obtain the upper bound

$$p_{\mathrm{L}} \leqslant (1-\beta_{t+1}) \binom{n}{t+1} p^{t+1} (1-p)^{n-t-1} + \sum_{j=t+2}^{n} \binom{n}{j} (1-p)^{n-j} p^j\,. \tag{2.11}$$

Also, we can approximate the logical error rate for $p \ll 1$ as

$$p_{\mathrm{L}} \approx (1-\beta_{t+1}) \binom{n}{t+1} p^{t+1}\,. \tag{2.12}$$

Note that equation (2.12) differs from (2.2) in that the latter assumes that all errors with weight greater than $t$ are not correctable. The asymptotic slope in log-log domain remains $t+1$, but with an offset depending on $(1-\beta_{t+1})$, compared to (2.2).

In a similar way, we can find the asymptotic error correction capability of an asymmetric code. In particular, for $p \ll 1$ the performance of the code becomes

$$p_{\mathrm{L}} \approx (1-\beta_{e_{\mathrm{g}}+e_{\mathrm{Z}}+1})\binom{n}{e_{\mathrm{g}}+e_{\mathrm{Z}}+1}p^{e_{\mathrm{g}}+e_{\mathrm{Z}}+1} + (1-\beta_{e_{\mathrm{g}}+1})\binom{n}{e_{\mathrm{g}}+1}p^{e_{\mathrm{g}}+1}\,. \tag{2.13}$$

As indicated by (2.8) and its approximations, to calculate the performance of quantum codes we need to determine the fraction of correctable errors $\beta_j$.

## 2.3 Undetectable Errors Weight Enumerator

**Definition 2.3** (Undetectable errors)**.** The undetectable errors operators are those coincident with the logical operators. They transform a codeword into another codeword and are therefore undetectable.

Thus, the set of logical operators coincides with the set of undetectable errors. In the following, we will use the two terms interchangeably.

The *undetectable errors weight enumerator* for a $[[n,k,d]]$ quantum code is

$$L(z) = \sum_{w=0}^{n} L_w z^w \tag{2.14}$$

where $L_w$ is the number of undetectable errors of weight $w$.

Then, we will show that $L(z)$ can be written as

$$L(z) = \frac{1}{2^k}B(z) - \frac{1}{4^k}A(z) \tag{2.15}$$

where $A(z) = \sum_{w=0}^{n} A_w z^w$, $B(z) = \sum_{w=0}^{n} B_w z^w$,

$$A_w = 4^k \sum_{\boldsymbol{E}_w} |\boldsymbol{E}_w \cap \mathcal{S}| \tag{2.16}$$

$$B_w = \frac{1}{2^n}\sum_{\ell=0}^{n}\sum_{s=0}^{w} \binom{\ell}{s}\binom{n-\ell}{w-s}(-1)^s 3^{w-s} A_\ell \tag{2.17}$$

and where the sum in (2.16) is over all operators $\boldsymbol{E}_w \in \mathcal{G}_n$ of weight $w$.

We prove now that (2.15), together with (2.16) and (2.17), gives indeed the undetectable weight enumerator (WE). To this aim, let us consider the operators $\boldsymbol{E}_w \in \mathcal{G}_n$ with weight $w$, i.e., containing exactly $w$ Pauli operators different from the identity. For any two hermitian operators $\boldsymbol{O}_1$ and $\boldsymbol{O}_2$ we can introduce two WEs $A_w$ and $B_w$ [74, 75]

$$A_w(\boldsymbol{O}_1, \boldsymbol{O}_2) = \sum_{\boldsymbol{E}_w} \mathrm{tr}\left(\boldsymbol{E}_w \boldsymbol{O}_1\right) \mathrm{tr}\left(\boldsymbol{E}_w \boldsymbol{O}_2\right) \tag{2.18}$$

$$B_w(\boldsymbol{O}_1, \boldsymbol{O}_2) = \sum_{\boldsymbol{E}_w} \mathrm{tr}\left(\boldsymbol{E}_w \boldsymbol{O}_1 \boldsymbol{E}_w \boldsymbol{O}_2\right) \tag{2.19}$$

where the sum is over all the $\boldsymbol{E}_w$, and $w = 0, \ldots, n$. We will often drop the operators $\boldsymbol{O}_1, \boldsymbol{O}_2$ when the dependence is clear in the context. In the case in which $\boldsymbol{O}_1 = \boldsymbol{O}_2 = \Pi_{\mathcal{C}}$, the projector onto a $[[n, k, d]]$ binary stabilizer code, $A(z)$ and $B(z)$ carry some important properties of the code. Indeed, let $\boldsymbol{G}_i \in \mathcal{S}$, with $i = 1, ..., n-k$, generators of the stabilizer group of a code, then [76]

$$\Pi_{\mathcal{C}} = \frac{1}{2^{n-k}} \prod_{i=1}^{n-k} \left(\boldsymbol{I} + \boldsymbol{G}_i\right) = \frac{1}{2^{n-k}} \sum_{\boldsymbol{S} \in \mathcal{S}} \boldsymbol{S}\,. \tag{2.20}$$

The WE $A(z)$ defined in (2.18) is proportional to the stabilizer WE [76]

$$\frac{1}{4^k} A(z) = \sum_{w=0}^{n} \sum_{\boldsymbol{E}_w} \left|\boldsymbol{E}_w \cap \mathcal{S}\right| z^w. \tag{2.21}$$

Moreover, $B(z)$ is proportional to the normalizer WE [76]

$$\frac{1}{2^k} B(z) = \sum_{w=0}^{n} \sum_{\boldsymbol{E}_w} \left|\boldsymbol{E}_w \cap \mathcal{N}(\mathcal{S})\right| z^w. \tag{2.22}$$

To find the relation between $A_w$ and $B_w$ we write the associated WEs in the form

$$A(v, z) = \sum_{w=0}^{n} A_w v^{n-w} z^w, \quad B(v, z) = \sum_{w=0}^{n} B_w v^{n-w} z^w\,. \tag{2.23}$$

These polynomials are related through the quantum MacWilliam identities [75–77]

$$B(v,z) = A\left(\frac{v+3z}{2}, \frac{v-z}{2}\right). \tag{2.24}$$

Using (2.24) in (2.23) we get (2.17). Finally, the number of undetectable errors of weight $w$ is given by the number of operators of weight $w$ which commute with $\mathcal{S}$, given in (2.22), minus the number of stabilizers of weight $w$, given by (2.21), which leads to (2.15).

The evaluation of the undetectable errors WE (2.15) requires therefore only computing the $A_w$ in (2.16), which can be carried out by direct inspection of $\mathcal{S}$ for small code sizes, or more in general by using the tools for the computation of the weight distribution of classical codes as discussed in Section 2.3.3. More in general, a trivial way to obtain $A(z)$ for a $[[n,k,d]]$ is to compute all linear combinations among the set of generators. Alternatively, it is possible to consider the connection between stabilizer codes and codes over the Galois field $GF(4)$ by identifying the operators $\boldsymbol{I}$, $\boldsymbol{X}$, $\boldsymbol{Z}$ and $\boldsymbol{Y}$ with the four elements of the field [72,78]. Hence, the evaluation of $A(z)$ can be seen as the computation of the weight distribution of classical codes over $GF(4)$. Although this problem may be classified as NP-hard [79], a variety of advanced and optimized algorithms have been developed. These algorithms surpass traditional brute force methods in efficiently calculating key metrics such as the weight distribution and the minimum weight. Some of them are the Brouwer–Zimmermann algorithm and its various modifications for cyclic codes, quasi-cyclic codes, and divisible codes [80–86]. Such algorithms are implemented in software tools related to coding theory, such as MAGMA [87]. Recently, new techniques have been developed for computing the quantum weight enumerator polynomial $A(z)$, which, for some degenerate stabilizer codes, provide up to an exponential speed up compared to the previous methods [88].

### 2.3.1 Bounds on $\beta_{t+1}$ via Undetectable Errors WE

The performance of an $[[n, k, d]]$ QECC is mainly determined, according to (2.11), (2.12), and (2.13), by the value of $\beta_{t+1}$. We show here that, even without analyzing in details the logical operators of a code, it is possible to exploit the undetectable error WE $L(z)$ in order to find upper bounds on the performance for a depolarizing channel. Specifically, considering the general case of a complete MW decoder, we will derive several lower bounds on the value of $\beta_{t+1}$, indicated as $\hat{\beta}_{t+1} \leqslant \beta_{t+1}$, for some families of codes. Unless otherwise stated we will assume $d$ odd.

A first bound, valid in general for stabilizer codes, is obtained assuming that each logical operator of weight $w$ can be caused by all the $3^{t+1}$ different Pauli errors of weight $t+1$, as

$$\hat{\beta}_{t+1} = \left( 1 - \frac{3^{t+1} \sum_{w=2t+1}^{2t+2} L_w \binom{w}{t+1}}{\binom{n}{t+1} 3^{t+1}} \right)^{+} . \tag{2.25}$$

In this equation, $L_w$ is the number of logical operators of weight $w$, $3^{t+1}$ is the number of permutations of the different Pauli operators of weight $t+1$ that cause a logical operator, and $\binom{w}{t+1}$ refers to the number of patterns of errors (made of fixed Pauli operators) with weight $t+1$ that can realize the corresponding logical operator. For instance, if we consider $t+1=2$, we have $3^2 = 9$ different Pauli errors: $\boldsymbol{ZZ}$, $\boldsymbol{XZ}$, $\boldsymbol{ZX}$, $\boldsymbol{XX}$, $\boldsymbol{ZY}$, $\boldsymbol{YZ}$, $\boldsymbol{YY}$, $\boldsymbol{YX}$, $\boldsymbol{XY}$. Moreover, a logical operator of weight $w=3$ can be produced by $\binom{3}{2} = 3$ patterns of each of the previous Pauli errors of weight $t+1$. Note that, the summation in (2.25) starts from $2t+1$ since $L_w = 0$ for $w \leqslant 2t+1$, and is limited to $2t+2$ because, when an error of weight $t+1$ is introduced by the channel, a MW decoder will never choose a codeword which differs in more than $t+1$ positions from the original one. The previous bound can be made more tight if we deal with CSS codes, where the $\boldsymbol{X}$ and $\boldsymbol{Z}$ corrections are performed independently. In this case the logical operators of weight $d$ are formed by one Pauli type only. Consequently, a logical operator may be caused by only two types of Pauli errors. For instance, let us consider the logical operator $\boldsymbol{ZZZ}$: this can be caused only by $\boldsymbol{ZZ}$, $\boldsymbol{ZY}$, $\boldsymbol{YZ}$ and $\boldsymbol{YY}$ Pauli errors. Therefore, for CSS, the number of permutations of Pauli errors that cause the generic logical operator is $2^{t+1}$. Moreover, since, for a MW decoder, the error

recovery operator has always a weight lower or equal to the weight of the channel error pattern, only half of the error patterns of weight $t+1$ cause a logical operator of weight $w = 2t + 2$. Hence, the fraction of corrected errors can be bounded by

$$\hat{\beta}_{t+1}^{\mathrm{CSS}} = \left(1 - \frac{2^{t+1}\sum_{w=2t+1}^{2t+2} L_w \binom{w}{t+1} / \left\lfloor \frac{w}{t+1} \right\rfloor}{\binom{n}{t+1} 3^{t+1}}\right)^{+}. \tag{2.26}$$

We remark that this expression applies also to surface codes with MWPM decoder. Furthermore, it is possible to obtain a tighter bound if the generators are composed by $\boldsymbol{X}$ or $\boldsymbol{Z}$ Pauli measurements on the same qubits. In particular, this condition holds for the category of CSS Dual-Containing codes [38]. These codes have a third of the logical operators of minimum weight composed by only $\boldsymbol{Y}$ Pauli operators. Hence, we know that these logical operators can be caused only by Pauli errors composed by $\boldsymbol{Y}$ operators. In this situation, (2.26) can be rewritten as

$$\hat{\beta}_{t+1}^{\mathrm{CSS-DC}} = \left(1 - \frac{\frac{1}{3}\sum_{w=2t+1}^{2t+2} L_w (2\left(2^{t+1}-1\right)+1) \binom{w}{t+1} / \left\lfloor \frac{w}{t+1} \right\rfloor}{\binom{n}{t+1} 3^{t+1}}\right)^{+}. \tag{2.27}$$

The values provided by (2.25), (2.26), and (2.27), can be used in (2.11), (2.12), and (2.13) to compute upper bounds on the error rate. These new bounds are easy to compute, as they just need the WE polynomial $L(z)$ derived from the MacWilliams identities.

In case we want to have a more precise estimation of the error performance we should get tighter bounds on $\beta_{t+1}$, and this is only possible by a closer analysis of the code. In particular, we should take into account the code degeneracy, which requires a more detailed description of the code logical operators, as discussed in Section 2.3.1. To explain the role of degeneracy, assume we have a code with logical operators that share the same Pauli operators on $t + 1$ common qubits. In the event of an error composed exclusively of these Pauli operators, a deterministic decoder will only trigger one of these logical operators. Therefore, by having knowledge of the structure of the logical operators within a stabilizer code, we could improve the previous bounds. For example, the estimation $\hat{\beta}_{t+1}^{\mathrm{CSS}}$ of (2.26)

can be extended to account for degeneracy as

$$\hat{\beta}_{t+1}^{\mathrm{CSS}} = \left( 1 - \frac{2^{t+1} \sum_{w=2t+1}^{2t+2} L_w \binom{w}{t+1} \gamma_w / \left\lfloor \frac{w}{t+1} \right\rfloor}{\binom{n}{t+1} 3^{t+1}} \right)^+ \tag{2.28}$$

where $\gamma_w \in \left[ 1/\binom{w}{t+1}, 1 \right]$ is the average fraction of potential Pauli error patterns of weight $t+1$ that are not shared between two or more logical operators of weight $w$. Note that this parameter is contingent on both the weight and the distinct Pauli operators that constitute the logical operators. For instance, if the code is asymmetrically degenerate with respect to the different Pauli operators, different logical operators of the same weight could share a different number of Pauli operators. We remark that the bounds (2.25) to (2.27) are already in closed form and do not necessitate the evaluation of $\gamma_w$. Their computational complexity solely arises from evaluating the weight enumerator of the specific quantum code.

The expressions derived in (2.25) to (2.27) can be easily modified to the case where $d$ is even by keeping only the weight $w = 2t+2$ in the summation, and avoiding the division by $\left\lfloor \frac{w}{t+1} \right\rfloor$. For example, for codes with even distance $d = 2t+2$, (2.26) becomes

$$\hat{\beta}_{t+1}^{\mathrm{CSS}} = \left( 1 - \frac{2^{t+1} \; L_{2t+2} \binom{2t+2}{t+1}}{\binom{n}{t+1} 3^{t+1}} \right)^+ . \tag{2.29}$$

Let us provide now examples for some important quantum codes. As regards the $[[7,1,3]]$ Steane code [89], using (2.15) we compute the WE as $L(z) = 21z^3 + 126z^5 + 45z^7$. Hence, by applying (2.27) it is possible to compute $\hat{\beta}_2$ as

$$\hat{\beta}_2^{\mathrm{CSS-DC}} = \left( 1 - \frac{\frac{2}{3} 21 \cdot 3 \cdot 3 + \frac{1}{3} 21 \cdot 3}{\binom{7}{2} 3^2} \right)^+ = \frac{2}{9} \simeq 0.22 \tag{2.30}$$

i.e., at least $22\%$ of the errors of weight $j = 2$ are corrected by a minimum weight decoder. For this particular code this estimate is exact, i.e., $\hat{\beta}_2 = \beta_2$, as will be shown in section 2.3.3. This is because the logical operators of weight $w = 3$ share only $t = 1$ Pauli operators on common qubits. Hence, the code degeneracy does not affect the computation of $\hat{\beta}_2$.

In case of the $[[9,1,3]]$ Shor code [20], from (2.15) the WE results $L(z) = 39z^3 + 208z^5 + 332z^7 + 189z^9$. Moreover, using (2.26) we get $\hat{\beta}_2 = 0$. This result, which gives us no more information with respect to the bounded distance performance, is due to the strong degeneracy of the code. In fact, even if the number of logical operators of weight $w = 3$ is quite large for a 9 qubits code, a lot of them share $t + 1 = 2$ Pauli operators, affecting the actual $\beta_2$ (i.e. $\gamma_3 \ll 1$). An accurate estimation taking into account the effect of degeneracy will be provided in Section 2.3.1 by a counting argument on the logical operators.

Taking into consideration the $[[13,1,3]]$ surface code, from (2.15) we first find $L(z) = 6z^3 + 24z^4 + 75z^5 + 240z^6 + 648z^7 + 1440z^8 + 2538z^9 + 3216z^{10} + 2634z^{11} + 1224z^{12} + 243z^{13}$. Then, considering a MWPM decoder, from (2.26) we have

$$\hat{\beta}_2^{\mathrm{CSS}} = \left(1 - \frac{2^2\left(6\binom{3}{2} + 24\binom{4}{2}/2\right)}{\binom{13}{2}3^2}\right)^+ = \frac{57}{117} \simeq 0.49 \tag{2.31}$$

so that at least $49\%$ of the errors of weight $j = 2$ is corrected.

The method works also for asymmetric CSS codes. For instance, taking the $[[23,1,3/5]]$ surface code, we have $L(z) = 5z^3 + 20z^4 + 51z^5 + 172z^6 + \ldots$ . Considering that errors of weight $j = 2$ can be caused only by $\boldsymbol{X}$ operators, while errors of weight $j = 3$ can be caused both by $\boldsymbol{X}$ and $\boldsymbol{Z}$ operators, from (2.26) we obtain

$$\hat{\beta}_2^{\mathrm{CSS}} = \left(1 - \frac{2^2\left(5\binom{3}{2} + 20\binom{4}{2}/2\right)}{\binom{23}{2}3^2}\right)^+ = \frac{659}{759} \simeq 0.87 \tag{2.32}$$

and

$$\hat{\beta}_3^{\mathrm{CSS}} = \left(1 - \frac{2^3\left(51\binom{5}{3} + 172\binom{6}{3}/2\right)}{\binom{23}{3}3^3}\right)^+ = \frac{17840}{47817} \simeq 0.63\,. \tag{2.33}$$

The exact values of $\beta_{t+1}$ for these codes are provided in Section 2.5. In literature, some lower bounds on the logical error rate of surface codes have been proposed for $p \ll 1$, not for the depolarizing but for the phase flip or the bit flip channels [90, 91]. These bounds are quite far from the true performance, as

they consider only channel errors leading to logical operators of minimum weight $w = 2t + 1$.

### 2.3.2 Closed form expression for the WE of Surface Codes

As evident from (2.26), when dealing with surface codes with MWPM, a channel error with weight $j = t + 1$ has the potential to induce logical operators of weight $w = 2t + 1$ and $w = 2t + 2$, when $d$ is odd. On the other hand, when $d$ is even $\beta_{t+1}$ requires only the knowledge of the number of logical operators of weight $w = 2t + 2$. Consequently, in order to assess the performance bounds of surface codes over a depolarizing channel, it suffices to determine the WE coefficients associated with these two degrees. In the following, we present an expression for determining these values without the necessity of computing the MacWilliams identities. Specifically, given a surface code with minimum distance $d$, there are exactly $2d$ logical operators of weight $w = d$. These operators correspond to straight horizontal $\boldsymbol{Z}$ and vertical $\boldsymbol{X}$ chains crossing the lattice. In fact, all other paths from a boundary to an opposite one have at least $d + 1$ Pauli operators. We conclude that $L_d = 2d$. Furthermore, logical operators of weight $w = d + 1$ can be classified into two distinct groups. The first category, composed by $4(d - 1)^2$ operators, representing the chains starting from a boundary and reaching the opposite side of the lattice with only one turn. Some examples of these $\boldsymbol{Z}_L$ and $\boldsymbol{X}_L$ operators for the $[[13, 1, 3]]$ surface code are depicted in Fig. 2.1a and in Fig. 2.1a′. The final group of logical operators includes some Pauli $\boldsymbol{Y}$ operators. In particular, these logical operators are obtained from the $\boldsymbol{Z}_L$ ($\boldsymbol{X}_L$) operators of weight $w = d$ by applying one site (plaquette) generator. Let us consider firstly a logical operator with $w = d$ that traverse qubits which are measured by four-qubits generators. If we apply a four-qubits generator to these logical operators we end up with another logical operator of weight $w = d + 2$. For this reason, we exclude these from the counting. On the other hand, considering logical operators running along a boundary of the lattice, we can apply a three-qubits generator to end up with another logical operator of weight $w = d + 1$, corresponding to the overall count of three-qubits generators, i.e., $4(d - 1)$. An example of these operators is shown in Fig. 2.1e and in Fig. 2.1f. Counting all the combinations, we conclude

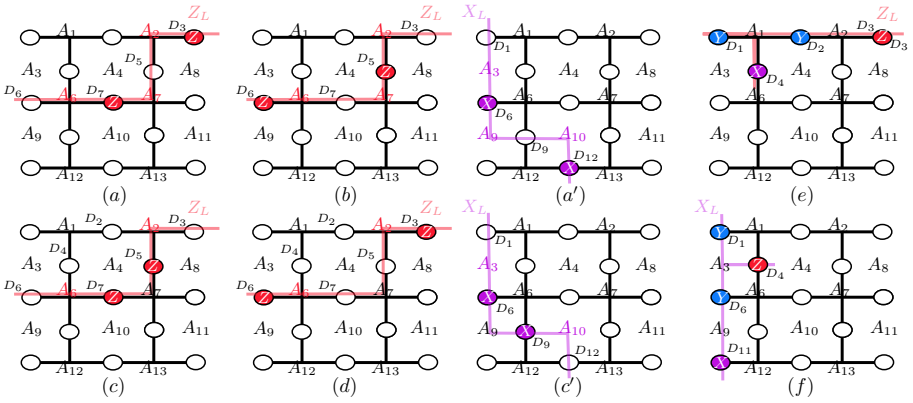


Figure 2.1: Example of errors leading to a logical operator of $w = 4$ for the $[[13, 1, 3]]$ surface code. $\boldsymbol{Z}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ errors on qubits are depicted in red, purple, and blue, respectively. (a) $\boldsymbol{Z}_L$ occurs if $\boldsymbol{Z}$ correction operators are applied on data qubits $D_5$ and $D_6$. The errors are corrected if the MWPM decoder applies $\boldsymbol{Z}$ on $D_3$ and $D_7$. (b) $\boldsymbol{Z}_L$ occurs if $\boldsymbol{Z}$ correction operators are applied on $D_3$ and $D_7$. The errors are corrected if the MWPM decoder applies $\boldsymbol{Z}$ on $D_5$ and $D_6$. (c) $\boldsymbol{Z}_L$ occurs if $\boldsymbol{Z}$ correction operators are applied on $D_3$ and $D_6$. The MWPM decoder could apply also $\boldsymbol{Z}$ operators on $D_5$ and $D_7$, correcting the error, or on $D_2$ and $D_4$, correcting the error. (d) $\boldsymbol{Z}_L$ occurs if $\boldsymbol{Z}$ correction operators are applied on $D_5$ and $D_7$. The MWPM decoder could apply also $\boldsymbol{Z}$ operators on $D_3$ and $D_6$ or on $D_2$ and $D_4$, causing a different logical operator. $(a', c')$ Analogous examples for $\boldsymbol{X}_L$ logical operator. $(e, f)$ Logical operators of $w = 4$ caused by $\boldsymbol{YY}$ errors.

that $L_{d+1} = 4d(d-1)$.

In summary, a $[[d^2 + (d-1)^2, 1, d]]$ surface code has WE

$$L_d = 2d \qquad L_{d+1} = 4d(d-1) \tag{2.34}$$

which can be used in (2.26) and (2.29). For example, for the $[[41, 1, 5]]$ surface code $L_5 = 10$ and $L_6 = 80$, resulting in $\hat{\beta}_3 = 0.975$.

### 2.3.3 Logical Operators Analysis

Now we show that it is possible to obtain the exact $\beta_j$ parameters with no need of simulations. This requires an analysis of the structure of the code logical operators, which we do explicitly for some small-size codes. The same approach may become too complicated for large codes, where the use of the closed-form bounds (2.26), and (2.27) is preferable. Since we are interested in the value of $\beta_j$, we have

to find the fraction of errors of weight $j$ that can cause the decoder to miscorrect and induce a logical error. In order to compute it, we will consider not only $L(z)$ but also the structure of the stabilizer.

In the case of a CSS stabilizer code, taking into account that the total number of different error patterns of weight $j$ is $\binom{n}{j}$, we can express $\beta_j$ as

$$\beta_j = 1 - \frac{\sum_{i=0}^{j} A^i \sum_{\ell=0}^{j-i} \sum_w L_w(i,\ell)\, \mu_j^{(w)}(i,\ell)\, \gamma_w(i,\ell)}{(A+2)^j \binom{n}{j}} \tag{2.35}$$

where: $L_w(i,\ell)$ stands for the number of logical operators of weight $w$ that can be caused by the combined action of channel errors of weight $j$ composed by $i$ $\boldsymbol{Z}$ and $\ell$ $\boldsymbol{X}$ Pauli operators and $w-j$ correction operators applied by a complete decoder; $\mu_j^{(w)}(i,\ell)$ refers to the number of different patterns of errors with weight $j$ that can induce the corresponding logical operator of weight $w$; and $\gamma_w(i,\ell) \in \left[1/\binom{w}{j}, 1\right]$ is the average fraction of potential Pauli error patterns of weight $j$ that are not shared between two or more logical operators. Note that (2.35) and (2.10) are equivalent, since

$$\sum_w L_w(i,\ell)\, \mu_j^{(w)}(i,\ell)\, \gamma_w(i,\ell) = \binom{n}{j}\binom{j}{i}\binom{j-i}{\ell} f_j(i,\ell). \tag{2.36}$$

In particular, on both sides of (2.36) we find the total number of logical errors induced by the weight $j$ error class identified by $i$ operators of type $\boldsymbol{Z}$ and $\ell$ operators of type $\boldsymbol{X}$.

In the following, we provide some examples over a depolarizing channel, in order to clarify the reasoning behind the evaluation of $\beta_j$. First, let us consider the $[[5,1,3]]$ perfect code [92]. Note that this is not a CSS code. Using (2.15) we obtain $L(z) = 30z^3 + 18z^5$, so we know that the number of logical operators of weight $3$ is $30$. Then, we have

$$\beta_2 = 1 - \frac{30 \cdot 3}{\binom{5}{2} 3^2} = 0 \tag{2.37}$$

where we have considered that the total number of pairs of Pauli errors is $\binom{5}{2}3^2$, and that each logical operator of weight $3$ can be caused exactly by three different

combinations of errors of weight $j = 2$. This result was expected since the code is perfect.

*Steane code:* A more interesting case is that of the $[[7, 1, 3]]$ Steane code [89]. Starting from $L(z) = 21z^3 + 126z^5 + 45z^7$ and assuming a MW decoder, we calculate

$$\begin{aligned}\beta_2 &= 1 - \frac{1}{\binom{7}{2}3^2}\left[7\mu_2^{(3)}(2,0) + 7\mu_2^{(3)}(1,0) + 7\mu_2^{(3)}(0,2) + 7\mu_2^{(3)}(0,1) + 7\mu_2^{(3)}(0,0)\right] \\ &= 1 - \frac{7\,(3+6) + 7\,(3+6) + 7\cdot 3}{\binom{7}{2}3^2} = \frac{2}{9} \simeq 0.22. \end{aligned} \tag{2.38}$$

To derive this value we observed that, unlike the perfect code, each logical operator of $w = 3$ is composed of only one kind of Pauli operator. Specifically, we have seven $\boldsymbol{XXX}$, seven $\boldsymbol{ZZZ}$, and seven $\boldsymbol{YYY}$ logical operators. In addition, $\gamma_3 = 1$ since there are no logical operators that share Pauli operators on the same qubits. Moreover, we have to consider that each one of these $\boldsymbol{X}$ ($\boldsymbol{Z}$) logical operators can be generated by $\boldsymbol{ZZ}$ ($\boldsymbol{XX}$) errors, and also by $\boldsymbol{XY}$ ($\boldsymbol{ZY}$), since single errors can always be corrected, while $\boldsymbol{Y}$ logical operators are caused only by two $\boldsymbol{Y}$ errors.

*Shor code:* As a third example let us analyze the $[[9, 1, 3]]$ Shor code [20]. From (2.15) the WE results $L(z) = 39z^3 + 208z^5 + 332z^7 + 189z^9$. Assuming a MW decoder, 39 undetectable errors of weight three are caused by channel errors of weight two. A closer look reveals that the 39 operators include three $\boldsymbol{XXX}$, 27 $\boldsymbol{ZZZ}$, and 9 $\boldsymbol{YYX}$ logical operators. The code is degenerate with respect to $\boldsymbol{Z}$ Pauli errors, so it is necessary to treat differently the logical operators of each type. In particular, since each channel error of the kind $\boldsymbol{ZZ}$ can cause three different $\boldsymbol{ZZZ}$ logical operators, $\gamma_3(2,0) = \gamma_3(1,0) = \frac{1}{3}$. For instance, the channel error $\boldsymbol{Z}_1\boldsymbol{Y}_4$ could cause either $\boldsymbol{Z}_1\boldsymbol{Z}_4\boldsymbol{Z}_7$, $\boldsymbol{Z}_1\boldsymbol{Z}_4\boldsymbol{Z}_8$, or $\boldsymbol{Z}_1\boldsymbol{Z}_4\boldsymbol{Z}_9$. (The error $\boldsymbol{X}_4$ is corrected by the decoder). Thus, considering that $\gamma_3(0,0) = \gamma_3(0,1) = \gamma_3(0,2) =$

1, we obtain

$$\begin{aligned}\beta_2 &= 1 - \frac{1}{\binom{9}{2}3^2}\Big[27\,[\mu_2^{(3)}(2,0) + \mu_2^{(3)}(1,0)]\,\gamma_3(2,0) \\ &\quad + 3\,[\mu_2^{(3)}(0,2) + \mu_2^{(3)}(0,1)] + (27+9)\,\mu_2^{(3)}(0,0)\Big] \\ &= 1 - \frac{27\,(3+6)\,\frac{1}{3} + 3\cdot 3 + 3\cdot 6 + 27 + 9}{\binom{9}{2}3^2} = \frac{5}{9} \simeq 0.56 \end{aligned} \tag{2.39}$$

*Surface codes:* We investigate the $\beta_j$ values for surface codes, which belong to the class of CSS codes, assuming MWPM decoding [27, 93, 94]. Considering the $[[13, 1, 3]]$ surface code, we have $L(z) = 6z^3 + 24z^4 + 75z^5 + 240z^6 + 648z^7 + 1440z^8 + 2538z^9 + 3216z^{10} + 2634z^{11} + 1224z^{12} + 243z^{13}$. As before, we need to analyze how channel errors of weight $j = 2$ cause logical operators. Let us first look at the six logical operators of weight three. For the surface codes we know that the logical operators cross the lattice from side to side. Thus, there are three $\boldsymbol{XXX}$ (crossing horizontally) and three $\boldsymbol{ZZZ}$ (crossing vertically) logical operators. As a result, $\mu_2^{(3)}(0,2) = \mu_2^{(3)}(2,0) = \mu_2^{(3)}(0,0) = 3$, given that, for instance, a $\boldsymbol{ZZZ}$ logical operator can arise from $\binom{3}{2}$ error patterns of the $\boldsymbol{ZZ}$ type. Furthermore, $\mu_2^{(3)}(0,1) = \mu_2^{(3)}(1,0) = 6$, as a $\boldsymbol{ZZZ}$ logical operator is induced by $2\binom{3}{2}$ error patterns of the $\boldsymbol{ZY}$ kind. In this code we have also logical operators of weight $w = 4$, as illustrated in Fig. 2.1. Specifically, among the $L_4 = 24$ logical operators, 16 are composed by $\boldsymbol{XXXX}$ and $\boldsymbol{ZZZZ}$, and the remaining eight are of type $\boldsymbol{YYXZ}$, as illustrated in Fig. 2.1e and in Fig. 2.1f. For these eight cases, we are left, after MWPM decoding, with a logical operator with three $\boldsymbol{Z}$ or three $\boldsymbol{X}$, which have been already counted when discussing the weight $w = 3$. In particular, for each of the 16 logical operators of weight $w = 4$ there are $\binom{w}{j} = \binom{4}{2} = 6$ different patterns of errors of weight $j = 2$ that can cause it. However, one of them is always corrected (due to the degeneracy of the code), while another one cause a logical operator of weight $w = 3$ that we have already taken into consideration. For instance, with reference to the logical operator shown in Fig. 2.1a, the error pattern $\boldsymbol{Z}_3\boldsymbol{Z}_5$ switches on the ancilla $A_7$, which leads to the correction operator $\boldsymbol{Z}_8$, thereby realizing the stabilizer generator $\boldsymbol{Z}_3\boldsymbol{Z}_5\boldsymbol{Z}_8$. Moreover, the error $\boldsymbol{Z}_6\boldsymbol{Z}_7$ switches on the ancilla $A_7$, and

Table 2.1: Coefficients for performance evaluation, $[[13, 1, 3]]$ surface code.

| | $i=2$ $\ell=0$ | $i=1$ $\ell=0$ | $i=0$ $\ell=0$ | $i=0$ $\ell=1$ | $i=0$ $\ell=2$ |
|---|---|---|---|---|---|
| $L_3, \gamma_3$ | $3, 1$ | $3, 1$ | $6, 1$ | $3, 1$ | $3, 1$ |
| $\mu_2^{(3)}$ | 3 | 6 | 3 | 6 | 3 |
| $L_4, \gamma_4$ | $8, 3/4$ | $8, 3/4$ | $16, 3/4$ | $8, 3/4$ | $8, 3/4$ |
| $\mu_2^{(4)}$ | 2 | 4 | 2 | 4 | 2 |

the resulting correction operator produces the logical operator $\boldsymbol{Z}_6\boldsymbol{Z}_7\boldsymbol{Z}_8$ of weight $w = 3$. The remaining four error patterns, shown in Fig. 2.1a–d, produce the same syndrome in pairs. We consider a deterministic decoder, such as the MWPM, that associate one error pattern to each syndrome. Thus, only two patterns will not be corrected. Hence, we have $\mu_2^{(4)}(2, 0) = \mu_2^{(4)}(0, 2) = \mu_2^{(4)}(0, 0) = 2$, while $\mu_2^{(4)}(1, 0) = \mu_2^{(4)}(0, 1) = 4$. Also, note that, among the four patterns of errors of weight $j = 2$ that can cause a logical operator of weight $w = 4$, one is in common with another logical operator. For example, the error pattern $\boldsymbol{Z}_3\boldsymbol{Z}_6$ depicted in Fig. 2.1d may also cause the logical operator $\boldsymbol{Z}_2\boldsymbol{Z}_3\boldsymbol{Z}_4\boldsymbol{Z}_6$. Hence, among the four faulty error patterns corresponding to the pair of logical operators, one pattern is repeated twice. Therefore we have $\gamma_4(i, \ell) = \frac{3}{4}$. In Tab. 2.1, we report, for the $[[13, 1, 3]]$ surface code, the values of $L(z)$, $\gamma_w(i, \ell)$, and $\mu_j^{(w)}(i, \ell)$ that are needed in order to compute $\beta_2$. If we put these parameters into (2.35), we obtain

$$\beta_2 = \frac{267}{351} \simeq 0.76. \tag{2.40}$$

Using (2.35) it is also possible to obtain the value of $\beta_j$ for asymmetric channels. For instance, in the case of the [[13,1,3]] surface code over a phase flip channel, we have only three $\boldsymbol{Z}$ logical operators with $w = 3$ and eight $\boldsymbol{Z}$ logical operators with $w = 4$, giving

$$\beta_2 = 1 - \frac{3\mu_2^{(3)}(0, 2) + 8\mu_2^{(4)}(0, 2)\gamma_4(0, 2)}{\binom{13}{2}} = 1 - \frac{3 \cdot 3 + 8 \cdot 2\frac{3}{4}}{\binom{13}{2}} = \frac{19}{26} \simeq 0.73\,. \tag{2.41}$$

*Rotated surface codes:* We derive the WE function for the $[[9, 1, 3]]$ rotated surface code [95] as $L(z) = 24z^3 + 192z^5 + 408z^7 + 144z^9$. In this case, we have 24

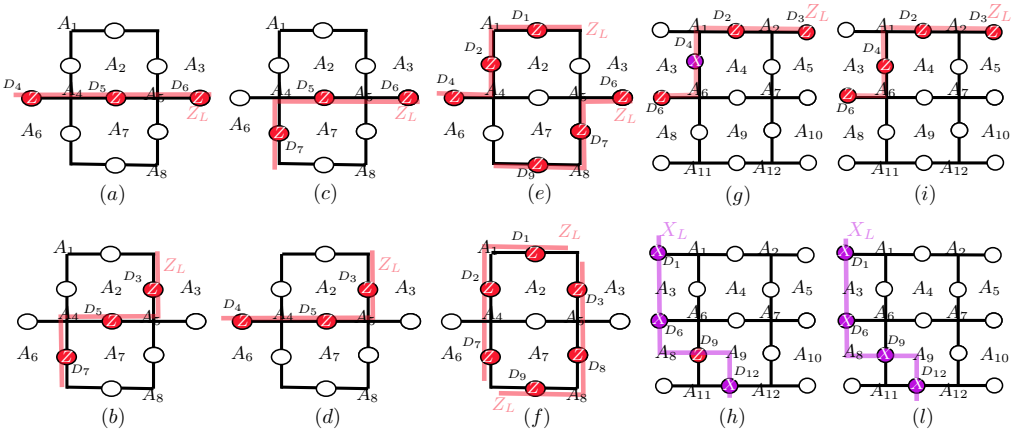

Figure 2.2: (a-f) Examples of $\boldsymbol{Z}$ logical operators of weight $w = 3$ of the $[[9, 1, 3]]$ rotated surface code. Data qubits with a $\boldsymbol{Z}_L$ error are depicted in red. The last four $\boldsymbol{Z}$ logical operators have the form of (e) and (f) but with $\boldsymbol{Y}$ operators on qubits $D_1$, $D_2$, $D_8$ and $D_9$. The other 12 $\boldsymbol{X}$ logical have the same structure on the dual lattice. (g,h) Examples of $\boldsymbol{Z}_L$ and $\boldsymbol{X}_L$ logical operators of weight $w = 4$ for the $[[13, 1, 3]]$ XZZX code. (i,l) Examples of $\boldsymbol{Z}_L$ and $\boldsymbol{X}_L$ logical operators of weight $w = 4$ for the $[[13, 1, 3]]$ surface code.

logical operators of weight $w = 3$ and zero of weight $w = 4$. However, as for the $[[13, 1, 3]]$ surface code when $w = 4$, we consider only 16 out of 24 logical operators, due to the fact that we are implicitly counting the 8 operators having both Pauli $\boldsymbol{Z}$ and $\boldsymbol{X}$ into the 16 ones. Focusing on $\boldsymbol{Y}_1\boldsymbol{Y}_2\boldsymbol{Z}_4$ in the code of Fig. 2.2e, consider the channel error $\boldsymbol{Y}_1\boldsymbol{Y}_2$. In this case, ancilla $A_4$ is switched on and the decoder applies the correction $\boldsymbol{Z}_4$, resulting in the logical operator $\boldsymbol{Z}_1\boldsymbol{Z}_2\boldsymbol{Z}_4$, since $\boldsymbol{X}_1\boldsymbol{X}_2$ is a stabilizer. Let us now consider the eight operators constituted of only Pauli $\boldsymbol{Z}$, reported pictorially in Fig. 2.2a-f. As for the $[[13, 1, 3]]$ surface code, $\mu_2^{(3)}(0, 2) = \mu_2^{(3)}(2, 0) = \mu_2^{(3)}(0, 0) = 3$, while $\mu_2^{(3)}(0, 1) = \mu_2^{(3)}(1, 0) = 6$. We are interested in the number of $\boldsymbol{Z}$ errors of weight $j = 2$ that can cause these logical operators. However, we can see that different logical operators share common patterns of $\boldsymbol{Z}$ errors with weight $j = 2$ and must be considered only once. For instance, the operator in Fig. 2.2a can be caused by three different patterns of $\boldsymbol{Z}$ errors: $\boldsymbol{Z}_4\boldsymbol{Z}_5$, $\boldsymbol{Z}_4\boldsymbol{Z}_6$ and $\boldsymbol{Z}_5\boldsymbol{Z}_6$. In particular, $\boldsymbol{Z}_4\boldsymbol{Z}_5$ can cause either $(a)$ or $(d)$, while $\boldsymbol{Z}_5\boldsymbol{Z}_6$ can lead either to $(a)$ or to $(c)$, depending on the MWPM implementation. Specifically, each logical operator of the kind (a-d) can be caused by two patterns of Pauli errors of weight $j = 2$ which are in common with others.

Table 2.2: Coefficients for performance evaluation, $[[9,1,3]]$ surface code.

| | $i=2$ $\ell=0$ | $i=1$ $\ell=0$ | $i=0$ $\ell=0$ | $i=0$ $\ell=1$ | $i=0$ $\ell=2$ |
|---|---|---|---|---|---|
| $L_3, \gamma_3$ | $8, \frac{3}{4}$ | $8, \frac{3}{4}$ | $16, \frac{3}{4}$ | $8, \frac{3}{4}$ | $8, \frac{3}{4}$ |
| $\mu_2^{(3)}$ | 3 | 6 | 3 | 6 | 3 |

On average, each of these four operators can be caused by $\binom{3}{2} - 1 = 2$ different patterns of errors of weight $j = 2$. As regards the four logical operators in Fig. 2.2e and Fig. 2.2g, they have just one pattern of errors of weight $j = 2$ in common with each other. Hence, considering them in pairs, we must take into account $2 \cdot \binom{3}{2} - 1 = 5$ different pattern of errors per couple. The resulting value of $\gamma_3(i,\ell) = (4\frac{2}{3} + 4\frac{5}{6})\frac{1}{8} = \frac{3}{4}$. In Tab. 2.2, we report, for the $[[9,1,3]]$ surface code, the values of $L(z)$, $\gamma_w(i,\ell)$, and $\mu_j^{(w)}(i,\ell)$ that are needed in order to compute $\beta_2$. If we put these parameters into (2.35), we obtain

$$\beta_2 = \frac{5}{9} \simeq 0.56. \tag{2.42}$$

It is also possible to derive $\beta_2$ over a phase flip channel, observing that in this case we have only $\boldsymbol{Z}$ Pauli errors. This leads for the same $[[9,1,3]]$ rotated surface code over a phase flip channel to

$$\beta_2 = 1 - \frac{3\mu_2^{(3)}(0,2)\gamma_3(0,2)}{\binom{9}{2}} = 1 - \frac{3 \cdot 8\frac{3}{4}}{\binom{9}{2}} = \frac{1}{2} \simeq 0.5\,. \tag{2.43}$$

*XZZX surface codes:* $XZZX$ codes have a similar structure to surface codes; however, while standard surface code stabilizers consist only of $\boldsymbol{X}$ or $\boldsymbol{Z}$ operators, the stabilizers in $XZZX$ codes take the form $\boldsymbol{XZZX}$ within the lattice [30]. As a result, $XZZX$ codes are not CSS codes, and the previous analysis does not directly apply. Nonetheless, a similar logical operator analysis can be carried out to compute the fraction of faulty error patterns of weight $t+1$. The value of $L(z)$ for the $[[13,1,3]]$ XZZX code is $L(z) = 6z^3 + 24z^4 + 75z^5 + 240z^6 + 648z^7 + 1440z^8 + 2538z^9 + 3216z^{10} + 2634z^{11} + 1224z^{12} + 243z^{13}$. As anticipated, this is the same as that obtained for the original surface code. Indeed, the logical operators have the same structure but they are composed of different Pauli operators. In

particular, notice that, even if the logical operators differ for some Pauli operators with respect to the surface codes, a decoder will make its decision based only on the resulting syndrome. As a result, over the depolarizing channel and for a MWPM decoder, for the XZZX variant of the $[[13, 1, 3]]$ code, we still have $\beta_2 \simeq 0.76$, as can be checked by deriving it in the same way as (2.40).

Moving to the analysis over asymmetric channels, we note that the 16 weight-$w = 4$ logical operators considered in the $[[13, 1, 3]]$ surface code now consist of both $\boldsymbol{X}$ and $\boldsymbol{Z}$ Pauli operators, as illustrated in Fig. 2.2g–h. As a consequence, some of these operators occur only in the presence of both $\boldsymbol{Z}$ and $\boldsymbol{X}$ errors. Considering an asymmetric channel where $\boldsymbol{X}$ errors are less probable than $\boldsymbol{Z}$ errors, it follows that logical operators of the kind shown in Fig. 2.2g are, on average, less probable to occur. This is the reason behind the XZZX performance advantage over asymmetric channels. For example, letting $A \to \infty$ (i.e., assuming phase flip channel), we have four error patterns of the type $\boldsymbol{ZX}$ that we have to exclude, since $\boldsymbol{X}$ errors cannot occur on this channel. Hence, over the phase-flip channel, the XZZX variant leads to

$$\beta_2 = 1 - \frac{3 \cdot 3 + 8 \cdot 2\frac{3}{4} - 4}{\binom{13}{2}} = \frac{61}{78} \simeq 0.782. \tag{2.44}$$

This shows how the error correction capability of XZZX codes increases with the asymmetry of the channel.

*Rotated XZZX codes:* We can compute for the $[[9, 1, 3]]$ rotated XZZX code the undetectable error WE, obtaining $L(z) = 24z^3 + 192z^5 + 408z^7 + 144z^9$. This, as expected, is the same as the non-XZZX version. For the depolariziong channel, the resulting $\beta_2$ is, therefore, equal to (2.42) computed for the $[[9, 1, 3]]$ rotated surface code. Thus, the XZZX variant does not change the performance over the depolarizing channel. However, the codes perform differently on asymmetric channels. In fact, let us now consider the 18 configurations of errors with weight $j = 2$ of (2.45), which, in the rotated surface code, correspond to only $\boldsymbol{Z}$ Pauli errors. As for non-rotated codes, it can be shown that 10 of these configurations include one $\boldsymbol{X}$ Pauli error in the case of the $[[9, 1, 3]]$ rotated XZZX code. Hence, this code has a great advantage over asymmetric channels. For example,

we compute for it

$$\beta_2 = 1 - \frac{4 \cdot 2 + 2 \cdot 5 - 10}{\binom{9}{2}} = \frac{7}{9} \simeq 0.778 \tag{2.45}$$

over a phase flip channel, to be compared with $\beta_2 = 1/2$ of the non-XZZX code given in (2.43).

## 2.4 Noisy Syndrome Measurement

In realistic quantum systems, syndrome measurements are inherently noisy and often require repeated execution over time to yield reliable outcomes. In this section, we shift our attention to syndrome extraction circuits and introduce a framework that accounts for the presence of noise during measurement. We recall that a gadget for a specific function is defined as a circuit to perform that function on the encoded state [72]. Then, we replace each measurement in the original circuit with a FT gadget that replicates its intended action on logical qubits, ensuring it behaves as the ideal measurement would in the unencoded circuit. By systematically propagating error probabilities through the circuit, we derive upper bounds on the resulting physical error rate. When combined with the previously established theoretical bounds, this approach enables us to estimate the logical error rate including the effects of the noisy gadget, offering a practical tool for analyzing performance in real-world quantum error correction scenarios. Specifically, in realistic syndrome extraction gadgets, two main challenges arise. First, syndrome measurements are inherently noisy, making a single-shot measurement unreliable [21, 72]. This issue is typically addressed by repeating the measurement until the same syndrome is obtained $t+1$ times consecutively [21]. In many practical scenarios, for distance-$d$ codes, it is common to perform $d$ repeated measurements [22,27]. The second challenge stems from error propagation during the extraction process. In particular, errors on the syndrome qubit can spread to multiple data qubits through entangling gates, leading to high-weight errors known as hook errors. To mitigate these effects, techniques such as the use of cat states or flag qubits have been developed, which can signal the presence of correlated faults [21, 96, 97]. An alternative is the Steane error correction gadget, which, by

construction, implements a fault-tolerant gate through transversal operations and destructive measurement of a logical ancilla [98]. In the following, we consider realistic syndrome extraction gadgets that include faulty initialization, single-qubit and two-qubit gates, and measurements. We assume that after each of these operations, a depolarizing channel acts on the qubits involved, with error probabilities specific to each type of gate. In addition to these, we also include an initial depolarizing channel acting on each data qubit before the syndrome extraction gadget is applied. Moreover, in practical quantum experiments, error correction is executed over multiple consecutive correction cycles [49, 51]. Between cycles, data qubits may accumulate errors due to decoherence or as a result of gate operations performed during encoded computation. Importantly, errors arising from an incorrect syndrome bit in a given cycle, as well as those introduced by the final two-qubit gate during syndrome extraction (which are not detected within the same cycle), can be dealt with in subsequent cycles.

### 2.4.1 Cat States

In this approach, the stabilizer generator $\boldsymbol{G}_i$ of weight $\gamma_i$ is measured using a cat state (or GHZ state) composed of $\gamma_i$ ancilla qubits. This configuration ensures that each controlled gate in the extraction circuit interacts with a dedicated syndrome qubit, thereby limiting the spread of errors [21]. The entire syndrome extraction procedure is assumed to be repeated for $r > t$ shots [21, 72]. The error probability of a specific syndrome bit can be bounded by considering that $t + 1$ consecutive syndrome extractions providing the same syndrome, i.e., no errors have occurred [21]. This is

$$P_i \leq 1 - (1 - p_i)^{t+1} \tag{2.46}$$

where $p_i$ is the error probability in a single shot syndrome bit measurement. This can be upper bounded as

$$\begin{aligned} p_i \leqslant 1 - (1 - \rho_{\text{init}}^{\text{cat}}) \Big[ (1 - \rho_{1Q}) \\ \times (1 - \rho_{2Q})(1 - \rho_{\text{meas}}) \Big]^{\gamma_i} \end{aligned} \tag{2.47}$$

with $\rho_{\text{init}}^{\text{cat}}$, $\rho_{1Q}$, $\rho_{2Q}$, and $\rho_{\text{meas}}$ denote the depolarizing error probabilities associated with the cat state initialization, single-qubit gates, two-qubit gates, and qubit measurement, respectively. Moreover, note that a MW decoder can apply corrections to at most $t$ qubits. As a result, a single syndrome bit error can, in the worst case, be equivalent to a Pauli error affecting $t$ qubits. For a given stabilizer generator, the probability that a physical qubit is affected due to a faulty syndrome bit is $t/n$, where $n$ is the total number of qubits. Therefore, the overall qubit error probability resulting from faulty syndrome measurements can be upper bounded as

$$P_{\text{syn}} \leqslant \frac{t}{n}\left[1 - \prod_{i=1}^{n-k}(1 - P_i)\right]. \tag{2.48}$$

To compute the depolarizing error probability after syndrome extraction, we also account for an initial depolarizing channel on each data qubit, as well as a depolarizing channel associated with each two-qubit gate that interacts with a data qubit. Since we aim to obtain an upper bound for the depolarizing error probability, which is assumed to be the same for every data qubit, we are interested in the maximum number of two-qubit gates applied to any single qubit. Thus, we define $\delta_{\max} = \max_j \delta_j$, where $\delta_j$ represents the number of two-qubit gates applied to qubit $j$ in a single round of syndrome extraction. Hence, the equivalent depolarizing error probability on each qubit at the end of the measurement based on cat state can be upper bounded as

$$\begin{aligned}\rho_{\text{eq}}^{\text{cat}} \leqslant\ & 1-(1-\rho)(1-P_{\text{syn}}) \\ & \times \left[(1-\rho_{2Q})(1-\rho_{\text{init}}^{\text{cat}})\right]^{r\,\delta_{\max}}\end{aligned} \tag{2.49}$$

where $\rho$ is the depolarizing error probability on each qubit before the gadget, and $P_{\text{syn}}$ is the faulty syndrome error probability due to the previous error correction cycle. Finally, an upper bound on the logical error rate after the gadget can be obtained by equation (2.11) with $\rho$ replaced with $\rho_{\text{eq}}^{\text{cat}}$ from equation (2.49), in conjunction with the results from Section 2.3.

### 2.4.2 Flag Error Detection

In this scenario, a single ancillary qubit is assigned to each syndrome bit. However, to mitigate hook errors, additional flag qubits can be introduced [96, 97]. These flag qubits are designed to propagate any hook errors, and their subsequent measurements allow for the detection or the correction of such errors. Upon detection, the computation can be post-selected and restarted, as an example, to maintain the integrity of the syndrome extraction process. Error detection and correction flag circuits used for measuring syndrome bits in codes of distance three and five are presented in [97, 99]. Here, the probability of syndrome error per single measurement can be upper bounded as

$$\begin{aligned} p_i \leqslant 1 - (1-\rho_{\text{init}})(1-\rho_{1Q})^2 \\ \times (1-\rho_{2Q})^{\gamma_i+2f_i}(1-\rho_{\text{meas}}) \end{aligned} \tag{2.50}$$

where $f_i$ denotes the number of flag qubits required for the $i-th$ generator, and $\rho_{\text{init}}$ represents the depolarizing error probability associated with qubit initialization. Note that each flag qubit is associated with two two-qubit gates involving the syndrome qubit. Also, we can compute $P_{\text{syn}}$ using (2.50) in (2.46) and (2.48). In this case, the equivalent depolarizing error probability per qubit at the end of the gadget $\rho_{\text{eq}}^{\text{flag}}$ is

$$\rho_{\text{eq}}^{\text{flag}} \leqslant 1 - (1-\rho)(1-P_{\text{syn}})(1-\rho_{2Q})^{r\,\delta_{\max}} \tag{2.51}$$

where $\rho$ is the depolarizing error probability on each qubit before the gadget, and $P_{\text{syn}}$ is the faulty syndrome error probability due to the previous error correction cycle.

Moreover, we must account for the possibility that a flag gadget might fail to detect a hook error, for instance, due to an additional error occurring during its measurement. Since we are computing an upper bound, we assume that such a failure results in a logical error. Specifically, the error probability in a flag gadget

can be upper bounded as

$$p_{\text{flag}} \leqslant 1-(1-\rho_{\text{init}})(1-\rho_{1Q})^2 \times (1-\rho_{2Q})^2(1-\rho_{\text{meas}}) \tag{2.52}$$

Given a generator $i$, with $1 \leqslant i \leqslant n-k$, and a specific two-qubit gate $c$, with $1 \leqslant c \leqslant \gamma_i$, used for measuring that generator, we define $F_{i,c}$ as the number of flag qubits that, in the absence of errors, are capable of detecting a hook error resulting from that particular two-qubit gate. This corresponds to all the flag qubits that have been entangled with the syndrome qubit before the two-qubit gate $c$, but have not yet been disentangled at the time $c$ is applied. Then, the probability that an hook error is undetected is upper bounded by

$$P_{\text{un,hook}} \leqslant \rho_{2Q} \sum_{i=1}^{n-k} \sum_{c=2}^{\gamma_i-2} p_{\text{flag}}^{F_{i,c}}. \tag{2.53}$$

Note that the inner summation excludes the first and last two two-qubit gates of the generator, as these cannot give rise to hook errors (up to a stabilizer generator). Finally, an upper bound on the logical error rate can be computed as

$$\rho_L' \leqslant \rho_L + P_{\text{un,hook}} - \rho_L P_{\text{un,hook}} \tag{2.54}$$

where $\rho_L$ is given by equation (2.11) with $\rho$ replaced with $\rho_{\text{eq}}^{\text{flag}}$ from equation (2.51), in conjunction with the results from Section 2.3.

### 2.4.3 Steane Error Correction

This gadget, specifically designed for CSS codes, involves preparing logical $|0\rangle$ and $|+\rangle$ ancilla states, followed by the transversal application of CNOT gates to propagate $X$ and $Z$ errors from the data block to the ancilla [98]. The ancilla states are then destructively measured to extract the syndrome. Due to the transversality of the CNOT operations, no hook errors can occur, and the circuit does not require repeated measurements. Here, the ancilla error probability per qubit can be upper

bounded as

$$P_a \leqslant 1 - (1 - \rho_a)(1 - \rho_{1Q})(1 - \rho_{2Q})(1 - \rho_{\text{meas}}) \tag{2.55}$$

where $\rho_a$ is the depolarizing error probability on each ancilla qubit before the gadget. Note that, employing this error correction gadget, a single qubit error in the ancilla state can only produce a single qubit error in the encoded state. The depolarizing error probability on each qubit after the gadget is upper bounded as

$$\rho_{\text{eq}}^{\text{Ste}} \leqslant 1 - (1 - \rho)(1 - \rho_{2Q})^2(1 - P_a)^2 \tag{2.56}$$

where $\rho$ is the depolarizing error probability on each qubit before the gadget. Finally, an upper bound on the logical error rate after the gadget can be obtained by equation (2.11) with $\rho$ replaced with $\rho_{\text{eq}}^{\text{Ste}}$ from equation (2.56), in conjunction with the results from Section 2.3.

## 2.5 Numerical Results

In this section, we evaluate the performance of several codes in terms of the logical error rate as a function of the physical error rate, based on the analytical framework introduced earlier. The results are compared with numerical simulations to assess the accuracy and practical relevance of the analysis. For each simulation point in the numerical analysis, $100$ error realizations were tested to ensure sufficient statistical accuracy.

*Error patterns search via decoding:* In Tab. 2.3 we report the percentage of non-correctable errors $f_j(i, \ell)$ (i.e., composed by $j$ Pauli operator, $i$ of type $\boldsymbol{Z}$ and $\ell$ of type $\boldsymbol{X}$) for some surface and XZZX codes. These values have been evaluated by enumerating the error patterns of interest and running the MWPM decoder. To this aim, we employ the library for efficient modeling and optimization in networks (LEMON) C++ software [100], which provides an efficient implementation of graphs and networks algorithms. Once we have $f_j(i, \ell)$, we can compute the value of $1 - \beta_j$ for arbitrary $A$, according to (2.10). Hence, by using these tabular values, we can write analytical expressions for the code performance. This result

Table 2.3: Fraction of non-correctable error patterns $f_j(i,\ell)$ per error class of several topological planar codes decoded by minimum weight perfect matching.

| **Surface** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Code** | $XX$ | $XZ$ | $XY$ | $ZZ$ | $ZY$ | $YY$ | | | | |
| $[[13,1,3]]$ | 0.27 | 0 | 0.27 | 0.27 | 0.27 | 0.51 | | | | |
| $[[9,1,3]]$ | 0.5 | 0 | 0.5 | 0.5 | 0.5 | 1 | | | | |
| $[[23,1,3/5]]$ | 0.174 | 0 | 0.174 | 0 | 0 | 0.174 | | | | |
| $[[15,1,3/5]]$ | 0.324 | 0 | 0.324 | 0 | 0 | 0.324 | | | | |
| **Code** | $XXX$ | $XXZ$ | $XXY$ | $XZZ$ | $XZY$ | $XYY$ | $ZZZ$ | $ZZY$ | $ZYY$ | $YYY$ |
| $[[23,1,3/5]]$ | 0.434 | 0.174 | 0.434 | 0 | 0.174 | 0.434 | 0.066 | 0.066 | 0.235 | 0.487 |
| $[[15,1,3/5]]$ | 0.635 | 0.324 | 0.635 | 0 | 0.324 | 0.635 | 0.279 | 0.279 | 0.603 | 0.868 |
| $[[41,1,5]]$ | 0.0212 | 0 | 0.0212 | 0 | 0 | 0.0212 | 0.0212 | 0.0212 | 0.0212 | 0.042 |
| $[[25,1,5]]$ | 0.127 | 0 | 0.127 | 0 | 0 | 0.127 | 0.127 | 0.127 | 0.127 | 0.254 |
| **XZZX** | | | | | | | | | | |
| **Code** | $XX$ | $XZ$ | $XY$ | $ZZ$ | $ZY$ | $YY$ | | | | |
| $[[13,1,3]]$ | 0.218 | 0.051 | 0.27 | 0.218 | 0.27 | 0.513 | | | | |
| $[[9,1,3]]$ | 0.222 | 0.278 | 0.5 | 0.222 | 0.5 | 1 | | | | |
| $[[23,1,3/5]]$ | 0.123 | 0.016 | 0.138 | 0 | 0.016 | 0.154 | | | | |
| $[[15,1,3]]$ | 0.086 | 0.086 | 0.171 | 0.067 | 0.152 | 0.324 | | | | |
| **Code** | $XXX$ | $XXZ$ | $XXY$ | $XZZ$ | $XZY$ | $XYY$ | $ZZZ$ | $ZZY$ | $ZYY$ | $YYY$ |
| $[[23,1,3/5]]$ | 0.281 | 0.153 | 0.311 | 0.043 | 0.18 | 0.353 | 0.043 | 0.086 | 0.248 | 0.443 |
| $[[15,1,3]]$ | 0.244 | 0.241 | 0.399 | 0.236 | 0.391 | 0.619 | 0.211 | 0.380 | 0.391 | 0.868 |
| $[[41,1,5]]$ | 0.014 | 0.002 | 0.016 | 0.002 | 0.007 | 0.021 | 0.013 | 0.016 | 0.021 | 0.042 |
| $[[25,1,5]]$ | 0.027 | 0.034 | 0.06 | 0.034 | 0.067 | 0.127 | 0.027 | 0.06 | 0.127 | 0.254 |

can be used to analyze and design complex systems without implementing the decoder. For instance, the logical error rate of a rotated XZZX $[[9,1,3]]$ code tends for small $p$ to

$$p_{\mathrm{L}} \approx \frac{0.222A^2 + 1.556A + 2.222}{(A+2)^2} \binom{9}{2} p^2 \,. \tag{2.57}$$

Similarly, for the original $[[13,1,3]]$ surface code we have

$$p_{\mathrm{L}} \approx \frac{0.27A^2 + 0.54A + 1.32}{(A+2)^2} \binom{13}{2} p^2 \,. \tag{2.58}$$

As an example, for $A = 10$ we obtain $p_{\mathrm{L}} \to 10p^2$ for the rotated $[[9,1,3]]$ XZZX code and $p_{\mathrm{L}} \to 18.3p^2$ for the $[[13,1,3]]$ surface code.

By examining Tab. 2.3, it becomes apparent that the rotated XZZX code on a

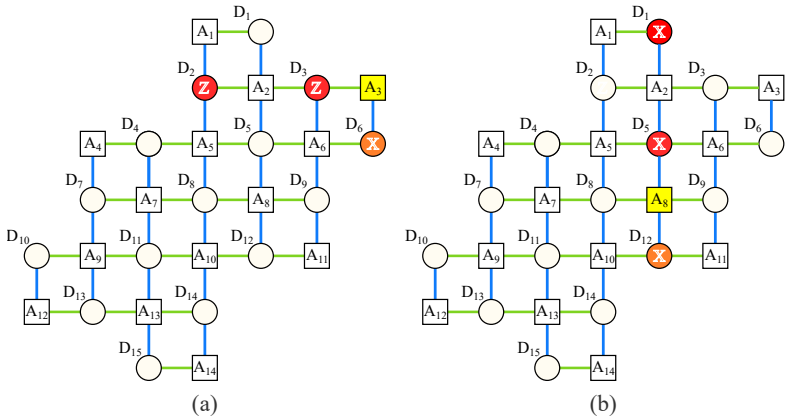


Figure 2.3: Examples of undetected error patterns on the rotated XZZX $[[15, 1, 3]]$. In red are represented errors introduced by the channel, in yellow the ancilla anticommuting with the error, and in orange the error correction applied by the MWPM decoder when the current highlighted ancilla is given as input. (a) A particular $\boldsymbol{ZZ}$ error producing a logical error when decoded. (b) A particular $\boldsymbol{XX}$ error producing a logical error when decoded.

rectangular lattice exhibits a degradation in code distance. Specifically, without the application of the XZZX variant, the code is characterized as $[[15, 1, 3/5]]$, but upon applying the XZZX variant, it becomes a $[[15, 1, 3]]$ code. Notably, in this specific configuration, there is no enhancement of asymmetric error correction capability through the XZZX structure. This is due to some error patterns which reduce the error correction capability of the starting code. We report in Fig. 2.3a an example where, starting from the $[[15, 1, 3/5]]$ rotated code, we apply the XZZX variant. Here we show a decoding error caused by an $\boldsymbol{ZZ}$ error pattern. Since a weight two pattern is not corrected, the code distance is not $d_{\mathrm{Z}} = 5$ for $\boldsymbol{Z}$ errors. In Fig. 2.3b we also show a failure caused by an $\boldsymbol{XX}$ error, which confirms that this asymmetric lattice with the XZZX variant has a symmetric distance $d = 3$.

*Comparison between the $\hat{\beta}_{t+1}$ from the WE and the exact $\beta_{t+1}$:* In Tab. 2.4 we compare the estimated $\hat{\beta}_{t+1}$ from Section 2.3.1, and the exact $\beta_{t+1}$ from Sections 2.3.3 and Tab. 2.3 over the depolarizing channel. We remark that the estimates $\hat{\beta}_{t+1}$ are derived solely by employing the logical weight enumerator of the particular quantum code (i.e., from the MacWilliams identities, or (2.34) for surface codes), whereas the exact values are obtained through a counting argument applied to the logical operators. Specifically, both the $[[5, 1, 3]]$ perfect code and

Table 2.4: Comparison between the bounds from Section 2.3.1, and the exact values from Sections 2.3.3 and Tab. 2.3, depolarizing channel.

| | $1-\hat{\beta}_{t+1}$ | $1-\beta_{t+1}$ |
|---|---|---|
| $[[5,1,3]]$ | 1 | 1 |
| $[[7,1,3]]$ | 0.88 | 0.88 |
| $[[9,1,3]]$ | 1 | 0.44 |
| $[[13,1,3]]$ | 0.51 | 0.24 |
| $[[23,1,3/5]]$ $e_{\mathrm{g}}=1$ | 0.13 | 0.07 |
| $[[23,1,3/5]]$ $e_{\mathrm{g}}+e_{\mathrm{z}}=2$ | 0.37 | 0.20 |
| $[[41,1,5]]$ | $2.5\cdot10^{-2}$ | $1.4\cdot10^{-2}$ |
| $[[85,1,7]]$ | $6.00\cdot10^{-4}$ | - |
| $[[145,1,9]]$ | $1.02\cdot10^{-5}$ | - |
| $[[181,1,10]]$ | $4.33\cdot10^{-7}$ | - |

the $[[7,1,3]]$ Steane code exhibit asymptotic non-degeneracy, leading to a congruence between the estimated values and their exact counterparts. Instead, the $[[9,1,3]]$ Shor code displays strong degeneracy, so that the estimate is useless ($\hat{\beta}_2 = 0$). In this case, we can resort to the logical operator analysis detailed in Section 2.3.3 which gives the exact $\beta_2$. Regarding surface codes, we observe that the estimated $\hat{\beta}_{t+1}$ closely approximate the exact values, with the disparity diminishing as the code's distance is increased.

*Comparison between analysis and simulation for the* $[[9,1,3]]$ *Shor code:* To verify the correctness of the proposed analytical approach we analyze the results for the $[[9,1,3]]$ Shor code. As observed, the estimation (2.26) is not useful, as it gives $\hat{\beta}_2 = 0$, which coincides with the BD decoding due to the strong degeneracy of the Shor code. Therefore, we use the logical operator analysis of Section 2.3.3. Fig. 2.4 shows, for the [[9,1,3]] code, a comparison between the upper bound (2.11), the asymptotic approximation (2.12), and the logical error rate obtained via Monte Carlo simulations, adopting a MW decoder. It can be seen that the results are in perfect agreement for $p < 0.1$, while there is a small gap for larger $p$. This gap arises because (2.11) implicitly assumes $\beta_j = 0$ for $j \geqslant 3$, while the Shor code is able to correct also a little percentage of errors of weight $j \geqslant 3$, as for instance $\boldsymbol{X}$ errors in three different qubit triplets ($\boldsymbol{X}_1\boldsymbol{X}_4\boldsymbol{X}_7$). Moreover, in the

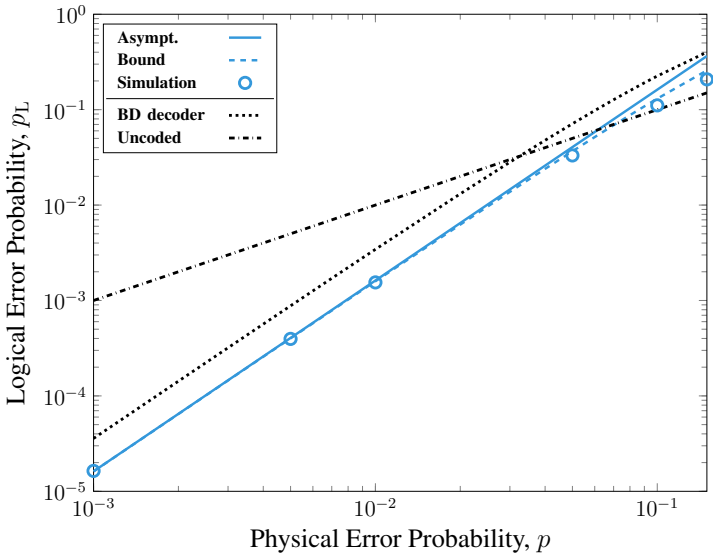


Figure 2.4: Logical error probability vs. physical error probability. Comparison between theoretical analysis (curves) and simulation (symbols) for the $[[9, 1, 3]]$ Shor code over a depolarizing channel. The curves refer to: the BD decoding performance (2.1); the MW decoding upper bound (2.11) and its asymptotic approximation (2.12) with the exact $\beta_2$ from Tab. 2.4.

plot, we report the error probability with the BD decoder, computed using (2.1), which has the same trend as the MW decoder. The gap between the two curves is due to the fraction of weight two errors which are corrected by the MW decoder.

*Asymptotic approximation and bounded distance decoder:* In Fig. 2.5 we report the asymptotic approximations computed using (2.57) for the $[[9, 1, 3]]$ rotated XZZX code, over the depolarizing and phase flip channels. We notice that the estimates provided in (2.12) closely align with the results obtained from simulations using MWPM decoding. This shows how the performance of an error-correcting quantum code with a complete decoder can be accurately described in the typical region of interest, i.e., $p < 0.1$, where the code is actually providing an improvement compared to the uncoded case. Moreover, we show in the same figure the bounded distance decoding performance, obtained by considering $\beta_j = 0$, for $j > t = \lfloor (d-1)/2 \rfloor$. The bounded distance decoding is unaffected by the asymmetry parameter. For this reason, it is not able to describe the advantage of XZZX codes over asymmetric channels. Specifically, the gap between bounded distance and MWPM decoding is due to the fact that topological codes are able to

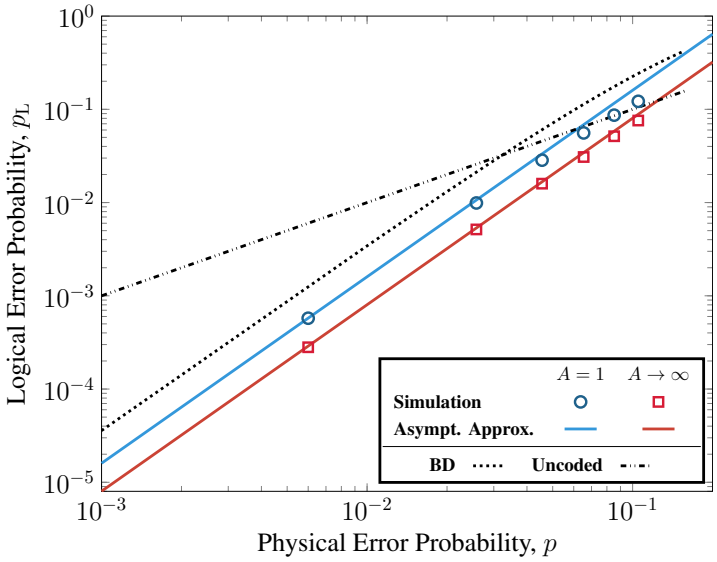


Figure 2.5: Logical error probability vs. physical error probability. Comparison between simulation (mark symbols), asymptotic approximations (2.12) (solid lines), and BD decoding performance (2.1) (dotted line) for the $[[9, 1, 3]]$ XZZX code over depolarizing and phase flip channels.

correct a large number of errors of weight $w \geqslant t + 1$. The $[[9, 1, 3]]$ XZZX code has the greatest error correction capability over the phase flip channel, where it can exploit its intrinsic symmetries which make one kind of Pauli error to align always in the same direction.

*Comparison of topological planar codes:* In Fig. 2.6 we depict some XZZX and surface codes over a channel with asymmetry $A = 10$. Among the codes with distance $d = 3$, the rotated $[[9, 1, 3]]$ XZZX code is to be preferred due to its good performance by using the lowest number of qubits. This shows that the rotation technique is able to obtain an increase of the coding rate without deteriorating the performance. As expected, the combination of a rectangular lattice, rotation, and XZZX deteriorates the performance, and here it is shown by the fact that the $[[15, 1, 3]]$ rotated XZZX code achieves the same performance of the shortest planar code (i.e., the $[[9, 1, 3]]$). Moving to lower coding rates, well-design rectangular lattices show a higher $\boldsymbol{Z}$ error correction capability. For this reason, they represent a good compromise between performance and codeword length. On the other hand, codes with large distance show the best performance.

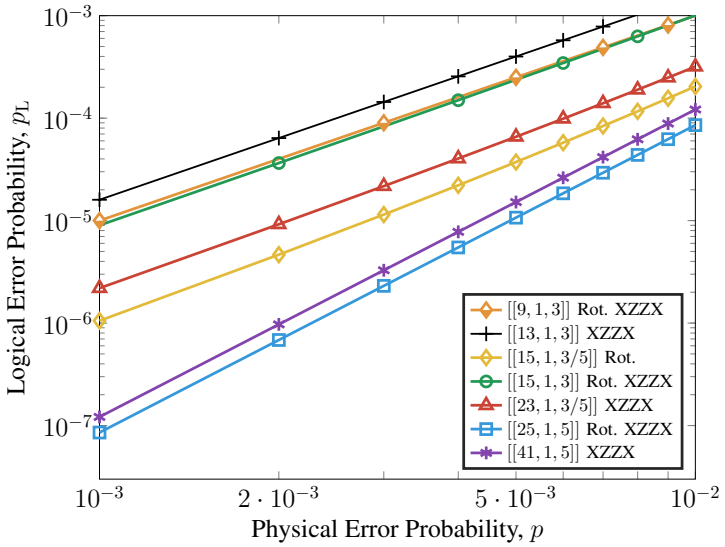


Figure 2.6: Logical error probability vs. physical error probability with channel asymmetry $A = 10$. The curves refer to the asymptotic approximations (2.12) for several topological planar codes of possible interest.

Finally, it is visible in the plot how the code distance influences the slope of the performance curve. For asymmetric lattices in the asymptotic regime $p \ll 1$, those with finite $A$ show the same slope as $d = 3$ codes, while those with $A = \infty$ match the slope of $d = 5$ codes.

*Effect of channel asymmetry on topological planar codes*: In Fig. 2.7, Fig. 2.8, and Fig. 2.9 we show how the logical error rate is affected by the channel asymmetry $A$, for all considered surface codes. To this aim, we fix $p = 5 \cdot 10^{-3}$ to be in the $p \ll 1$ regime. In Fig. 2.7 we report the logical error rate for topological planar codes with $d = 3$. Since they have the same code distance the relation between them is independent by the chosen $p$. As expected, we have that XZZX technique does not provide any advantage when $A = 1$, while it shows a clear performance improvement in the presence of channel asymmetries. The rotated codes for $d = 3$ are able to achieve advantages both in terms of codeword length and performance compared to the non-rotated counterparts. We can conclude that the rotated $[[9, 1, 3]]$ XZZX code is the best choice among the surface codes with distance $d = 3$. On the contrary, for $d = 5$ the performance of the rotated codes slightly degrades on the depolarizing channel as shown in Fig. 2.8, due to the

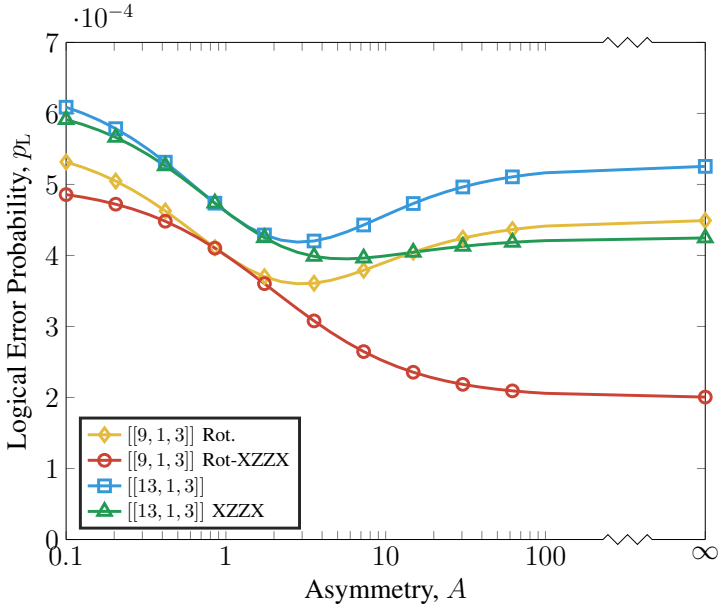


Figure 2.7: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.005$. Surface codes with $d = 3$. The curves refer to the asymptotic approximations (2.12).

larger number of error patterns beyond the minimum distance that the decoder can correct. However, given the reduced number of qubits, the small loss for the depolarizing channel, and the performance improvement for asymmetric channels, the rotated $[[25, 1, 5]]$ XZZX code can be considered the best choice among those with $d = 5$. Finally, in Fig. 2.9 we report codes constructed on rectangular lattices with dimension $3 \times 5$. The surface codes are the $[[23, 1, 3/5]]$, with and w/o XZZX, and its rotated versions $[[15, 1, 3/5]]$ and $[[15, 1, 3]]$ XZZX. As highlighted before, the latter is the only one not able to guarantee $d_{\mathrm{Z}} = 5$.

*6) Noisy syndrome extraction performance analysis.* In Fig. 2.10 and Fig. 2.11, for the $[[13, 1, 3]]$ surface code, we compare the simulated logical error rates under noisy syndrome extraction with the upper bounds described in Section 2.4. In Fig. 2.10 we set the error rates as $\rho_{2Q} = \rho_{1Q} = \rho_{\mathrm{init}} = \rho_{\mathrm{init}}^{\mathrm{cat}} = \rho_{\mathrm{meas}} = \rho/100$, while in Fig. 2.11 we set $\rho_{2Q} = \rho_{1Q} = \rho_{\mathrm{init}} = \rho_{\mathrm{init}}^{\mathrm{cat}} = \rho_{\mathrm{meas}} = \rho/10$. Furthermore, we set the Steane ancillary state preparation error $\rho_a = \rho/10$, or $\rho_a = \rho/2$, to ensure that the ancillary resource state employed for syndrome extraction is more reliable than the data state it is used to correct. To achieve FT syndrome extrac-

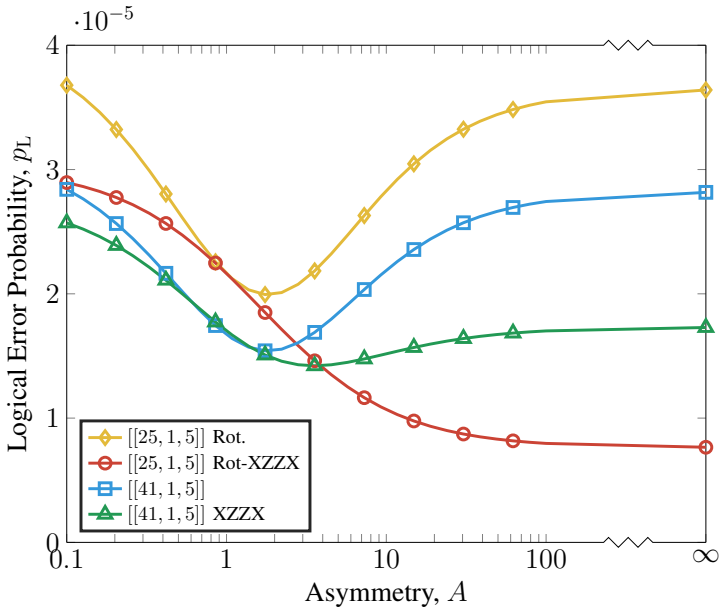


Figure 2.8: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.005$. Surface codes with $d = 5$. The curves refer to the asymptotic approximations (2.12).

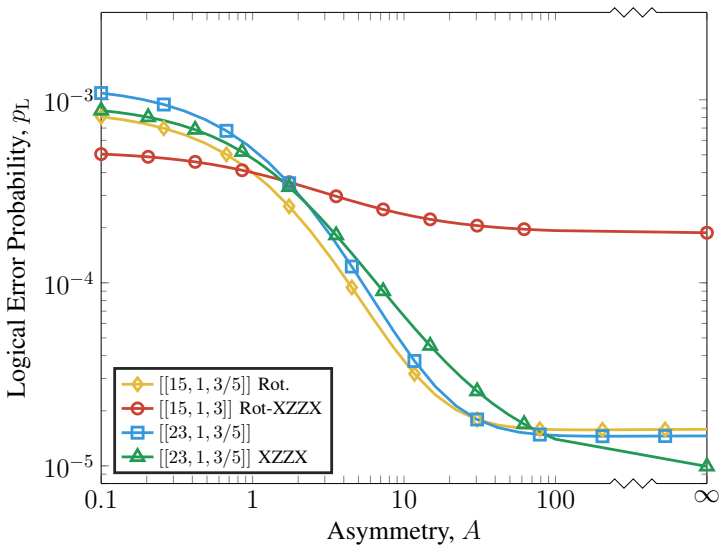


Figure 2.9: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.005$. Surface codes with $d_X = 3/d_Z = 5$. The curves refer to the asymptotic approximations (2.12).

tion, one strategy is to repeatedly measure all stabilizer generators until the same syndrome is obtained $t+1$ times consecutively. Considering that up to $t$ errors may occur during this process, in the worst-case scenario, $(t+1)^2$ measurement rounds are required [19,21]. Hence, we choose $r = (t+1)^2 = 4$, while $\delta_{\max} = 4$, since some qubits participate in two $\boldsymbol{X}$ and two $\boldsymbol{Z}$ generators. For weight-four generators, two flags ancillas are needed for fault detection. Setting $\beta_2 = 0.76$, we compute upper bounds on the logical error rate by applying equations (2.49), (2.51), and (2.56) in (2.11), for the various gadgets discussed in Section 2.4. From our analysis, we observe that the proposed upper bounds are tight with respect to the numerical simulations. For circuit error rates equal to $\rho/100$ and $\rho_a = \rho/10$, the three techniques exhibit comparable performance. In particular, the effectiveness of the Steane syndrome extraction gadget is highly dependent on the fidelity of the resource state used in the procedure. Specifically, when $\rho_a = \rho/10$, it achieves the lowest logical error rates among the considered gadgets; however, for $\rho_a = \rho/2$, its performance deteriorates significantly. In contrast, the performance of the cat and flag syndrome extraction gadgets depends strongly on the circuit error rates: when these are set to $\rho/10$, the resulting logical error rates are substantially higher.

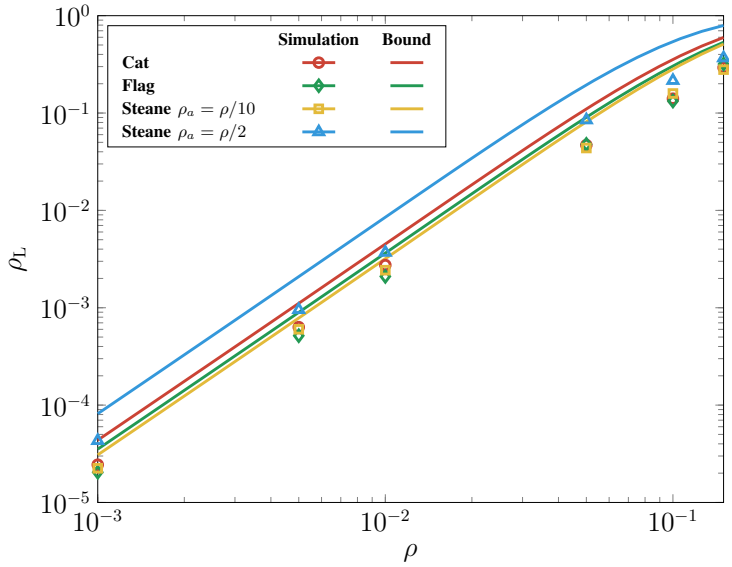


Figure 2.10: Logical error rates considering noisy syndrome extraction: comparison between theoretical analysis and simulation using the MWPM decoder for the $[[13, 1, 3]]$ surface code over a depolarizing channel, assuming uniform noise parameters $\rho_{2Q} = \rho_{1Q} = \rho_{\text{init}} = \rho_{\text{init}}^{\text{cat}} = \rho_{\text{meas}} = \rho/100$.

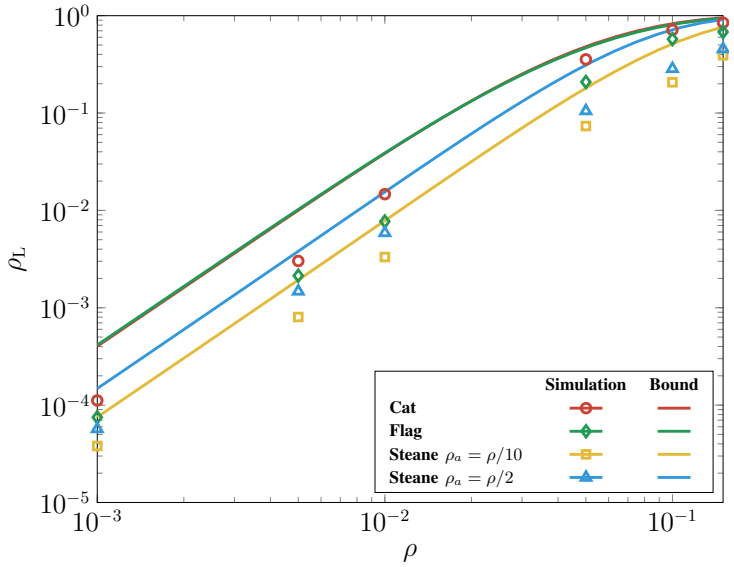


Figure 2.11: Logical error rates considering noisy syndrome extraction: comparison between theoretical analysis and simulation using the MWPM decoder for the $[[13, 1, 3]]$ surface code over a depolarizing channel, assuming uniform noise parameters $\rho_{2Q} = \rho_{1Q} = \rho_{\text{init}} = \rho_{\text{init}}^{\text{cat}} = \rho_{\text{meas}} = \rho/10$.

# Chapter 3

# Design of Quantum Codes Tailored for Asymmetric Channels

In practical scenarios, quantum technologies are often more prone to dephasing noise, therefore suffering more phase-flip errors compared to bit-flip ones [61, 62, 101, 102]. Due to this, asymmetric quantum codes have been proposed as a solution for scenarios where strong asymmetries in quantum channel errors arise [30, 103]. In [103], codes are constructed starting from classical codes, using the CSS construction. In [73, 104], non-CSS asymmetric codes are constructed using the quantum Hamming bound, by syndrome assignment in order to find the shortest possible codes able to guarantee a certain asymmetric error correction capability. In [30], the XZZX variant of surface codes has been proposed. As shown in Section 2.5, XZZX codes exhibit a performance boost in presence of quantum channel asymmetries when compared to the conventional surface code.

In this chapter, we propose a new class of quantum codes, named ZZZY surface codes, in which a few Pauli $\boldsymbol{Y}$ measurements are incorporated at carefully selected locations within the lattice. This approach aims to improve code performance over asymmetric channels while considering decoder complexity. To preserve the use of the MWPM decoder, while admitting an additional lowcomplexity pre-processing phase, we insert at most one $\boldsymbol{Y}$ measurement per plaquette. The resulting ZZZY surface code shows a marked improvement in the correction of error patterns consisting of $\boldsymbol{Z}$ operators.

Moreover, we introduce two classes of quantum topological codes referred to as cylindrical and Möbius codes, particular cases of the fiber bundle codes [43]. Cylindrical codes exhibit a two-dimensional topological structure that can be configured over planar lattices, and therefore necessitating only local qubit interactions in two dimensions. On the other hand, despite the topological structure of Möbius codes, in two dimension they are quasi-planar, in the sense that few non-local qubit interactions are required. In this scenario, a natural implementation choice could be to leverage the inherent mobility offered by reconfigurable atom arrays [105]. These architectures allow for processor connectivity to be reconfigured during quantum evolution by shuttling atoms around in optical tweezers, with minimal decoherence. As a result, they are particularly well-suited for realizing a limited number of remote connections.

In the following, we focus on the design and performance assessment of the proposed codes under asymmetric noise, highlighting their advantages for quantum memory applications. When extending their use to quantum computation, standard lattice surgery techniques [95, 106, 107] developed for surface codes should be suitably adapted to support FT logical operations within these architectures.

## 3.1 Quantum ZZZY Codes

The ZZZY codes are obtained starting from the lattice of a non rotated surface code, by modifying some of the measurements of the generators, as shown in Fig. 3.1. We emphasize that these codes are still planar and they require only local connectivity between qubits. Moreover, for the decoding it is possible to employ the MWPM algorithm, with the addition of some conditional statements. In the case of standard squared surface codes, each generator is responsible for only one kind of Pauli error (e.g., $\boldsymbol{X}$ or $\boldsymbol{Z}$), since they are composed by either all $\boldsymbol{X}$ or all $\boldsymbol{Z}$ operators. As a result, such codes have a balanced error correction capability and perform best over symmetric channels. The basic idea behind ZZZY codes is to sacrifice some $\boldsymbol{X}$ error correction capability to enhance the performance of the code over channels where phase flip errors are the most probable. Hence, we substitute a $\boldsymbol{Z}$ with a $\boldsymbol{Y}$ measurement for a subset of generators. For instance, we

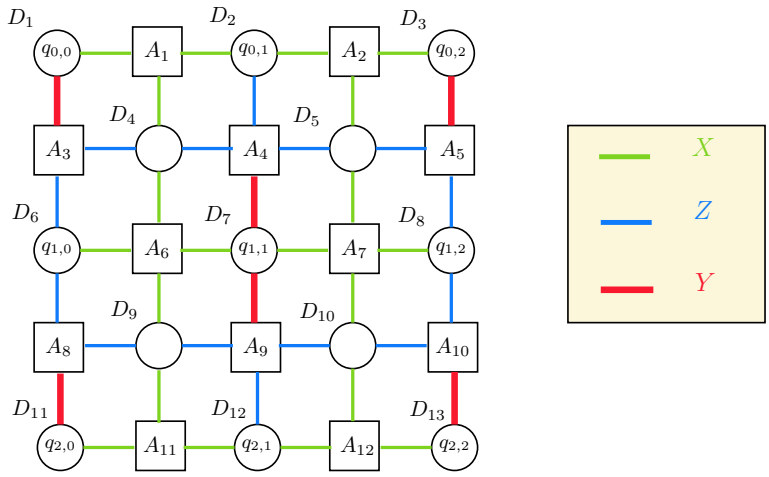


Figure 3.1: $[[13, 1, 3]]$ ZZZY code. Circles stand for data qubits $D$, and squares for ancillae $A$. The six edges depicted in red denote a modified $\boldsymbol{Y}$ measurement with respect the standard surface code. $\boldsymbol{X}$, $\boldsymbol{Z}$, and $\boldsymbol{Y}$ measurements are depicted in green, blue, red, respectively.

design the $[[13, 1, 3]]$ ZZZY code with the following generators

$$
\begin{aligned}
&\boldsymbol{G}_1 = \mathbf{X}_1\mathbf{X}_2\mathbf{X}_4 \quad \boldsymbol{G}_2 = \mathbf{X}_2\mathbf{X}_3\mathbf{X}_5 \\
&\boldsymbol{G}_3 = \mathbf{Y}_1\mathbf{Z}_4\mathbf{Z}_6 \quad \boldsymbol{G}_4 = \mathbf{Z}_2\mathbf{Z}_4\mathbf{Z}_5\mathbf{Y}_7 \quad \boldsymbol{G}_5 = \mathbf{Y}_3\mathbf{Z}_5\mathbf{Z}_8 \\
&\boldsymbol{G}_6 = \mathbf{X}_4\mathbf{X}_6\mathbf{X}_7\mathbf{X}_9 \quad \boldsymbol{G}_7 = \mathbf{X}_5\mathbf{X}_7\mathbf{X}_8\mathbf{X}_{10} \\
&\boldsymbol{G}_8 = \mathbf{Z}_6\mathbf{Z}_9\mathbf{Y}_{11} \quad \boldsymbol{G}_9 = \mathbf{Y}_7\mathbf{Z}_9\mathbf{Z}_{10}\mathbf{Z}_{12} \quad \boldsymbol{G}_{10} = \mathbf{Z}_8\mathbf{Z}_{10}\mathbf{Y}_{13} \\
&\boldsymbol{G}_{11} = \mathbf{X}_9\mathbf{X}_{11}\mathbf{X}_{12} \quad \boldsymbol{G}_{12} = \mathbf{X}_{10}\mathbf{X}_{12}\mathbf{X}_{13}
\end{aligned}
$$

which are shown in Fig 3.1. Hereafter, we will denote these modified generators as $\boldsymbol{ZY}$ generators. To build larger ZZZY codes, it is sufficient to start from the corresponding $[[n, k, d]]$ surface code, as follows. Considering data qubits only on odd rows, let assign two indices $i$ and $j$ to each data qubit in the lattice, where $i, j = 0, \ldots, d-1$, denoting the row and column of the respective qubit $q$. Some examples of these labels are depicted in Fig. 3.1. Next, transform the $\boldsymbol{Z}$ measurements on qubits $q_{2\ell,0}$ and $q_{2\ell,d-1}$, with $\ell = 0, \ldots, d-1$, into $\boldsymbol{Y}$ measurements. Finally, convert the $\boldsymbol{Z}$ measurements on qubits $q_{2\ell+1,1}$ and $q_{2\ell+1,d-2}$ to $\boldsymbol{Y}$ measurements. For the particular case $d = 3$, depicted in Fig. 3.1, $d-2 = 1$ and then $q_{2\ell+1,1}$ and $q_{2\ell+1,d-2}$ is the same qubit, for each $\ell$. This leads to $3(d-1) = 6$

modifications to $\boldsymbol{X}$ generators when $d = 3$. It is easy to show that, when $d > 3$, the procedure leads to $4(d-1)$ modifications of $\boldsymbol{X}$ generators. Note that the dual construction, where some $\boldsymbol{X}$ are replaced by $\boldsymbol{Y}$ to improve the error correction capability of bit flip errors, can be achieved in a similar manner.

To transform a surface code into its corresponding ZZZY code, it is sufficient to apply a specific sequence of gates to a subset of data qubits, thereby mapping the $\boldsymbol{Z}$-type operators in the stabilizer generators to $\boldsymbol{Y}$-type operators. This transformation is achieved by the composite operator $\boldsymbol{H}\boldsymbol{S}^{\dagger}\boldsymbol{H}$.

In the following, we will examine the logical operators of the $[[13, 1, 3]]$ ZZZY code to elucidate the advantage it attains in the presence of $\boldsymbol{Z}$ channel errors. The number of logical operators of each weight can be computed starting from MacWilliams identities as shown in Section 2.3. Specifically, for the $[[13, 1, 3]]$ code we find that the undetectable error WE polynomial is $L(z) = 6z^3 + 24z^4 + 75z^5 + 240z^6 + 648z^7 + 1440z^8 + 2538z^9 + 3216z^{10} + 2634z^{11} + 1224z^{12} + 243z^{13}$. Since this code has distance three, its asymptotic logical error rate depends on the fraction of errors of weight $j = 2$ that it is able to correct. In particular, errors of weight $j = 2$ can cause logical operators of weight $w = 3$ and $w = 4$. We observe that the $[[13, 1, 3]]$ ZZZY code, as for the surface code, has six logical operators with $w = 3$ and 24 logical operators with $w = 4$. Specifically, by comparing the logical operators with those of the standard surface code, as discussed in Section 2.3.3, we observe that the logical operators composed of $\boldsymbol{X}$ Pauli operators remain unchanged for the $[[13, 1, 3]]$ ZZZY code, whereas those composed of $\boldsymbol{Z}$ Pauli operators have some of their $\boldsymbol{Z}$ terms replaced by $\boldsymbol{Y}$.

Since this code is designed for channels in which phase-flip errors occur more frequently, we focus our discussion on $\boldsymbol{Z}$ Pauli errors. Referring to Fig 3.1, some examples of logical operators for the standard $[[13, 1, 3]]$ surface code with $w = 3$ and $w = 4$ are $\boldsymbol{Z}_1\boldsymbol{Z}_2\boldsymbol{Z}_3$ and $\boldsymbol{Z}_1\boldsymbol{Z}_2\boldsymbol{Z}_5\boldsymbol{Z}_8$, respectively. The first one can be caused by three error patterns: $\boldsymbol{Z}_1\boldsymbol{Z}_2$, $\boldsymbol{Z}_1\boldsymbol{Z}_3$, and $\boldsymbol{Z}_2\boldsymbol{Z}_3$. In surface codes, where these errors are detected exploiting only information coming from $\boldsymbol{X}$ generators, whenever one of these patterns occurs, the MWPM is not able to recover it. For instance, if the channel introduces a $\boldsymbol{Z}_1\boldsymbol{Z}_2$ error, the decoder will apply a $\boldsymbol{Z}_3$, realizing the correspondent logical operator. However, ZZZY codes have additional information coming from $\boldsymbol{ZY}$ generators. Indeed, in case of a $\boldsymbol{Z}_1\boldsymbol{Z}_2$ occurs, an-

cilla qubit $A_3$, which performs $\boldsymbol{Y}_1\boldsymbol{Z}_4\boldsymbol{Z}_6$ measurements, anticommutes with the error and it is switched on during the error correction. In particular, the corresponding logical operator in the ZZZY code is $\boldsymbol{Y}_1\boldsymbol{Z}_2\boldsymbol{Y}_3$; hence, it cannot result solely from the action of Pauli $\boldsymbol{Z}$ errors.

A similar reasoning can be done also for logical operators with $w = 4$. Let us consider the surface code logical operator $\boldsymbol{Z}_1\boldsymbol{Z}_2\boldsymbol{Z}_5\boldsymbol{Z}_8$. This operator is due to $\binom{4}{2} - 2 = 4$ pattern of errors of weight two: $\boldsymbol{Z}_1\boldsymbol{Z}_5$, $\boldsymbol{Z}_2\boldsymbol{Z}_8$, $\boldsymbol{Z}_1\boldsymbol{Z}_8$, and $\boldsymbol{Z}_2\boldsymbol{Z}_5$. This is because $\boldsymbol{Z}_1\boldsymbol{Z}_2$ causes a logical operator with $w = 3$, while $\boldsymbol{Z}_5\boldsymbol{Z}_8$ is always corrected. In particular, the decoding error is due to the fact that the MWPM is not able to distinguish between $\boldsymbol{Z}_1\boldsymbol{Z}_5$ and $\boldsymbol{Z}_2\boldsymbol{Z}_8$ ($\boldsymbol{Z}_1\boldsymbol{Z}_8$ and $\boldsymbol{Z}_2\boldsymbol{Z}_5$) since they give the same syndrome. However, in the $[[13, 1, 3]]$ ZZZY code, a $\boldsymbol{Z}_1\boldsymbol{Z}_5$, contrary to $\boldsymbol{Z}_2\boldsymbol{Z}_8$, would switch on $A_3$, which can be exploited to identify the correct channel error. Indeed, the corresponding logical operator for the $[[13, 1, 3]]$ ZZZY code is $\boldsymbol{Y}_1\boldsymbol{Z}_2\boldsymbol{Z}_5\boldsymbol{Z}_8$.

Hence, in the $[[13, 1, 3]]$ ZZZY code, all $\boldsymbol{Z}$ error patterns of weight $t + 1$ are corrected, except for one: $\boldsymbol{Z}_6\boldsymbol{Z}_8$, resulting in $\beta_2 = 0.987$. This cannot be corrected as it results in the same syndrome as the error $\boldsymbol{Y}_7$.

## 3.2 ZZZY Minimum Weight Perfect Matching

In decoding ZZZY codes, we must adapt the standard MWPM algorithm to leverage the insights gained from $\boldsymbol{Y}$ measurements. Notably, as surface codes fall under the category of CSS codes, the decoding process for $\boldsymbol{Z}$ generators operates independently from that for $\boldsymbol{X}$ generators [93]. Consequently, the MWPM can be divided into two phases: $\text{MWPM}_{\mathbf{X}}$, focusing solely on $\boldsymbol{X}$ generators, followed by $\text{MWPM}_{\mathbf{Z}}$ for the $\boldsymbol{Z}$ stabilizers. As detailed in Section 3.1, $\boldsymbol{ZY}$ generators offer insights into certain $\boldsymbol{Z}$ errors. However, without careful handling, they can erroneously trigger $\boldsymbol{X}$ error detections. Take, for example, Fig 3.1, where a $\boldsymbol{Z}_1$ error activates ancillas $A_1$ and $A_3$. Neglecting to deactivate ancilla $A_3$ before $\text{MWPM}_{\mathbf{Z}}$ would falsely attribute an additional $\boldsymbol{X}_1$ error. To address this, we introduce a preprocessing step to both $\text{MWPM}_{\mathbf{X}}$ and $\text{MWPM}_{\mathbf{Z}}$. The algorithm's complete description utilizes binary representation for the generators (i.e., for the parity check matrix $\boldsymbol{H}$) and the estimated channel error vector $\hat{\boldsymbol{e}}$. For instance, in a code

**Algorithm 1:** `ZZZY_Decoder`

**input :** $\boldsymbol{s}$, syndrome
$\boldsymbol{H}$, matrix of the generators
$n_{\mathbf{ZY}}$, $n_{\mathbf{X}}$, number of $\boldsymbol{ZY}$ and $\boldsymbol{X}$ generators
**output:** $\hat{\boldsymbol{e}}$, vector of the estimated channel errors

```
init q to all ones, vector of the weights associated to each data qubit of the lattice
q ← update_weights(s, q, H, n_ZY, n_X)
D ← compute_distance(s), matrix of the distances between switched on ancilla
ê ← MWPM_X(D)
forall i ∈ {1, ..., n_ZY} do
    forall j ∈ {1, ..., n} do
        if ê(j) = 1
            if H(i, j) = 1 and H(i, j + n) = 1
                s(j) ← 1 − s(j)
ê ← MWPM_Z(D)
```

with $n$ qubits, the matrix $\boldsymbol{H}$ comprises $2n$ columns, with each row representing a generator. The first $n$ columns contain a 1 where the corresponding generator features a $\boldsymbol{Z}$ or $\boldsymbol{Y}$ Pauli measurement, while the second $n$ columns contain a 1 if the generators measure $\boldsymbol{X}$ or $\boldsymbol{Y}$ [72]. We also use the first $n_{\mathbf{ZY}}$ rows to describe the $\boldsymbol{ZY}$ generators. The decoder for ZZZY codes is presented as Algorithm 1 above. Excluding the function `update_weights` (to be introduced later), the algorithm ensures the minimum distance for ZZZY surface codes. After evaluating the syndrome, the function `compute_distance` utilizes Dijkstra's algorithm to find the shortest paths on a graph, where vertices correspond to switched on ancillas and edges' weights are the sums of the underlying qubit weights. Subsequently, via $\text{MWPM}_{\mathbf{X}}$, pairs of $\boldsymbol{X}$ ancillas are connected, producing the estimated $\boldsymbol{Z}$ channel errors. Next, the parity of all ancillas measuring $\boldsymbol{Y}$ operators on qubits involved in $\boldsymbol{Z}$ errors is inverted. Finally, $\text{MWPM}_{\mathbf{Z}}$ also allows for finding the $\boldsymbol{X}$ channel errors. The algorithm corrects, therefore, all patterns of weight up to $t$. Further, we would like to correct as much as possible $\boldsymbol{Z}$ errors of weight $t+1$. To this aim we can exploit the information coming from $\boldsymbol{ZY}$ generators. Specifically, after evaluating the syndrome, if some of the $\boldsymbol{ZY}$ generators are activated, we modify the weights of the edges of the MWPM graph using the function `update_weights`. Since we are considering minimum weight decoders, error patterns of weight $t+1$ could trigger only logical operators of weight $d$ and $d+1$ (if, as assumed, $d$ is odd).

**Algorithm 2:** `update_weights`

**input :** $\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{H}, n_{\mathbf{ZY}}, n_{\mathbf{X}}$
**output:** $\boldsymbol{q}$

**forall** $i \in \{1, \ldots, n_{\mathbf{ZY}}\}$ **do**
  **forall** $j \in \{1, \ldots, n\}$ **do**
    **if** $\boldsymbol{s}(i) = 1$
      **if** $\boldsymbol{H}(i,j) = 1$ *and* $\boldsymbol{H}(i, j+n) = 1$
        $\boldsymbol{q}(j) \leftarrow 0.9$
    **if** $\boldsymbol{s}(i) = 0$
      **if** $\boldsymbol{H}(i,j) = 1$ *and* $\boldsymbol{H}(i, j+n) = 1$
        $\boldsymbol{q}(j) \leftarrow 1.1$

init $\mathcal{A}$ to the empty set
**forall** $i \in \{1, \ldots, n_{\mathbf{ZY}}\}$ **do**
  **if** $\boldsymbol{s}(i) = 1$
    $\mathcal{A} \leftarrow \mathcal{A} \cup n_i$
    **forall** $j \in g(h(n_i))$ **do**
      **if** $\boldsymbol{s(j)} = 1$
        $\mathcal{A} \leftarrow \mathcal{A} \setminus n_i$

**forall** $i \in \mathcal{A}$ **do**
  **forall** $j \in \{1, \ldots, n\}$ **do**
    **if** $\boldsymbol{H}(i,j) = 1$ *and* $\boldsymbol{H}(i, j+n) = 1$
      $\boldsymbol{q}(j) \leftarrow -0.1$

Let us start improving the correction in case of possible logical operators of weight $d+1$. We can achieve this if we apply the following procedure: if one of the generators performing a $\boldsymbol{Y}$ measurement on the $i$-th qubit is activated, the weight $\boldsymbol{q}(i)$ of the corresponding edge is modified to a number slightly smaller than one, e.g., $\boldsymbol{q}(i) = 0.9$. Moreover, if a generator performing a $\boldsymbol{Y}$ measurement on the $i$-th qubit is switched off, the weight $\boldsymbol{q}(i)$ is set to a number slightly larger than one, e.g., $\boldsymbol{q}(i) = 1.1$. In this way, during the MWPM$_{\mathbf{X}}$, the decoder is pushed to choose paths where the $\boldsymbol{ZY}$ generators are switched on. If the $i$-th qubit is actually affected by a $\boldsymbol{Z}$ Pauli error, this strategy allows the decoder to choose correctly between different paths composed by the same number of edges. We elucidate this with an example reported in Fig 3.2a. In particular, if $\boldsymbol{Z}$ errors occur on data qubits $D_6$ and $D_3$, $\boldsymbol{ZY}$ ancilla $A_5$ is switched on. If we directly apply MWPM$_{\mathbf{X}}$, the decoder has to choose between three error patterns of the same weight: $\boldsymbol{Z}_3\boldsymbol{Z}_6$, $\boldsymbol{Z}_2\boldsymbol{Z}_4$, and $\boldsymbol{Z}_5\boldsymbol{Z}_7$. This ambiguity could lead to an error

with high probability. However, with our modification, the weight of qubit $D_3$ is set to $0.9$, guiding MWPM$_{\mathbf{X}}$ to select it for correction.

Let us now focus on logical operators of weight $d$. In this case, if an error pattern with $t+1$ Pauli $\boldsymbol{Z}$ operators occurs activating a $\boldsymbol{ZY}$ generator, we would like the decoder to select a path composed of a higher number of qubits if certain conditions are met. In doing so, we need to be sure that we are dealing with a potential logical operator of weight $d$. For this reason, if a $\boldsymbol{ZY}$ generator measuring a $\boldsymbol{Y}$ operator on qubit $i$-th is activated, and there are no $\boldsymbol{X}$ generators activated in the rows of the lattice adjacent to the one of qubit $i$-th, $\boldsymbol{q}(i)$ is set to a small negative number, e.g., $-0.1$, to force its selection. To formalize the algorithm, let us define $h(\cdot)$ as a function that takes as input the index of a $\boldsymbol{ZY}$ generator and returns the index $\ell$ of the qubit under $\boldsymbol{Y}$ measurement. Additionally, we define the function $g(\cdot)$, which takes as input a qubit index and returns a list of $\boldsymbol{X}$ generator indexes located in the row above and in the row below the input qubit. This function can be implemented efficiently using modulo operations. An example is depicted in Fig 3.2 b. Specifically, $\boldsymbol{Z}$ errors have occurred on qubits $D_2$ and $D_3$. Applying the function $h(\cdot)$ to $A_5$, we obtain $h(5) = 3$, representing $D_3$. Consequently, $g(3)$ returns $\{6, 7\}$.The list has only two elements due to the fact that $D_3$ is on a boundary. Since ancillas $A_6$ and $A_7$ are both deactivated, the weight of qubit data $D_3$, measured by $A_5$, is set to $-0.1$, ensuring the correction of the error. On the other hand, without our `Update_weights`, the MWPM$_{\mathbf{X}}$ decoder would apply a $\boldsymbol{Z}_1$ correction, leading to the logical operator $\boldsymbol{Z}_1\boldsymbol{Z}_2\boldsymbol{Z}_3$. In the worst case scenario, for each of the $4(d-1)$ $\boldsymbol{ZY}$ generators we could perform an assignment based on two conditional statements. In practice, this can be easily implemented in hardware by means of simple logic gates, resulting in a pre-processing complexity of $O(1)$.

**Lemma 1.** *Given an $[[n, k, d]]$ ZZZY code, for $d > 3$, the fraction of $\boldsymbol{Z}$ errors of weight $t + 1$ that cannot be corrected by the ZZZY decoder over a phase flip channel is $d\binom{d-2}{t+1}/\binom{n}{t+1}$*

*Proof.* Over a phase-flip channel, all errors of weight $t + 1$ that can cause logical operators of weight $2t + 2$ are corrected. Indeed, the decoder has to choose between two solutions composed of the same number of qubits. Hence, by modifying the weight of the paths using the function `update_weights`, the actual error

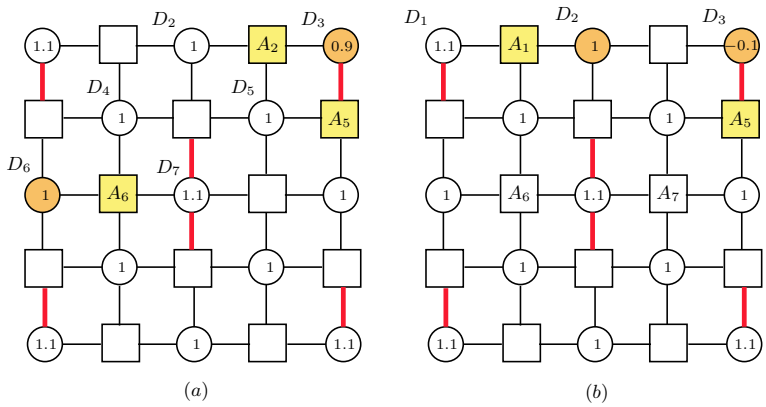


Figure 3.2: Decoding of the $[[13, 1, 3]]$ ZZZY code. Qubits affected by $\boldsymbol{Z}$ errors are highlighted in orange. Switched on ancillas are depicted in yellow. Each qubit $i$ is associated with the corresponding weight $\boldsymbol{q}(i)$ resulting from the function `Update_weights`.

pattern is always identified. In case the $t + 1$ errors occur on the same row of the lattice, they can cause a logical operator of weight $2t + 1$. To correct these errors, it is necessary that at least one of them occurs on a qubit measured by one of the two $\boldsymbol{ZY}$ generators, since the ZZZY decoder has to set the weight of the corresponding qubit to $-0.1$. Hence, the uncorrected error patterns for each of the $d$ rows are $\binom{d-2}{t+1}$. Finally, the total number of $\boldsymbol{Z}$ error patterns of weight $t + 1$ is $\binom{n}{t+1}$. □

Note that, as the code distance increases, the fraction of errors of weight $t + 1$ that cannot be corrected becomes smaller.

## 3.3 Cylindrical and Möbius Quantum Codes

In what follows, we firstly introduce several concepts from topology, which we later use to construct quantum cylindrical codes and determine their parameters.

### 3.3.1 Topological Interpretation of Linear Codes

A *chain complex* $C$ is a collection of vector spaces $\{C_i\}$ for $i = 0, 1, ..., m$, and linear maps [108, 109]

$$\partial_i : C_i \to C_{i-1} \tag{3.1}$$

with the requirement that $\partial_i \partial_{i+1} = 0$, or, equivalently, that $\mathrm{im}\ \partial_{i+1} \subseteq \ker \partial_i$. Note that the trivial operators $\partial_0 : C_0 \to \{0\}$ and $\partial_{m+1} : \{0\} \to C_m$ are implicit. In particular, each vector space has a canonical basis, which elements are called *i-cells*, while vectors of $C_i$ are named *i-chains*. It is possible to express the *i-th homology* as the quotient space

$$H_i(C) = \frac{\ker \partial_i}{\mathrm{im}\ \partial_{i+1}} \tag{3.2}$$

where elements of $\ker \partial_i$ and $\mathrm{im}\ \partial_{i+1}$ are called *i-cycles* and *i-boundaries*, respectively. Each chain complex has a dual chain, or *cochain*. This is defined as the sequence of dual vector spaces $C_i^*$ (the space of linear functionals of $C_i$) and dual maps

$$\partial_{i+1}^* : C_{i+1}^* \leftarrow C_i^*. \tag{3.3}$$

The *i-th cohomology* is

$$H^i(C) = \frac{\ker \partial_{i+1}^*}{\mathrm{im}\ \partial_i^*} \tag{3.4}$$

where $\dim H_i(C) = \dim H^i(C)$. Moreover, we define

$$\xi_i \triangleq \min\{f_i(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \ker \partial_i \setminus \mathrm{im}\ \partial_{i+1}\} \tag{3.5}$$

$$\zeta_i \triangleq \min\left\{f_i(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \ker \partial_{i+1}^* \setminus \mathrm{im}\ \partial_i^*\right\} \tag{3.6}$$

where $f_i(\boldsymbol{c}) : C_i \to \mathbb{N}$ is a function that counts how many formally summed terms are in $\boldsymbol{c} \in C_i$. For $C_i \subseteq \mathbb{F}_2^m$, $f_i(\cdot)$ coincides with the Hamming weight function $w_{\mathrm{H}}(\cdot)$ on a binary $m$-tuple.

**Example 3.1** (*Topological interpretation of classical linear codes*)**.** Any classical $[n, k, d]$ binary linear code can be seen as a 2-term chain complex

$$C = C_1 \xrightarrow{\partial_1} C_0 \tag{3.7}$$

where $\partial_1 = \boldsymbol{H} \in \mathbb{F}_2^{r \times n}$, with $r \geq n - k$ and $\mathrm{rank}(\boldsymbol{H}) = n - k$, is the linear map given by the parity check matrix, $C_1 = \mathbb{F}_2^n$, and $C_0 = \mathbb{F}_2^r$. We can compute $H_1(C) = \ker \partial_1$ and $\xi_1 = \min\{w_{\mathrm{H}}(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \ker \partial_1\}$. By definition, we have that $\ker \partial_1 \subseteq \mathbb{F}_2^n$ is the space of the codewords which implies that $k = \dim(\ker \partial_1) =$

$\dim(H_1(C))$, $\text{im}\ \partial_1$ is the space of the parity checks (or error syndromes), and the code minimum distance is $d = \xi_1$. Finally, we point out that, when $\boldsymbol{H}$ is full rank, we have $r = n - k$, $\text{im}\ \partial_1 = \mathbb{F}_2^{n-k}$, and $H_0(C) = \mathbb{F}_2^{n-k}/\mathbb{F}_2^{n-k} \cong 0$. On the other hand, when $\boldsymbol{H}$ is not full rank, $r > n - k$, $\text{im}\ \partial_1 \subset \mathbb{F}_2^r$ and in particular $\dim(\text{im}\ \partial_1) = n - k$, and $H_0(C) = \mathbb{F}_2^r/\,\text{im}\ \partial_1 \cong {\mathbb{F}_2}^{r-n+k}$.

**Lemma 2** (*CSS binary construction [103]*)**.** *Consider two linear codes $\mathcal{C}_x$ and $\mathcal{C}_z$ with parameters $[n, k_x]$ and $[n, k_z]$, respectively. If $\mathcal{C}_x^\perp \subseteq \mathcal{C}_z$ there exists an $[[n, k_x + k_z - n, d_\text{X}/d_\text{Z}]]$ quantum code where $d_\text{X} = \min\left\{w_H(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \mathcal{C}_x \setminus \mathcal{C}_z^\perp\right\}$ and $d_\text{Z} = \min\left\{w_H(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \mathcal{C}_z \setminus \mathcal{C}_x^\perp\right\}$.*

Similarly to classical codes, the chain complex formalism can be used to describe CSS quantum codes [41, 43, 110–112]. The following example illustrates this interpretation in detail.

**Example 3.2** (*Topological interpretation of quantum CSS codes*)**.** In general, a CSS code corresponds to a three terms chain complex

$$C = C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \tag{3.8}$$

where $\partial_2 = \boldsymbol{H}_Z^\top \in \mathbb{F}_2^{n \times r_z}$, $\partial_1 = \boldsymbol{H}_X \in \mathbb{F}_2^{r_x \times n}$, $r_z \geqslant \text{rank}(\boldsymbol{H}_Z) = n - k_z$, $r_x \geqslant \text{rank}(\boldsymbol{H}_X) = n - k_x$, $C_2 = \mathbb{F}_2^{r_z}$, $C_1 = \mathbb{F}_2^n$, and $C_0 = \mathbb{F}_2^{r_x}$. In this case, $H_1(C) = \ker \partial_1/\,\text{im}\ \partial_2$, $\xi_1 = \min\{w_\text{H}(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \ker \partial_1 \setminus \text{im}\ \partial_2\}$, $H^1(C) = \ker \partial_2^\top/\,\text{im}\ \partial_1^\top$ and $\zeta_1 = \min\left\{w_\text{H}(\boldsymbol{c}) \,|\, \boldsymbol{c} \in \ker \partial_2^\top \setminus \text{im}\ \partial_1^\top\right\}$. As regards the code parameters, we have by definition that $n = \dim(C_1)$, $\ker \partial_1 \subseteq \mathbb{F}_2^n$ is the space of the codewords on which act only $\boldsymbol{X}$ parity checks ($\mathcal{C}_x$), $\ker \partial_2^\top \subseteq \mathbb{F}_2^n$ is the space of the codewords checked by $\boldsymbol{Z}$ stabilizers ($\mathcal{C}_z$), and $k = \dim(H_1) = \dim(H^1)$. In addition, $\text{im}\ \partial_1$ is the space of the $\boldsymbol{X}$ generators and $\text{im}\ \partial_2^\top$ is the space of the $\boldsymbol{Z}$ stabilizers. The code minimum distance are $d_X = \zeta_1$ and $d_Z = \xi_1$, corresponding to the minimum weight of a nontrivial representative of $H^1$ and $H_1$. Note that, if we ensure $\partial_i \partial_{i+1} = 0$ by construction, then $\mathcal{C}_x^\perp \subseteq \mathcal{C}_z$. Hence, the chain stands for a valid CSS code. On the other hand, if we choose codes such that $\mathcal{C}_x^\perp \subseteq \mathcal{C}_z$ we obtain a valid chain complex. This proves that this structure can be used to represent any CSS code.

To construct a chain complex representing a CSS code it is common to use double complexes of a total complex. For any two chain complexes $C$ and $D$, of length $M$ and $N$, it is possible to define the *double complex* $C \boxtimes D$ as [113, 114]

$$(C \boxtimes D)_{p,q} = C_p \otimes D_q \tag{3.9}$$

where $p = 0, 1, ..., M$ and $q = 0, 1, ..., N$. In this construction we have two types of boundary maps: $\partial_i^v = \partial_i^C \otimes I^D$ and $\partial_i^h = I^C \otimes \partial_i^D$, such that $\partial_i^v \partial_{i+1}^v = 0$, $\partial_i^h \partial_{i+1}^h = 0$ and $\partial_i^v \partial_j^h = \partial_j^h \partial_i^v$.

We can collect vector spaces of equal dimensions by summing along the diagonals, in order to obtain the *total complex*

$$E_n = Tot\,(C \boxtimes D)_n = \bigoplus_{p+q=n} C_p \otimes D_q \tag{3.10}$$

where $n = 0, 1, ..., N + M$. The resulting boundary maps are $\partial^E = \partial^v \oplus \partial^h$. Finally, we can obtain a new chain complex, called *tensor product complex*, starting from $C$ and $D$ as

$$E = C \otimes D = Tot\,(C \boxtimes D)\,. \tag{3.11}$$

Moreover, the Künneth formula gives a method to compute the homology of a tensor product complex from the homology of the original chains

$$H_n\,(C \otimes D) \cong \bigoplus_{p+q=n} H_p(C) \otimes H_q(D). \tag{3.12}$$

In the following, we will use these concepts to describe the cylindrical and Möbius codes.

### 3.3.2 Cylindrical Codes: Design using Topology

As shown in Section 3.3.1, it is possible to describe a quantum CSS code using a 3-term chain. Here we report how to construct CSS codes starting from 2-term chain complexes representing classical linear binary codes. In particular, we detail the construction of the more general family of hypergraph product codes [41], within which the cylindrical code is included.

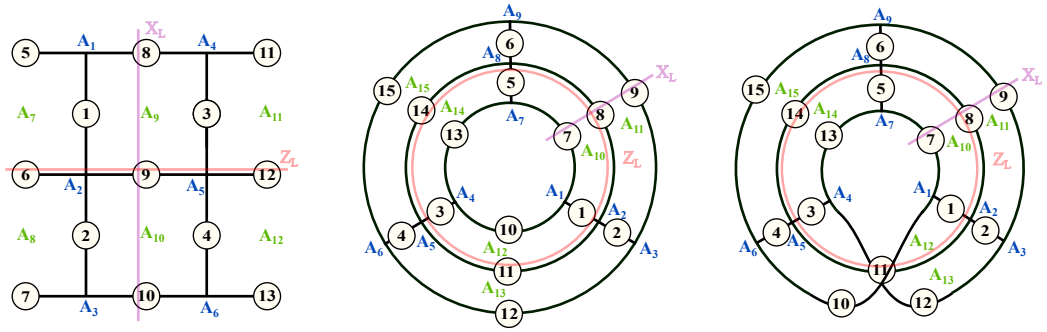


Figure 3.3: (a) $[[13, 1, 3]]$ surface code. Data qubits are depicted as circles, blue ancillas represent $\boldsymbol{Z}$ stabilizers while red ancillas stand for $\boldsymbol{X}$ stabilizers. Examples of $\boldsymbol{X}_{\mathbb{L}}$ and $\boldsymbol{Z}_{\mathbb{L}}$ logical operators are drawn on the lattice. (b) $[[15, 1, 3]]$ cylindrical code. (c) $[[15, 1, 3]]$ Möbius code.

Let us denote with $C$ and $F$ their respective chain complexes

$$C = C_1 \xrightarrow{\boldsymbol{H}_C} C_0 \tag{3.13}$$

$$F = F_1 \xrightarrow{\boldsymbol{H}_F} F_0 \tag{3.14}$$

where $\boldsymbol{H}_C \in \mathbb{F}_2^{r_c \times n_c}$ and $\boldsymbol{H}_F \in \mathbb{F}_2^{r_f \times n_f}$ are the code parity check matrices. At this point, we take the dual of $F$ obtaining the cochain complex

$$F^* = F_0^* \xrightarrow{\boldsymbol{H}_F^*} F_1^* \cong F_0 \xrightarrow{\boldsymbol{H}_F^\top} F_1 \,. \tag{3.15}$$

Considering the isomorphism between chain and cochain, we rewrite $F^*$ as a chain complex $D$, resulting in

$$D = D_1 \xrightarrow{\boldsymbol{H}_F^\top} D_0 \,. \tag{3.16}$$

Having $C$ and $D$ representing our two initial codes, we construct the double com-

plex according to (3.9) as

$$\begin{array}{ccc} C_1 \otimes D_1 & \xrightarrow{\boldsymbol{I}_{n_c} \otimes \boldsymbol{H}_F^\top} & C_1 \otimes D_0 \\ \downarrow{\scriptstyle \boldsymbol{H}_C \otimes \boldsymbol{I}_{r_f}} & & \downarrow{\scriptstyle \boldsymbol{H}_C \otimes \boldsymbol{I}_{n_f}} \\ C_0 \otimes D_1 & \xrightarrow{\boldsymbol{I}_{r_c} \otimes \boldsymbol{H}_F^\top} & C_0 \otimes D_0. \end{array}$$

The tensor product complex $E$ assumes the form

$$E = \underbrace{C_1 \otimes D_1}_{E_2} \xrightarrow{\partial_2^E} \underbrace{C_0 \otimes D_1 \oplus C_1 \otimes D_0}_{E_1} \xrightarrow{\partial_1^E} \underbrace{C_0 \otimes D_0}_{E_0} \tag{3.17}$$

where

$$\partial_2^E = \boldsymbol{H}_Z^\top = \begin{pmatrix} \boldsymbol{H}_C \otimes \boldsymbol{I}_{r_f} \\ \boldsymbol{I}_{n_c} \otimes \boldsymbol{H}_F^\top \end{pmatrix} \tag{3.18}$$

$$\partial_1^E = \boldsymbol{H}_X = \left( \boldsymbol{I}_{r_c} \otimes \boldsymbol{H}_F^\top | \; \boldsymbol{H}_C \otimes \boldsymbol{I}_{n_f} \right). \tag{3.19}$$

Since a tensor product complex is a complex chain, we have a valid CSS by construction (i.e., $\partial_1^E \partial_2^E$). We can also verify that

$$\partial_1^E \partial_2^E = \boldsymbol{H}_X \boldsymbol{H}_Z^\top = \left( \boldsymbol{H}_C \otimes \boldsymbol{H}_F^\top \right) + \left( \boldsymbol{H}_C \otimes \boldsymbol{H}_F^\top \right) = \boldsymbol{0}. \tag{3.20}$$

We use the developed topological framework to describe and evaluate the characteristics of the cylindrical code. To glue the boundaries of a surface code in only one direction and obtain a structure homeomorphic to a cylinder (i.e., an annulus in 2D), we choose two repetition codes, one with full rank parity check matrix and one with a square parity check matrix. Specifically, the complex chain $C$ represents a repetition code $[L, 1, L]$ with $L$ parity checks. Its homologies are $H_1(C) \cong \mathbb{F}_2$ and $H_0(C) \cong \mathbb{F}_2$, since one of the checks is linearly dependent. Furthermore, the complex chain $F$ is also a repetition code $[L, 1, L]$, but with $L - 1$ parity checks. In this case, the code has full rank parity check matrix and the homologies are $H_1(F) \cong \mathbb{F}_2$ and $H_0(F) \cong 0$, since all the checks are linearly independent. Using (3.12) it is straightforward to find the num-

ber of logical qubits encoded by the tensor product code: $k = \dim H_1(E) = \dim(H_0(C) \otimes H_1(D) \oplus H_1(C) \otimes H_0(D)) = 1 \cdot 1 + 1 \cdot 0 = 1$. Moreover, $n = \dim E_1 = \dim(C_0 \otimes D_1 \oplus C_1 \otimes D_0) = L \cdot (L-1) + L \cdot L$, while the distance of the code is still $L$. The resulting CSS code has parameters $[[L^2 + L \cdot (L-1), 1, L]]$.

For example, using $L = 3$ and

$$\boldsymbol{H}_{\mathrm{C}} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad \boldsymbol{H}_{\mathrm{F}} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

we obtain the generators of the $[[15, 1, 3]]$ cylindrical code through (3.18) and (3.19) as

$$\begin{aligned}
\boldsymbol{G}_1 &= \mathbf{X}_1\mathbf{X}_7\mathbf{X}_{10} & \boldsymbol{G}_2 &= \mathbf{X}_1\mathbf{X}_2\mathbf{X}_8\mathbf{X}_{11} & \boldsymbol{G}_3 &= \mathbf{X}_2\mathbf{X}_9\mathbf{X}_{12} \\
\boldsymbol{G}_4 &= \mathbf{X}_3\mathbf{X}_{10}\mathbf{X}_{13} & \boldsymbol{G}_5 &= \mathbf{X}_3\mathbf{X}_4\mathbf{X}_{11}\mathbf{X}_{14} & \boldsymbol{G}_6 &= \mathbf{X}_4\mathbf{X}_{12}\mathbf{X}_{15} \\
\boldsymbol{G}_7 &= \mathbf{X}_5\mathbf{X}_7\mathbf{X}_{13} & \boldsymbol{G}_8 &= \mathbf{X}_5\mathbf{X}_6\mathbf{X}_8\mathbf{X}_{14} & \boldsymbol{G}_9 &= \mathbf{X}_6\mathbf{X}_9\mathbf{X}_{15} \\
\boldsymbol{G}_{10} &= \mathbf{Z}_1\mathbf{Z}_5\mathbf{Z}_7\mathbf{Z}_8 & \boldsymbol{G}_{11} &= \mathbf{Z}_2\mathbf{Z}_6\mathbf{Z}_8\mathbf{Z}_9 & \boldsymbol{G}_{12} &= \mathbf{Z}_1\mathbf{Z}_3\mathbf{Z}_{10}\mathbf{Z}_{11} \\
\boldsymbol{G}_{13} &= \mathbf{Z}_2\mathbf{Z}_4\mathbf{Z}_{11}\mathbf{Z}_{12} & \boldsymbol{G}_{14} &= \mathbf{Z}_3\mathbf{Z}_5\mathbf{Z}_{13}\mathbf{Z}_{14} & \boldsymbol{G}_{15} &= \mathbf{Z}_4\mathbf{Z}_6\mathbf{Z}_{14}\mathbf{Z}_{15}\,.
\end{aligned}$$

The structure of cylindrical codes can be visualized by gluing together two plaquette (or site) boundaries of a surface code (see Fig. 3.3a), including $d - 1$ additional qubits, obtaining an annulus (see Fig. 3.3b). Note that, the qubit indexing of the surface code in Fig. 3.3a is obtained with

$$\boldsymbol{H}_{\mathrm{C}} = \boldsymbol{H}_{\mathrm{F}} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

It is important to note that cylindrical codes are still planar and their generators require only local connectivity, as in surface codes.

### 3.3.3 Möbius Codes: Design using Topology

A possible method to close in a loop a surface code, although admitting $2(d-1)$ non-local measurements, is to attach the boundaries together with a twist. Due to

the particular construction we name these codes as Möbius codes. An example is depicted in Fig. 3.3c. Before proceeding with the construction, let us recall some concepts from the *fiber boundle* literature [43, 113]. In fact, the Möbius codes are particular cases of the family of fiber bundle codes [43].

Let us consider the complex chains $C$ and $D$ in (3.13) and (3.16). Denote by $\mathrm{Aut}(D)$ the finite group of linear automorphisms of the complex $D$, i.e. linear automorphisms of $D_1$ and $D_0$ that commute with the differential $\boldsymbol{H}_F^\top$. Moreover, denote basis vectors of $C_i$ by $c^i$ and write $c^0 \in \boldsymbol{H}_C\, c^1$ if $c^0$ appears with a nonzero coefficient in $\boldsymbol{H}_C c^1$. It is possible to twist the vertical differentials in the double complex $C \boxtimes D$ by an automorphism $\phi(c^1, c^0) \in \mathrm{Aut}(D)$ to every pair $(c^1, c^0)$ such that $c^0 \in \boldsymbol{H}_C\, c^1$. The resulting fiber boundle double complex $C \boxtimes_\phi D$ is

$$\begin{array}{ccc} C_1 \otimes D_1 & \xrightarrow{\boldsymbol{I}_{n_c} \otimes \boldsymbol{H}_F^\top} & C_1 \otimes D_0 \\ \Big\downarrow{\scriptstyle \partial_{\phi_1}} & & \Big\downarrow{\scriptstyle \partial_{\phi_0}} \\ C_0 \otimes D_1 & \xrightarrow{\boldsymbol{I}_{r_c} \otimes \boldsymbol{H}_F^\top} & C_0 \otimes D_0 \end{array}$$

where $\partial_{\phi_i}(c^1 \otimes d^i) = \sum_{c^0 \in \boldsymbol{H}_C\, c^1} c^0 \otimes \phi_i(c^1, c^0)(d^i)$ and $\phi_i(c^1, c^0) \in \mathrm{Aut}(D_i)$.

Aiming to construct a topological code having a Möbius structure, we define as: $i)$ $\boldsymbol{P}_{\mathrm{plaq}}$, a $(L-1) \times (L-1)$ matrix with all elements to zero, except for the elements in the secondary diagonal (also called the anti-diagonal or counter-diagonal), which are ones; $ii)$ $\boldsymbol{P}_{\mathrm{site}}$, a $L \times L$ matrix with all elements to zero, except for the elements in the secondary diagonal, which are ones; $iii)$ $\boldsymbol{S}_x$, a $L \times L$ matrix with all elements to zero, except for the element in position $(x, x)$, which is a one[1]. The matrices $\boldsymbol{P}_{\mathrm{plaq}}$ and $\boldsymbol{P}_{\mathrm{site}}$ are permutation matrices we use to produce the twist of the plaquettes and sites, respectively. The matrix $\boldsymbol{S}_x$ is used to select a single location to perform the cut of the cylindrical structure, enabling the twist operation. Then, a Möbius code is derived from the fiber boundle complex $C \boxtimes_\phi D$ imposing

$$\begin{aligned} \partial_{\phi_0} &= (\boldsymbol{H}_C - \boldsymbol{S}_{(L+1)/2}) \otimes \boldsymbol{I}_{n_f} + \boldsymbol{S}_{(L+1)/2} \otimes \boldsymbol{P}_{\mathrm{site}} \\ \partial_{\phi_1} &= (\boldsymbol{H}_C - \boldsymbol{S}_{(L+1)/2}) \otimes \boldsymbol{I}_{r_f} + \boldsymbol{S}_{(L+1)/2} \otimes \boldsymbol{P}_{\mathrm{plaq}}\,. \end{aligned} \tag{3.21}$$

[1]Here, we assume that both the rows and columns are indexed starting from one.

Finally, the construction proceed as usual, obtaining

$$\partial_2^E = \boldsymbol{H}_Z^\top = \begin{pmatrix} \partial_{\phi_1} \\ \boldsymbol{I}_{n_c} \otimes \boldsymbol{H}_F^\top \end{pmatrix} \tag{3.22}$$

$$\partial_1^E = \boldsymbol{H}_X = \left(\boldsymbol{I}_{r_c} \otimes \boldsymbol{H}_F^\top | \; \partial_{\phi_0}\right). \tag{3.23}$$

Note that, the defined $\boldsymbol{P}_{\text{plaq}}$, $\boldsymbol{P}_{\text{site}}$, and $\boldsymbol{S}_x$ matrices correctly construct a Möbius code if $\boldsymbol{H}_C$ and $\boldsymbol{H}_F$ are defined accordingly. In particular, take the binary vector $\boldsymbol{v}$ of length $L$ in which the first two entries are ones and the other entries are zeros. Therefore, $\boldsymbol{H}_C$ and $\boldsymbol{H}_F$ should be constructed having as a first row the vector $\boldsymbol{v}$, and the other rows are obtained performing a cyclic shift to the right of the above rows. Then, $\boldsymbol{H}_C$ and $\boldsymbol{H}_F$ should be constructed with the first row being the vector $\boldsymbol{v}$, the second row obtained by performing a cyclic shift to the right on the above row, and so on. As an example, using $L = 3$, we can construct the $[[15, 1, 3]]$ Möbius code adopting

$$\boldsymbol{H}_{\text{C}} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad \boldsymbol{H}_{\text{F}} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

similarly to the cylindrical code. Then, considering that

$$\boldsymbol{P}_{\text{plaq}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \boldsymbol{P}_{\text{site}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \boldsymbol{S}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

we obtain from (3.22) and (3.23) the same generators of the cylindrical code, except for

$$\begin{aligned} \boldsymbol{G}_4 &= \mathbf{X}_3\mathbf{X}_{12}\mathbf{X}_{13} \quad \boldsymbol{G}_6 = \mathbf{X}_4\mathbf{X}_{10}\mathbf{X}_{15} \\ \boldsymbol{G}_{12} &= \mathbf{Z}_1\mathbf{Z}_4\mathbf{Z}_{10}\mathbf{Z}_{11} \; \boldsymbol{G}_{13} = \mathbf{Z}_2\mathbf{Z}_3\mathbf{Z}_{11}\mathbf{Z}_{12} \end{aligned}$$

### 3.3.4 Cylindrical Codes: Performance Analysis

For the $[[15, 1, 3]]$ cylindrical code, we compute $L(z) = 6z^3 + 18z^4 + 66z^5 + 228z^6 + 678z^7 + 1836z^8 + 4236z^9 + 7920z^{10} + 11274z^{11} + 11442z^{12} + 7746z^{13} + 3132z^{14} + 570z^{15}$. This code features six fewer logical operators with weight $w = 4$ than the surface code. In particular, for the $[[13, 1, 3]]$ surface code, we find eight $\boldsymbol{ZZZZ}$ logical operators of weight $w = 4$ that cross the lattice from boundary to boundary, as shown in Section 2.3.3. The main advantage of the $[[15, 1, 3]]$ cylindrical code is that these are no longer logical operators, since boundaries are periodic. As an example, using notation from Fig. 3.3a, the $\boldsymbol{Z}_L$ logical operator $\boldsymbol{Z}_3\boldsymbol{Z}_5\boldsymbol{Z}_8\boldsymbol{Z}_{12}$ has no counterpart in the cylindrical code. This is due to the fact that an error of the kind $\boldsymbol{Z}_3\boldsymbol{Z}_7\boldsymbol{Z}_{11}\boldsymbol{Z}_{13}$ in the cylindrical code turns on both ancillas $A_1$ and $A_2$ (see Fig. 3.4a and Fig. 3.4b). Moreover, in the surface code we have two logical operators of weight $2t+2$ of the kind $\boldsymbol{YYZX}$, such as $\boldsymbol{Z}_1\boldsymbol{Y}_5\boldsymbol{X}_7\boldsymbol{Z}_6$, that are no longer present in the cylindrical structure. However, in the $[[15, 1, 3]]$ cylindrical code we can find four $\boldsymbol{X}_L$ logical operators of weight $w = 4$ which are not present in the $[[13, 1, 3]]$ surface code and involve the two additional qubits. For instance, there are two logical operators, i.e., $\boldsymbol{X}_1\boldsymbol{X}_6\boldsymbol{X}_7\boldsymbol{X}_8$ and $\boldsymbol{X}_3\boldsymbol{X}_9\boldsymbol{X}_{10}\boldsymbol{X}_{11}$, crossing the surface code from the bottom boundary to the top one, each making a single rightward turn at the upper part of the lattice (see Fig. 3.4c). As anticipated, looking at the cylindrical code, there are three such patterns, i.e., $\boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{13}\boldsymbol{X}_{14}$, $\boldsymbol{X}_2\boldsymbol{X}_7\boldsymbol{X}_8\boldsymbol{X}_{12}$ and $\boldsymbol{X}_4\boldsymbol{X}_{10}\boldsymbol{X}_{11}\boldsymbol{X}_{15}$ (see Fig. 3.4d). Since a path crossing four qubits from bottom to top can have the turn placed either at the upper or lower part, and it can be directed to the right or left, there is a difference of four logical operators of this type.

We now discuss the logical operator analysis introduced in Section 2.3.3 to obtain exact asymptotic performance for the $[[15, 1, 3]]$ cylindrical code. For larger codes, we defer to Section 3.3.6. To compute the value of $\beta_2$ of the $[[15, 1, 3]]$ cylindrical code, we focus on the logical operators of weight $w = 3$ and $w = 4$, in order to find the fraction of faulty errors of weight $j = 2$. From the undetectable error WE, we have six logical operators with $w = 3$. Specifically, we have three $\boldsymbol{XXX}$ and three $\boldsymbol{ZZZ}$ logical operators (see Fig. 3.3b), all of which exhibit channel errors triggering a failure when composed of two $\boldsymbol{X}$ or two $\boldsymbol{Z}$ errors.

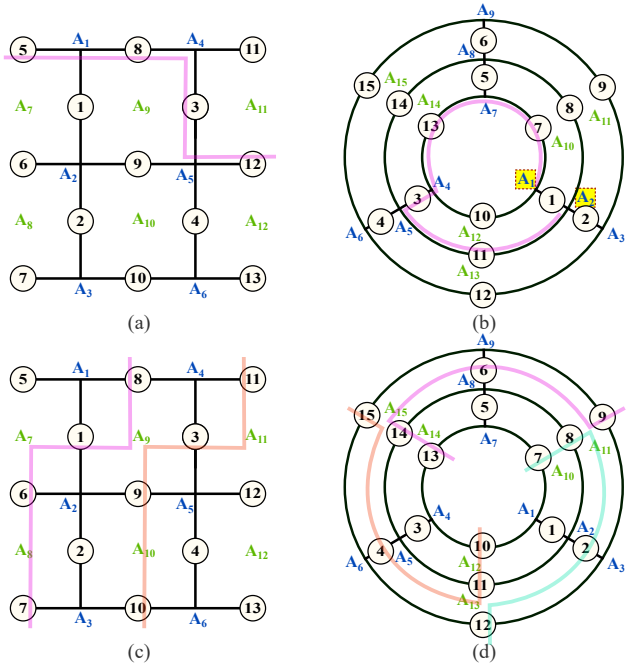


Figure 3.4: Some comparison between logical operators on the surface and the cylindrical codes. The pattern highlighted in (b) is not a logical operator. In (a) and (b) error patterns are composed by $\boldsymbol{Z}$ operators, while in (c) and (d) are composed by $\boldsymbol{X}$ operators.

As for surface codes, $\mu_2^{(3)}(0,2) = \mu_2^{(3)}(2,0) = \mu_2^{(3)}(0,0) = 3$, given that, for instance, a $\boldsymbol{ZZZ}$ logical operator can arise from $\binom{3}{2}$ error patterns of the $\boldsymbol{ZZ}$ type. Furthermore, $\mu_2^{(3)}(0,1) = \mu_2^{(3)}(1,0) = 6$, as a $\boldsymbol{ZZZ}$ logical operator is induced by $2\binom{3}{2}$ error patterns of the $\boldsymbol{ZY}$ kind.

Moving to logical operators of weight $w = 4$, we observe that we have to search for $L_4 = 18$ logical operators. Among them, twelve are composed by $\boldsymbol{XXXX}$, and the remaining six are in the form $\boldsymbol{YYXZ}$. Regarding the $\boldsymbol{YYXZ}$ logical operators, we have a similar behaviour observed for the surface code, i.e., after MWPM decoding, we are always left with a logical operator with three $\boldsymbol{Z}$. Since all the possible errors arising from this kind of logical operators have been already accounted for, we neglect them. To help the visualization of such patterns let us make another example. Consider the $\boldsymbol{X}_1\boldsymbol{Y}_7\boldsymbol{Y}_{10}\boldsymbol{Z}_{13}$ operator. Focusing on channel errors of weight $j = 2$, this logical operator could be triggered by a $\boldsymbol{Y}_7\boldsymbol{Y}_{10}$, since the MWPM decoder applies an $\boldsymbol{X}_1$ and a $\boldsymbol{Z}_{13}$. However, the resulting $\boldsymbol{X}_1\boldsymbol{X}_7\boldsymbol{X}_{10}$ is a stabilizer generator, and we are left with the logical operator $\boldsymbol{Z}_7\boldsymbol{Z}_{10}\boldsymbol{Z}_{13}$. Since we have already accounted for the case in which $\boldsymbol{Z}_L = \boldsymbol{Z}_7\boldsymbol{Z}_{10}\boldsymbol{Z}_{13}$ is generated by $\boldsymbol{Y}_7\boldsymbol{Y}_{10}$ when considering logical operators of weight $w = 3$, we correctly discard it. Regarding the twelve $\boldsymbol{XXXX}$ logical operators, we have already observed three of them in Fig. 3.4d. In fact, these error patterns can be derived enumerating all possible paths traversing from the inside to the outside boundary of the lattice, performing a single turn. This lead to a total of four paths starting from the same inner qubit, that makes twelve when considering that we have three inner qubits (i.e., qubit 7, 10, and 13). In particular, for each logical operator of weight $w = 4$ there are $\binom{w}{j} = \binom{4}{2} = 6$ different patterns of errors of weight $j = 2$ that can cause it. However, one of them is always corrected due to the degeneracy of the code. For instance, let us consider the logical operator $\boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{13}\boldsymbol{X}_{14}$. In case of an error pattern $\boldsymbol{X}_6\boldsymbol{X}_9$, the decoder applies $\boldsymbol{X}_{15}$ and it leads to a stabilizer. Moreover, an error pattern causes a logical operator of weight $w = 3$ that has already been considered. In the previous example, $\boldsymbol{X}_{13}\boldsymbol{X}_{14}$ can be ignored because it generates the logical operator $\boldsymbol{X}_{13}\boldsymbol{X}_{14}\boldsymbol{X}_{15}$ of weight $w = 3$. About the remaining four patterns, they cause, in pairs, the same syndrome. We consider a deterministic decoder, such as the MWPM, that associate one error pattern to each syndrome. Thus, only

Table 3.1: Coefficients for performance evaluation, $[[15, 1, 3]]$ cylindrical code.

| | $i=2$ $\ell=0$ | $i=1$ $\ell=0$ | $i=0$ $\ell=0$ | $i=0$ $\ell=1$ | $i=0$ $\ell=2$ |
|---|---|---|---|---|---|
| $L_3, \gamma_3$ | $3, 1$ | $3, 1$ | $6, 1$ | $3, 1$ | $3, 1$ |
| $\mu_2^{(3)}$ | 3 | 6 | 3 | 6 | 3 |
| $L_4, \gamma_4$ | $0, -$ | $0, -$ | $12, 3/4$ | $12, 3/4$ | $12, 3/4$ |
| $\mu_2^{(4)}$ | $-$ | $-$ | 2 | 4 | 2 |

two patterns will not be corrected. Indeed, the error pattern $\boldsymbol{X}_6\boldsymbol{X}_{13}$ and $\boldsymbol{X}_9\boldsymbol{X}_{14}$, and $\boldsymbol{X}_9\boldsymbol{X}_{13}$ and $\boldsymbol{X}_6\boldsymbol{X}_{14}$ produce the same syndrome in pairs. Hence, we have $\mu^{(4)}(0,2) = \mu^{(4)}(0,0) = 2$, while $\mu^{(4)}(0,1) = 4$. Lastly, among the four patterns of errors of weight $j = 2$ that can cause a logical operator of weight $w = 4$, one is in common with another logical operator. Returning to the example, the two operators $\boldsymbol{X}_L = \boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{13}\boldsymbol{X}_{14}$ and $\boldsymbol{X}_L = \boldsymbol{X}_5\boldsymbol{X}_8\boldsymbol{X}_9\boldsymbol{X}_{13}$ share $\boldsymbol{X}_9\boldsymbol{X}_{13}$. Among the four faulty error patterns corresponding to the pair of logical operators, one pattern is repeated twice. Hence, the value of $\gamma_4(i,\ell) = \frac{3}{4}$. In Tab. 3.1, we report, for the $[[15, 1, 3]]$ cylindrical code, the values of $L(z)$, $\gamma_w(i,\ell)$, and $\mu_j^{(w)}(i,\ell)$ that are needed in order to compute $\beta_2$. If we put these parameters into (2.35), we obtain

$$\beta_2 = \frac{89}{105} \simeq 0.85. \tag{3.24}$$

Moving to a phase flip channel, we need to count only the patterns in the form $\boldsymbol{ZZZZ}$. Since the cylindrical code has no logical operators of such a type with weight $w = 4$, only the three $\boldsymbol{ZZZ}$ operators of weight $w = 3$ remain. Furthermore, for each logical operator, there exist three distinct ways to distribute two errors among the three available locations. This results, for the $[[15, 1, 3]]$ cylindrical code over a phase flip channel, in

$$\beta_2 = 1 - \frac{3\mu_2^{(3)}(0,2)}{\binom{15}{2}} \simeq 0.91\,. \tag{3.25}$$

### 3.3.5 Möbius Codes: Performance Analysis

Employing the procedure described above, we compute the undetectable error WE for the $[[15, 1, 3]]$ Möbius code, resulting in $L(z) = 4z^3 + 18z^4 + 60z^5 + 220z^6 + 666z^7 + 1836z^8 + 4288z^9 + 7968z^{10} + 11280z^{11} + 11378z^{12} + 7668z^{13} + 3156z^{14} + 610z^{15}$. Differently from the cylindrical code, we highlight that only four logical operators with a weight of $w = 3$ are present. Taking as example Fig. 3.3c, the three $\boldsymbol{XXX}$ logical operators are the same of the cylindrical code in Fig. 3.3b. On the other hand, only one $\boldsymbol{ZZZ}$ logical operator of the cylindrical code is still present, i.e., $\boldsymbol{Z}_8\boldsymbol{Z}_{11}\boldsymbol{Z}_{14}$.

Regarding logical operators of weight $w = 4$, we have that all $L_4 = 18$ of them are in the form $\boldsymbol{XXXX}$ Pauli operators. Within them, there are twelve operators that cross the lattice vertically (e.g., $\boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{13}\boldsymbol{X}_{14}$) and six that traverse it horizontally (e.g., $\boldsymbol{X}_2\boldsymbol{X}_3\boldsymbol{X}_6\boldsymbol{X}_{14}$). Let us consider the $\boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{13}\boldsymbol{X}_{14}$ vertical logical operator. Channel errors $\boldsymbol{X}_6\boldsymbol{X}_9$ and $\boldsymbol{X}_9\boldsymbol{X}_{13}$ are corrected by a MWPM decoder realizing the stabilizers $\boldsymbol{X}_6\boldsymbol{X}_9\boldsymbol{X}_{15}$ and $\boldsymbol{X}_2\boldsymbol{X}_3\boldsymbol{X}_9\boldsymbol{X}_{13}$, respectively. Furthermore, the occurrence of the error $\boldsymbol{X}_{13}\boldsymbol{X}_{14}$ results in the logical operator $\boldsymbol{X}_{13}\boldsymbol{X}_{14}\boldsymbol{X}_{15}$, a contribution that is taken into consideration when analyzing operators with a weight of $w = 3$. Moreover, the error patterns $\boldsymbol{X}_6\boldsymbol{X}_{13}$ and $\boldsymbol{X}_9\boldsymbol{X}_{14}$ give rise to identical syndromes, meaning that the MWPM decoder consistently corrects one of these patterns, while the other induces the logical operator. Lastly, the channel error $\boldsymbol{X}_6\boldsymbol{X}_{14}$, turning on the ancillas $A_{11}$ and $A_{14}$, causes the horizontal logical operator $\boldsymbol{X}_2\boldsymbol{X}_4\boldsymbol{X}_6\boldsymbol{X}_{14}$. Hence, we note that a vertical logical operator can arise solely from one specific $\boldsymbol{XX}$ error pattern. Let us shift our attention to the horizontal logical operators, such as $\boldsymbol{X}_2\boldsymbol{X}_3\boldsymbol{X}_6\boldsymbol{X}_{14}$. In this scenario, among the $\binom{4}{2} = 6$ error patterns with a weight of $j = 2$, three pairs of error patterns result in identical syndromes. The decoder successfully corrects half of these patterns, i.e., for each of the six horizontal logical operator there are $\binom{4}{2}/2 = 3$ pattern that trigger an error. Then, we have $\mu^{(4)}(0, 2) = \mu^{(4)}(0, 0) = (12 + 6 \cdot 3)\frac{1}{18} = \frac{5}{3}$, while $\mu^{(4)}(0, 1) = \frac{10}{3}$. In Tab. 3.2, we report, for the $[[15, 1, 3]]$ Möbius code, the values of $L(z)$, $\gamma_w(i, \ell)$, and $\mu_j^{(w)}(i, \ell)$ that are needed in order to compute $\beta_2$. If we put these parameters into (2.35), we

Table 3.2: Coefficients for performance evaluation, $[[15, 1, 3]]$ Möbius code.

| | $i=2$ $\ell=0$ | $i=1$ $\ell=0$ | $i=0$ $\ell=0$ | $i=0$ $\ell=1$ | $i=0$ $\ell=2$ |
|---|---|---|---|---|---|
| $L_3, \gamma_3$ | $3, 1$ | $3, 1$ | $4, 1$ | $1, 1$ | $1, 1$ |
| $\mu_2^{(3)}$ | 3 | 6 | 3 | 6 | 3 |
| $L_4, \gamma_4$ | $0, -$ | $0, -$ | $18, 1$ | $18, 1$ | $18, 1$ |
| $\mu_2^{(4)}$ | $-$ | $-$ | $5/3$ | $10/3$ | $5/3$ |

obtain

$$\beta_2 = \frac{37}{45} \simeq 0.82. \tag{3.26}$$

Since no logical operators in the form $\boldsymbol{ZZZZ}$ exists and only one in the form $\boldsymbol{ZZZ}$ is present, for the $[[15, 1, 3]]$ Möbius code over phase flip channel, we have

$$\beta_2 = 1 - \frac{\mu_2^{(3)}(0, 2)}{\binom{15}{2}} \simeq 0.97\,. \tag{3.27}$$

This shows us that Möbius codes, employing the same number of qubits of cylindrical codes, exhibit superior performance over asymmetric channels.

### 3.3.6 Analytical Upper Bounds

In this section, we provide an upper bound on the logical qubit error rate for both cylindrical and Möbius codes, without limiting the code distance $d$. In order to derive a bound valid for any $d$, we require a closed-form expression for the $L(z)$ coefficients related to the logical operators of weight $2t + 1$ and $2t + 2$. Observing the topological structure of the cylindrical code, we have $2d$ logical operators of weight $2t + 1$: $d$ $\boldsymbol{Z}$ logical operators crossing horizontally, and $d$ $\boldsymbol{X}$ logical operators crossing vertically across the lattice. Two examples of these are depicted in Fig. 3.3b. Moving to logical operators of weight $2t + 2$, we search for the ones composed only by $\boldsymbol{X}$. These logical operators traverse the lattice from the outer ring of the cylindrical structure to the inner ring, performing a single turn along the path. An example of this is given by the logical operator $\boldsymbol{X}_5\boldsymbol{X}_8\boldsymbol{X}_9\boldsymbol{X}_{13}$ in Fig. 3.3b. In general, there are $d$ different starting points (i.e., qubits in the outer

ring) and $d-1$ possible locations where the turn can occur, either to the right or to the left. Then, the logical operators composed only by $\boldsymbol{X}$ Pauli are $2d(d-1)$. Additionally, there are $2d$ logical operators in the form $\boldsymbol{Y}\boldsymbol{Y}\boldsymbol{X}\boldsymbol{Z}\ldots\boldsymbol{Z}$, with $2t-1$ Pauli $\boldsymbol{Z}$. In conclusion, we have that $L_{2t+1}=2d$ and $L_{2t+2}=2d^2$.

Regarding the Möbius code, we have $d+1$ logical operators of weight $2t+1$: one $\boldsymbol{Z}_L$ operator crossing horizontally, and $d$ $\boldsymbol{X}_L$ operators crossing vertically across the lattice (e.g., see Fig. 3.3c). Regarding $\boldsymbol{X}_L$ operators of weight $2t+2$, the ones traversing horizontally the lattice are the same of the cylindrical code, i.e., $2d(d-1)$. On the other hand, there are $d(d-1)$ $\boldsymbol{X}_L$ operators of weight $2t+2$ crossing horizontally the lattice. Indeed, we have $d-1$ starting points (i.e., qubits connecting the outer and the inner rings) and $d$ possible locations where the turn can occur. In conclusion, we have $L_{2t+1}=d+1$ and $L_{2t+2}=3d(d-1)$. Furthermore, we define as $L_w^X$ the total number of $\boldsymbol{X}_L$ operators composed only by $\boldsymbol{X}$ operators of weight $w$. Similarly, $L_w^Z$ is defined as the total number of $\boldsymbol{Z}_L$ operators composed solely of $\boldsymbol{Z}$ operators of weight $w$. Therefore, for cylindrical codes $L_{2t+1}^X = L_{2t+1}^Z = d$ and $L_{2t+2}^X = 2d(d-1)$, while, for Möbius codes $L_{2t+1}^X = d$, $L_{2t+1}^Z = 1$, and $L_{2t+2}^X = 3d(d-1)$. For both codes we have that $L_{2t+2}^Z = 0$.

**Theorem 3.1.** *The value of $\beta_{t+1}$, for a cylindrical or Möbius code of distance $d=2t+1$, can be upper bounded as*

$$\begin{aligned}\beta_{t+1} \geqslant 1 &- \frac{\binom{2t+1}{t+1} L_{2t+1}^Z}{\binom{n}{t+1}} \left(\frac{p_{\mathrm{Z}}+p_{\mathrm{Y}}}{p}\right)^{t+1} \\ &- \frac{\binom{2t+1}{t+1} L_{2t+1}^X + \binom{2t+2}{t+1} L_{2t+2}^X/2}{\binom{n}{t+1}} \left(\frac{p_{\mathrm{X}}+p_{\mathrm{Y}}}{p}\right)^{t+1}\end{aligned} \tag{3.28}$$

*Proof.* Cylindrical and Möbius codes are CSS codes, therefore, $t+1$ errors comprising both $\boldsymbol{X}$ and $\boldsymbol{Z}$ Pauli operators are always corrected. Hence, $t+1$ errors can cause a logical operator of weight $2t+1$ or $2t+2$ only if they are made of either $\boldsymbol{X}$ and $\boldsymbol{Y}$, or $\boldsymbol{Z}$ and $\boldsymbol{Y}$ Pauli operators. As a consequence of this, we have that each error pattern composed by $\boldsymbol{X}$ and $\boldsymbol{Y}$, which triggers an $\boldsymbol{X}_L$, has to be weighted by $(p_{\mathrm{X}}+p_{\mathrm{Y}})^{t+1}$. For $\boldsymbol{Z}_L$ operators, the weighting is $(p_{\mathrm{Z}}+p_{\mathrm{Y}})^{t+1}$. Then, consider that each logical operator of weight $2t+1$, can be caused by $\binom{2t+1}{t+1}$ pos-

sible error patterns of weight $t+1$. Similarly, for each logical operator of weight $2t+2$, there are $\binom{2t+2}{t+1}$ possible error patterns of weight $t+1$. However, due to the MWPM decoding, there are always a pair of different error patterns of weight $t+1$ causing the same syndrome when occurring over the same logical operator of weight $2t+2$. As an example, referring to Fig. 3.3b, the errors $\boldsymbol{X}_5\boldsymbol{X}_{15}$ and $\boldsymbol{X}_7\boldsymbol{X}_{14}$ result in the same syndrome. As a consequence, only half of the $\binom{2t+2}{t+1}$ combinations should be counted. In conclusion, the enumerators in (3.28) are the total number of faulty error patterns comprising $t+1$ Pauli operators, weighted by their probability of occurrence. On the other hand, the common denominator is the total number of error patterns of weight $t+1$ that can occur over $n$ qubits, $\binom{n}{t+1}$, weighted by its probability. We remark that this is an upper bound since we are not taking into account the degeneracy of the code. Specifically, we are not considering the overlapping of logical operators, which reduces the total number of possible error patterns given by the binomials, as explained in Section 2.3.1. □

**Corollary 1.** *For a cylindrical or Möbius code of distance $d = 2t+1$ over an asymmetric channel with asymmetry parameter $A = 2p_{\mathrm{Z}}/(p-p_{\mathrm{Z}})$ and $p_{\mathrm{X}} = p_{\mathrm{Y}}$, the logical error rate can be upper bounded as*

$$p_{\mathrm{L}} \leqslant \frac{(A+1)^{t+1}\binom{2t+1}{t+1}L^{Z}_{2t+1}}{(A+2)^{t+1}}\,p^{t+1} + \frac{2^{t+1}\left[\binom{2t+1}{t+1}L^{X}_{2t+1} + \binom{2t+2}{t+1}L^{X}_{2t+2}/2\right]}{(A+2)^{t+1}}\,p^{t+1}\,. \tag{3.29}$$

## 3.4 Numerical Results

In this section we numerically evaluate the performance of ZZZY codes with the proposed decoder and cylindrical codes under MWPM decoding, providing a comparison with surface codes. For each simulation point in the numerical analysis, $100$ error realizations were tested to ensure sufficient statistical accuracy. In Tab. 3.3 we report the fraction of non-correctable errors for each error class $f_j(i,\ell)$, evaluated by exhaustive search.

*ZZZY codes :* In Fig. 3.5 we report the logical error rates of surface, XZZX, and ZZZY codes with distance $d=3$ and $d=5$ for a physical error rate $p = 0.001$, varying the channel asymmetry. We note that ZZZY codes, while exhibiting com-

Table 3.3: Fraction of non-correctable error patterns $f_j(i, \ell)$.

| **Code** | $XX$ | $XZ$ | $XY$ | $ZZ$ | $ZY$ | $YY$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $[[13, 1, 3]]$ ZZZY | 0.27 | 0.013 | 0.37 | 0.013 | 0.28 | 0.59 | | | | |
| $[[15, 1, 3]]$ Cyl. | 0.257 | 0 | 0.257 | 0.086 | 0.086 | 0.343 | | | | |
| $[[15, 1, 3]]$ Möb. | 0.371 | 0 | 0.371 | 0.029 | 0.029 | 0.400 | | | | |
| $[[25, 1, 3/5]]$ Cyl. | 0.150 | 0 | 0.150 | 0 | 0 | 0.150 | | | | |
| $[[25, 1, 3/5]]$ Möb. | 0.150 | 0 | 0.150 | 0 | 0 | 0.150 | | | | |
| **Code** | $XXX$ | $XXZ$ | $XXY$ | $XZZ$ | $XZY$ | $XYY$ | $ZZZ$ | $ZZY$ | $ZYY$ | $YYY$ |
| $[[41, 1, 5]]$ ZZZY | 0.021 | 0 | 0.021 | 0.001 | 0.005 | 0.021 | $5 \cdot 10^{-4}$ | 0.008 | 0.020 | 0.047 |
| $[[25, 1, 3/5]]$ Cyl. | 0.384 | 0.150 | 0.384 | 0 | 0.150 | 0.384 | 0.013 | 0.013 | 0.163 | 0.397 |
| $[[25, 1, 3/5]]$ Möb. | 0.396 | 0.150 | 0.396 | 0 | 0.150 | 0.396 | 0.004 | 0.004 | 0.154 | 0.401 |
| $[[45, 1, 5]]$ Cyl. | 0.019 | 0 | 0.019 | 0 | 0 | 0.019 | 0.004 | 0.004 | 0.004 | 0.023 |
| $[[45, 1, 5]]$ Möb. | 0.025 | 0 | 0.025 | 0 | 0.150 | 0.0251 | $7 \cdot 10^{-4}$ | $7 \cdot 10^{-4}$ | $7 \cdot 10^{-4}$ | 0.401 |

parable error correction capabilities with respect to surface codes over a depolarizing channel, show a significant performance advantage as the channel asymmetry increases. For $A < 1$ we can just use the dual version of the ZZZY code, which will give the same performance as for $A > 1$. Finally, Fig. 3.6 shows, for $A = 100$, a comparison varying the physical error rate. We observe that for high physical error rate the advantage of ZZZY codes over XZZX codes diminishes.

*Cylindrical and Möbius codes :* Fig. 3.7 shows the asymptotic performance, computed using (2.12), of surface, cylindrical, and Möbius codes of distance $d = 3$ and $d = 5$, as a function of the channel asymmetry parameter $A$. These curves are computed for a value of physical error rate $p = 0.01$. In the case of the $[[13, 1, 3]]$ and the $[[41, 1, 5]]$ surface codes, the logical error rate has a minimum value for $A = 2.9$ and for $A = 1.8$, respectively. Moreover, increasing or decreasing the asymmetry of the channel with respect to this value, reduces the error correction capability of the same amount. Indeed, symmetric surface codes, especially those with distance $d > 3$, do not perform well over channels where one kind of Pauli error happens rather more frequently than the others. On the contrary, for the $[[15, 1, 3]]$ and the $[[45, 1, 5]]$ cylindrical codes, the logical error rate has a minimum for $A = 10.7$ and for $A = 5.3$, respectively. Specifically, for values of asymmetry higher than these minima, the logical error rate increases very slowly. This asses how the error correction capability of cylindrical codes is enhanced over asymmetric channels. The logical error rate of the $[[15, 1, 5]]$ Möbius code has its minimum value for $A = 78.8$, while the $[[45, 1, 5]]$ for $A = 14.9$. Moreover, these codes outperform surface codes across all values

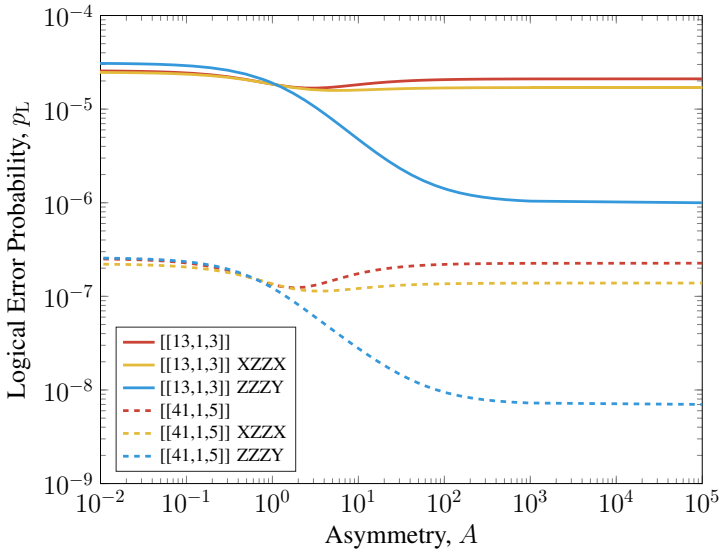


Figure 3.5: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.001$. Surface, XZZX, and ZZZY codes with $d = 3$ and $d = 5$. In the plot are reported the asymptotic approximations (2.12).

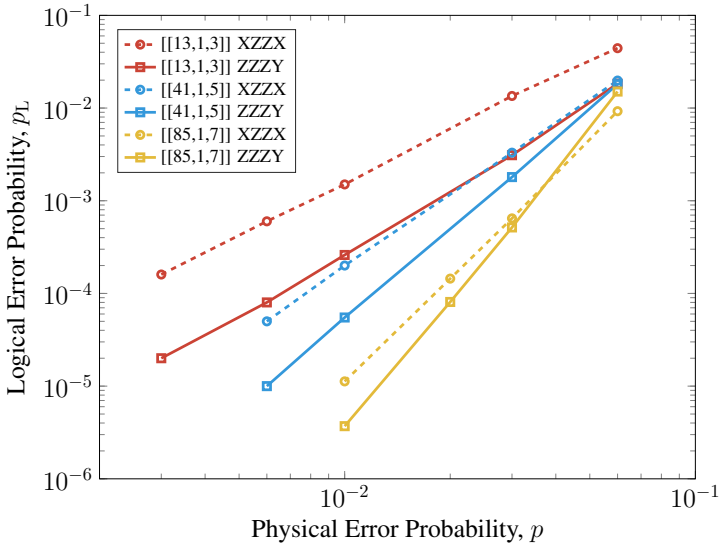


Figure 3.6: Logical error probability vs. physical error rate with channel asymmetry $A = 100$. XZZX, and ZZZY codes with $d = 3$, $d = 5$, and $d = 7$. The curves refer to numerical simulations.

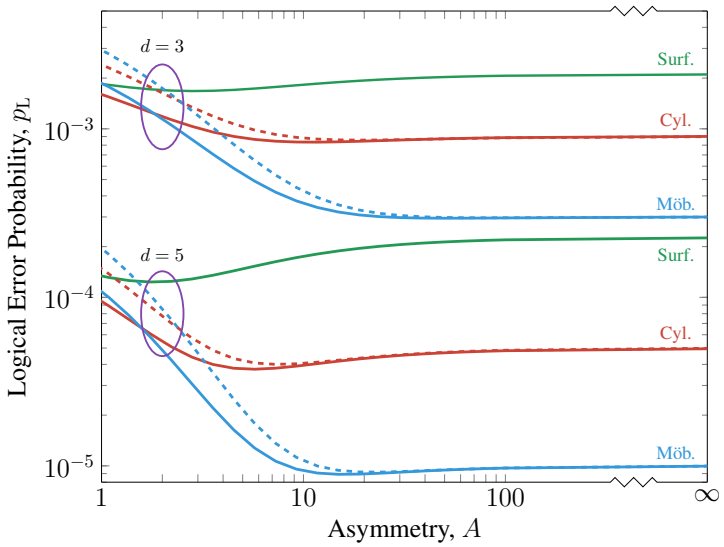


Figure 3.7: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.01$. Surface and cylindrical, and Möbius codes with $d = 3$ and $d = 5$. The curves refer to the exact (2.12) (solid lines) and the upper bound (3.29) (dashed lines).

of channel asymmetry, exhibiting a significant improvement in error correction capability, i.e. exceeding an order of magnitude for moderate asymmetry values. To validate our theoretical findings, we compute bounds on error correction capability of cylindrical and Möbius codes, employing (3.29).

Fig. 3.8 shows the analytical upper bound (3.29) for cylindrical and Möbius codes of distance $d = 7, 9, 11$, for a physical error probability $p = 0.001$. Note that for codes of these dimensions, it becomes impractical to compute the exact asymptotic logical error rate through exhaustive search of faulty error patterns, as was feasible for smaller codes, or to simulate performance using the MWPM decoder. Consequently, this bound represents an important alternative for estimating the logical error probability of these codes, particularly when the channel asymmetry is significantly pronounced.

Fig. 3.9 shows the asymptotic logical error rate of the distance $d = 5$ surface, cylindrical and Möbius codes, over channels with different values of asymmetries. As anticipated, over a depolarizing channel ($A = 1$), the Möbius code's error correction capability is inferior to that of the cylindrical code. Indeed, although Möbius code has only $\binom{d}{t+1}$ faulty $\boldsymbol{Z}$ error patterns (which are part of the

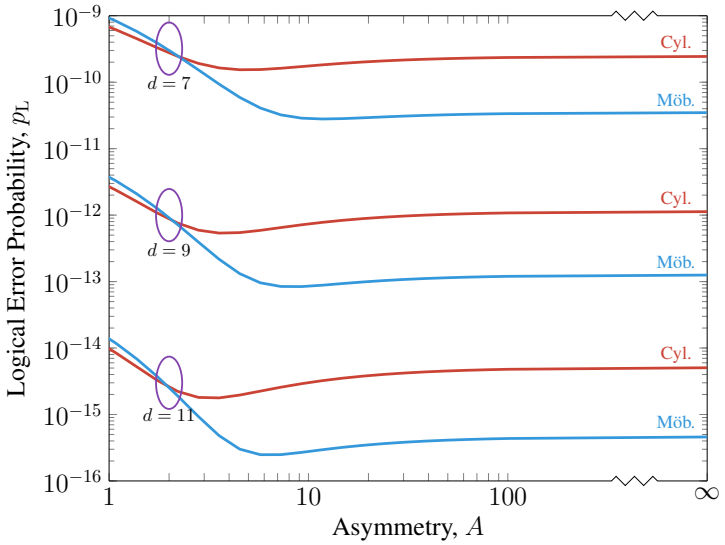


Figure 3.8: Effect of channel asymmetry on the logical error rate for a physical error rate $p = 0.001$. Surface and cylindrical, and Möbius codes with distance $d = 7$, $d = 9$, and $d = 11$. The curves refer to the analytical upper bound (3.29).

untwisted central row), it has a bigger amount of faulty $\boldsymbol{X}$ errors of weight $t + 1$ with respect to the cylindrical version. Moreover, as the asymmetry of the channel increases, the error correction capability of cylindrical and Möbius codes is enhanced. In particular, in the asymptotic condition $A \to \infty$, the $[[45, 1, 5]]$ Möbius code outperforms both surface and cylindrical codes. Additionally, it's worth noting that the performance of the Möbius code over a channel with $A = 10$ is nearly indistinguishable from that of a phase flip channel. This observation underscores the great advantage this structure provides in terms of reducing logical error rates when dealing with slightly asymmetric channels.

Finally, in Fig. 3.10 displays the logical error rates for various asymmetric surface, cylindrical, and Möbius codes. Specifically, for moderately asymmetric channels, the performance of these asymmetric codes is quite similar. However, as the channel asymmetry increases, the Möbius code emerges as the best performer. Note that the slope of the three curves changes moving from $A = 10$ to $A \to \infty$ since these codes have distance $d = 5$ over a phase flip channel.

For the sake of completeness, we also report the threshold values, a metric related to the performance in the high physical error rate regime. In particular,

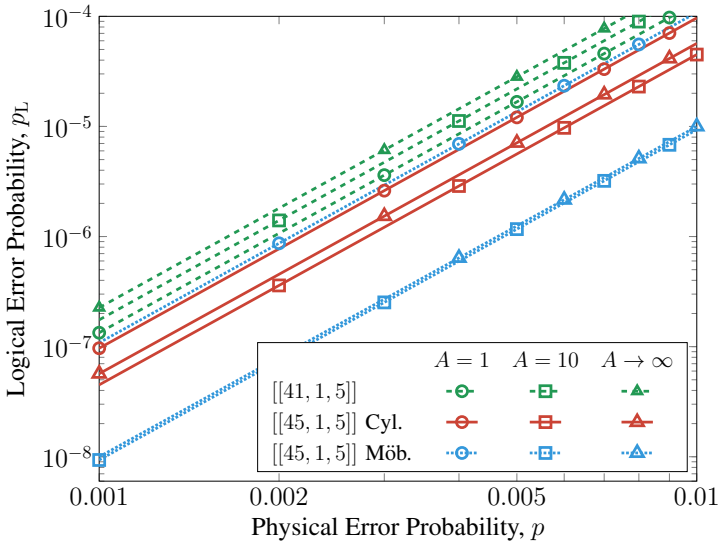


Figure 3.9: Logical error probability vs. physical error probability with channel asymmetry $A = 1$, $A = 10$, and $A \to \infty$. $[[41, 1, 5]]$ surface code, $[[45, 1, 5]]$ cylindrical code, and $[[45, 1, 5]]$ Möbius code. The curves refer to the asymptotic approximations (2.12).

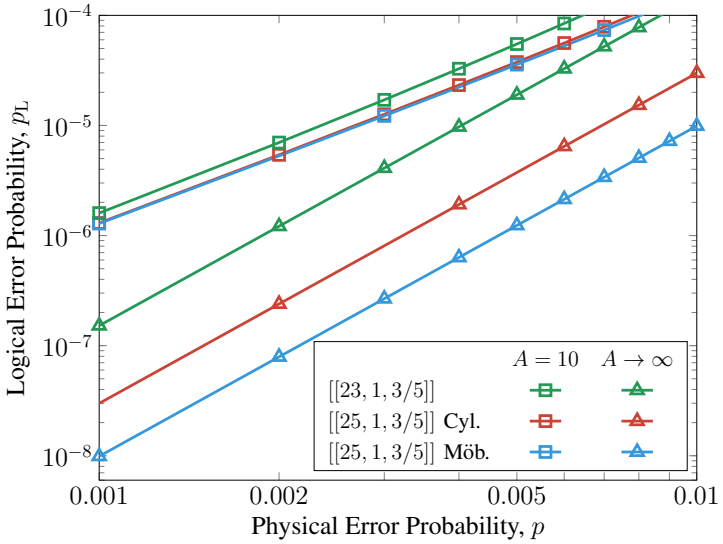


Figure 3.10: Logical error probability vs. physical error probability with channel asymmetry $A = 10$ and $A \to \infty$. $[[23, 1, 3/5]]$ surface code, $[[25, 1, 3/5]]$ cylindrical code, and $[[25, 1, 3/5]]$ Möbius code. The curves refer to the asymptotic approximations (2.12).

Table 3.4: Threshold estimations of Cylindrical and Möbius codes.

| | $A = 1$ | $A = 10$ | $A = \infty$ |
|---|---|---|---|
| Cylindrical Codes | 0.14 | 0.12 | 0.10 |
| Möbius Codes | 0.14 | 0.12 | 0.10 |

Tab. 3.4 shows the threshold values obtained via Monte Carlo simulations for the cylindrical and Möbius codes. The simulations were conducted for channels with varying asymmetry values, revealing that an increase in asymmetry leads to a decrease in the threshold value.

# Chapter 4

# Fast Decoders for Quantum Topological Codes

In recent years, considerable effort has been devoted to identifying quantum codes that are practical to efficient implementation. Among them, one of the most promising approaches in QEC is represented by surface codes [26, 93]. With their high error threshold, locality, and scalability, surface codes have therefore emerged as a leading candidate for implementation in FT quantum computing architectures [47, 49, 53]. The MWPM decoder is currently the prevalent choice for surface code decoding [115, 116]. While it offers high threshold error rates, its high order polynomial time complexity can lead to latency issues, potentially hindering the performance of quantum computation architectures [117, 118]. Indeed, if the decoder is unable to process measurements quickly enough, the accumulating backlog of syndrome information can lead to an exponential increase in computation time [51]. To overcome this latency problem, the PyMatching sparse blossom implementation was introduced, delivering a remarkable speed boost compared to the standard MWPM decoder [52]. Among the alternatives to the MWPM discussed in the literature, the union-find (UF) decoder stands out as a prominent choice [119, 120]. The UF decoding begins by initializing clusters, each containing a single vertex representing an ancilla that has detected an error during the syndrome measurement, often referred to as *defect*. Finally, each cluster is processed in order to obtain a suitable matching. This decoder achieves an

almost-linear worst-case runtime relative to the number of physical qubits. However, despite its ability to correct errors up to the code distance, the UF decoder is less accurate than the MWPM decoder [48, 121]. As evidenced by the numerous proposals in the field, the pursuit of a fast decoder suitable for real-time decoding of surface codes remains a highly active research topic [48, 52, 68, 119–127].

In this chapter, we firstly introduce two fast decoding techniques tailored for surface codes: the spanning tree matching (STM) and Rapid-Fire (RFire) decoders. Specifically, we define the defect graph as a complete graph in which the vertices represent defects, and the edges are weighted based on the error probabilities of the qubits that lie between each pair of defects. We prove that, under a particular metric, all distinct matchings in the defect graph of a surface code are equivalent to the minimum-weight matching. Consequently, using this metric to identify errors ensures accurate correction up to the code minimum distance. Building on this, the STM decoder applies a modified version of the minimum spanning tree (MST) algorithm to a selected subset of ancilla qubits on the lattice. This is followed by an efficient construction of an appropriate perfect matching, yielding the estimated error pattern. The RFire decoder is an even faster approach, optimized for scenarios where decoding speed is critical. Both decoders achieve significantly reduced decoding times compared to the standard LEMON implementation of the MWPM decoder, albeit with some performance trade-offs.

To mitigate these limitations, we introduce the bubble clustering (BC) decoder. This method places each defect at the center of a bubble with a uniform radius, which is carefully chosen to guarantee error correction up to the code distance. Defects are then efficiently grouped into cluster trees, storing only the edges between adjacent defects. Finally, these trees are peeled to obtain an appropriate matching.

## 4.1 Efficient decoders for surface codes

In this section, we introduce several well-known algorithms from the literature that are used for efficiently decoding quantum surface codes.

A matching in a graph is defined as a set of edges where no two edges share a common vertex. A perfect matching is a type of matching that includes every ver-

tex in the graph [128]. A minimum weight perfect matching is a perfect matching with the smallest possible total edge weight. In the context of QEC, the MWPM decoder builds a complete graph where vertices represent defects. The edges in this graph are weighted according to the error probabilities of the qubits that lie between each pair of defects. Given an error syndrome, the standard implementation of the MWPM decoder involves three sequential steps [115]. First, Dijkstra's algorithm is used to assign weights and construct the graph of defects as previously described. Next, the Blossom algorithm is employed to find the minimum weight perfect matching solution [117]. Finally, Dijkstra's algorithm is utilized again to map the paired defects back to chains of faulty qubits in the actual lattice. Note that, when dealing with independent identically distributed (i.i.d.) data qubit errors, the process can be significantly simplified by using the Manhattan distance[1] between defects. Moreover, for surface codes with boundaries, it is essential to introduce ghost defects before applying Dijkstra's algorithm [129]. Indeed, error chains can terminate at a boundary, creating an odd number of defects. To address this, the decoder includes a corresponding ghost defect for each real defect. In the final graph, to ensure the correct decoding procedure, these ghost defects are connected to each other with zero distance. A similar implementation of the MWPM decoder exhibits a worst-case complexity in the number of nodes $N$ in the graph of $O(N^3 \log(N))$, yet empirically, the expected running time for typical instances is approximately $O(N^2)$ [48, 115].

Given that executing the three aforementioned steps sequentially can be computationally demanding, recent proposals have emerged to circumvent the Dijkstra step in constructing the edges of the defect graph. Notably, the PyMatching sparse blossom implementation dynamically discovers and stores an edge only when necessary for the blossom algorithm's operation [52]. This proposal adopts an error model where each error mechanism is defined by the generators and logical operators it affects, rather than by its Pauli type and circuit location. This approach enables the use of techniques such as compressed tracking, a sparse representation of paths in the defect graph. Compressed tracking stores only the endpoints of a path, along with the logical observables it affects (represented as a

[1]The Manhattan distance between two points in an $n$-dimensional space, with coordinates $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, is given by $\sum_{i=1}^{n} |x_i - y_i|$.

bitmask). Specifically, the authors define compressed edges, which denote a path through the defect graph linking two defects or a defect and a boundary. These compressed edges retain information about the two endpoints and the list of logical operators flipped by inverting each edge of the real lattice included in the path. This method enables a highly efficient transition from the final matching to the decoded channel error, resulting from the algorithm. Such an implementation of the MWPM is expected to run between $100$ and $1000$ times faster than the standard implementation [48].

Among the various alternatives to the MWPM proposed in the literature, one of the most notable is the UF decoder [119, 120]. The UF decoder comprises a syndrome validation step and an erasure decoder. First, the process involves transforming the set of Pauli errors into clusters distinguished by an even parity of defects. During this stage, all defects are initialized as separate clusters. Then, in each iteration, every odd cluster extends by half an edge in each direction, facilitating connections between defects rather than between a defect and a boundary. If a cluster encounters another one, they merge, with each defect from the smaller cluster becoming part of the larger one. In particular, the use of a tree representation for each cluster enables efficient execution of this step. When two odd clusters merge, or when a cluster encounters a boundary, they acquire even parity and cease to grow. Finally, after a cycle detection and removal within clusters, the erasure decoder employs a peeling decoder to identify an appropriate matching for each cluster. This decoder shows an almost-linear worst-case running time concerning the number of physical qubits [119]. However, the UF decoder, while offering error correction capability up to the code distance, is less accurate than the MWPM decoder [48, 121].

## 4.2 Spanning Tree Matching Decoder

In this section, we introduce the STM decoder for surface codes. Since the surface codes belong to the class of CSS codes, for the sake of simplicity we will refer to the lattice where the sites constitute the $\boldsymbol{X}$ generators. The same reasoning can be applied to the dual lattice.

The STM decoder consists of three phases: MST evaluation, tree matching

procedure, and error correction.

### 4.2.1 Minimum Spanning Tree Phase

We first construct the complete graph $\mathcal{G} = (n_\mathrm{d}, \binom{n_\mathrm{d}}{2})$ connecting all defects, by using Dijkstra or the Manhattan distances, where $n_\mathrm{d}$ denotes the number of defects. Note that, since there are no ghost ancillas at this stage, this graph has fewer edges and vertices than the one used by the MWPM. Then, we execute the MST algorithm on the graph. This can be achieved with a complexity of $O(M \log N)$, where $M = \binom{n_\mathrm{d}}{2}$ and $N = n_\mathrm{d}$ stand for the edges and the vertices of the graph, respectively [130]. At this stage, we construct two MSTs starting from the obtained one. If the number of defects is already even, one of the output trees is the original one, while the other is derived by introducing a ghost defect on both the left and right sides. If the number of defects is odd, an output tree is obtained by adding a ghost defect to the left, while the alternative output tree is built by adding a ghost defect to the right. Then, the added ghost defects are connected to the nearest defect in the lattice. An example of this phase is reported in Fig. 4.1b. In case of multiple defects with a ghost defect at the same distance, we select the one with the highest minimum distance to other non-ghost defects.

### 4.2.2 Tree Matching Phase

During this step, we match the trees to obtain the estimated error patterns. For a tree $\mathcal{T}$ let us define as $\mathcal{E}$ its perfect matching graph. We use $\mathrm{adj}(v, \mathcal{G})$ to indicate the set of vertices adjacent to the vertex $v$ in the graph $\mathcal{G}$, and $\deg(v, \mathcal{G})$ for the degree of $v$ in the graph $\mathcal{G}$. $\mathcal{B}(\mathcal{T}) = \{v \in \mathcal{T} \,|\, \deg(v, \mathcal{T}) = 1\}$ is the set of boundary vertices in $\mathcal{T}$.

A simple algorithm for obtaining the perfect matching of an MST consists of the following steps.

1. For each $b \in \mathcal{B}(\mathcal{T})$

   Let $a = \mathrm{adj}(b, \mathcal{T})$;
   If $\deg(a, \mathcal{T}) = 2$: call the edges $e_1 = (a, b)$ and $e_2 = (a, \mathrm{adj}(a, \mathcal{T}) \setminus \{b\})$.
   Add $e_1$ to $\mathcal{E}$, and remove the subgraph $(\{a, b\}, \{e_1, e_2\})$ from $\mathcal{T}$.

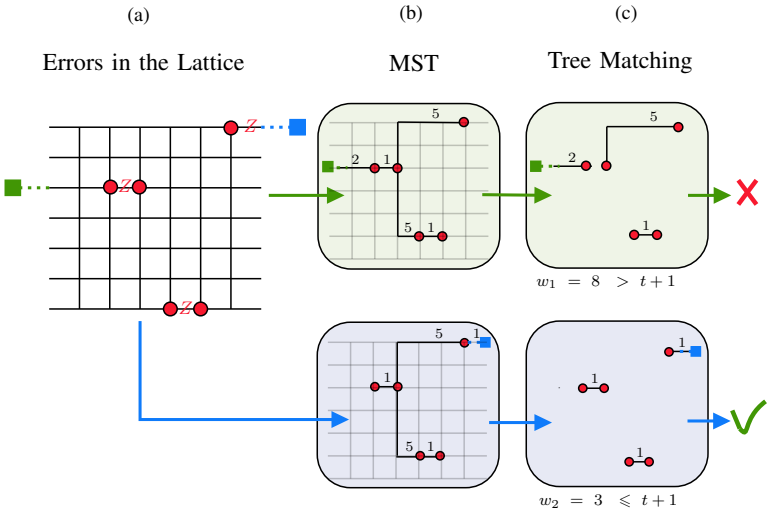


Figure 4.1: Spanning tree matching decoder with a $[[85, 1, 7]]$ surface code. a) Three $Z$ channel errors occur on the lattice. Exited ancillas are depicted in red. b) Two alternative minimum spanning trees obtained with the nearest ghost ancilla to the left (above) and to the right (below) boundary, respectively. c) Resulting $\mathcal{E}$ from the tree matching procedure. The weight $w_2$ of the matching $\mathcal{E}_2$ satisfies $w_2 \leqslant t - 1$; therefore, this matching is chosen as the final solution.

2. For each $b \in \mathcal{B}(\mathcal{T})$

   Let $a = \text{adj}(b, \mathcal{T})$;
   If $\deg(a, \mathcal{T}) = 3$: call $\{v_1, v_2\} = \text{adj}(a, \mathcal{T}) \setminus \{b\}$, and the edges $e_1 = (a, b)$, $e_2 = (a, v_1)$, $e_3 = (a, v_2)$. Add the edge $e_1$ to $\mathcal{E}$ and remove from $\mathcal{T}$ the subgraph $(\{a, b\}, \{e_1, e_2, e_3\})$. Insert the edge $e = (v_1, v_2)$ with weight $w(e) = w(e_2) + w(e_3)$ in $\mathcal{T}$. Return to step 1.

3. For each $b \in \mathcal{B}(\mathcal{T})$

   Let $a = \text{adj}(b, \mathcal{T})$;
   If $\deg(a, \mathcal{T}) = 4$: remove from $\mathcal{T}$ the edge $e = (a, v)$ where $v$ is the sole vertex in $\text{adj}(a, \mathcal{T})$ with $\deg(v, \mathcal{T}) > 1$. Return to step 2.

Note that, considering i.i.d. data qubit errors, a vertex $v$ in the MST cannot have $\deg(v, \mathcal{T}) > 4$. The algorithm terminates when the graph $\mathcal{T}$ becomes empty, and gives a set of edges $\mathcal{E}$ representing a valid solution to the error correction problem. This procedure is carried out for both the MSTs obtained in Section 4.2.1, and the two solutions are indicated as $\mathcal{E}_1$ and $\mathcal{E}_2$. Examples of the tree matching

procedure described above are depicted in Fig. 4.1c and in Fig. 4.2c.

### 4.2.3 Error Correction

After the previous phase we have two possible sets of faulty qubits, $\mathcal{E}_1$, $\mathcal{E}_2$, with a total number of data qubit errors $w_1$ and $w_2$, respectively. An instance of STM decoding, involving both MSTs, is illustrated in Fig. 4.1. If $w_1 \leq t+1$ or $w_2 \leq t+1$, then we have found the correction operator[2]. If, instead, both $w_1$ and $w_2$ have weight $> t+1$, a simple processing considering both $\mathcal{E}_1$ and $\mathcal{E}_2$ can be applied, which guarantees the correction if the channel errors has weight $\leq t$ (see Section 4.2.4). The basic idea is choosing the solution with the smallest number of horizontally traversed edges. This is motivated by the fact that logical operators traverse the lattice from one side to the other one. Hence, the correction with more operators along the horizontal dimension will more likely cause a logical operator.

**Definition 4.1.** A column is the set of horizontal edges aligned in the vertical direction of the lattice, as shown in Fig. 4.2a.

In this way, we can enumerate the columns from left to right ranging from 1 to $d$. Then, we can define two vectors $\boldsymbol{c}_i$ with entries $c_{i,j}$, where $i = 1, 2$ and $j = 1, \ldots, d$, representing the cardinality of the intersection between $\mathcal{E}_i$ and the $j$-th column. Since a solution $\mathcal{E}_i$ with two edges in the $j$-th column is equivalent, by adding a stabilizer, to another solution without any edge in the $j$-th column, we also define as $\boldsymbol{u}_i$ a vector with entries $u_{i,j} = c_{i,j} \mod 2$. Hence, the function

$$f(\mathcal{E}_i) = \sum_{j=1}^{d} u_{i,j} \tag{4.1}$$

can be used as a metric to quantify how much correcting $\mathcal{E}_i$ is likely to cause a logical operator. Thus, the final solution is that minimizing (4.1). For instance, in Fig. 4.2c there are three edges in solution $\mathcal{E}_1$, with weights $w = 1$, $w = 2$, and $w = 5$. They consist of one, two, and three horizontal qubits, respectively. However, the horizontal qubit of the edge with weight $w = 1$ belongs to the same column as the edge with weight $w = 5$. Hence, $f(\mathcal{E}_i) = 4$.

[2]Note that, if $w_1 \leqslant t$, the matching of the second tree can be avoided to save time.

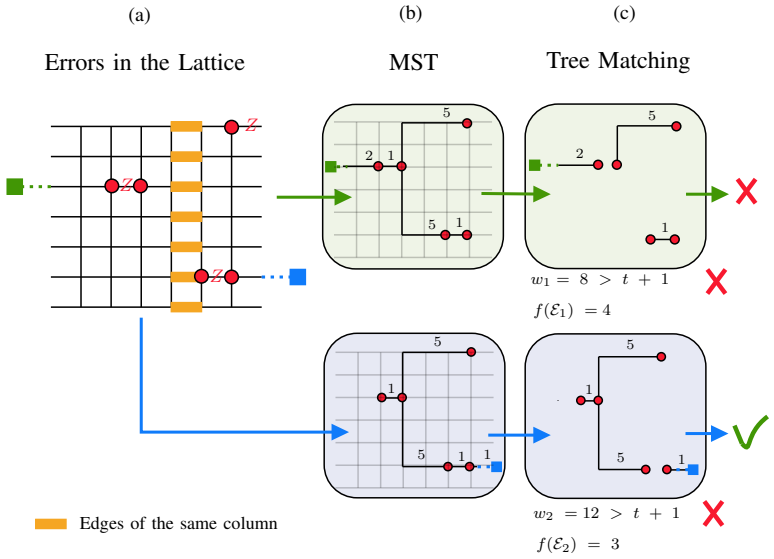


Figure 4.2: Spanning tree matching decoder with a $[[85, 1, 7]]$ surface code. Both matched spanning trees have $w > t+1$. Hence, the error correction is performed according to (4.1).

### 4.2.4 Distance-preserving Decoding

In this section we show that (4.1), in the case of a $[[n, k, d = 2t+ 1]]$ surface code, assures the correction of channel errors with weight $w \leqslant t$.

**Lemma 3.** *Given a surface code, let us call $\mathcal{C}$ an arbitrary Pauli $\boldsymbol{Z}$ error chain connecting two sites defects $v_1$ and $v_2$. Then, the intersection of $\mathcal{C}$ with any column between $v_1$ and $v_2$ has cardinality $1 \pmod 2$. The intersection with any of the other columns has cardinality $0 \pmod 2$.*

*Proof.* The minimal-weight chain linking $v_1$ and $v_2$ clearly satisfies the theorem. Every other chain can be obtained from this one, through the application of plaquette generators. Since each plaquette has two edges in the same column, it does not affect the modulo 2 counting. □

**Lemma 4.** *Given any error pattern over an $[[n, k, d]]$ surface code, resulting in an even number of site defects (including also ghost defects), all possible perfect matchings $\mathcal{M}$ have the same metric $f(\mathcal{M})$.*

*Proof.* We label the horizontal position of the sites in the lattice, from left to right, as $0, 1, \ldots, d + 1$, with $0$ for the left ghost defect and $d + 1$ for the right ghost

defect. Vertically aligned sites share the same label. Let us firstly examine the case where the lattice has four site defects. Reordering them we obtain four indexes $0 \leq j_1 \leq j_2 \leq j_3 \leq j_4 \leq d+1$, representing the site positions. In this setup we can have three possible perfect matchings $\mathcal{M}_i$, with $i = 1, 2, 3$. Considering $\mathcal{M}_1$ as the one connecting site $1$ with $2$, and site $3$ with $4$, by application of Lemma 3, we have that $f(\mathcal{M}_1) = (j_2 - j_1) + (j_4 - j_3)$. It is easy to check that the other two matchings have the same metric. Finally, we observe that each perfect matching $\mathcal{M}$ in any graph $\mathcal{G}$ with an even number of defects can be obtained by iteratevely removing two edges and reconnecting them as needed. Hence, the claim follows since this operation does not affect the metric $f(\mathcal{M})$. □

**Corollary 2.** *Each perfect matching has the same metric* (4.1) *as the MWPM. It follows that* (4.1) *can be interpreted as the minimum number of traversed columns for all perfect matchings of the same graph.*

We have seen in Section 4.2.1 that, starting from a complete graph $\mathcal{G}$ formed by connecting all defects, we obtain two distinct graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ by adding ghost ancillas. Due to the even number of defects in these graphs, we can compute two perfect matchings, denoted as $\mathcal{E}_1$ and $\mathcal{E}_2$. It is noteworthy that applying a logical operator to one matching we get the other. Consequently, one of the solutions will correct the error (call it $\mathcal{E}_c$), while the alternative, $\mathcal{E}_a$, will produce a logical error.

**Theorem 4.1.** *Let us consider an $[[n, k, d]]$ surface code and an error pattern of weight $w \leq t$. A perfect matching leading to the correct solution, $\mathcal{E}_c$, has $f(\mathcal{E}_c) < f(\mathcal{E}_a)$, with $\mathcal{E}_a$ being any perfect matching on the alternative graph.*

*Proof.* The number of columns in the lattice is $d$. Since we are considering error patterns with weight $w \leq t$, we have that any perfect matching $\mathcal{E}_\mathrm{c}$ representing the correct solution, due to Lemma 4, satisfies $f(\mathcal{E}_\mathrm{c}) \leq t$. Regarding the alternative solution $\mathcal{E}_\mathrm{a}$, we have that $f(\mathcal{E}_\mathrm{a}) = d - f(\mathcal{E}_\mathrm{c})$, since they differ by a logical operator. This leads to $f(\mathcal{E}_\mathrm{a}) \geq t + 1$ for $d$ odd and $f(\mathcal{E}_\mathrm{a}) \geq t + 2$ for $d$ even, proving the statement. □

The previous theorem states that a surface decoder chosing a matching with the minimum traversed column metric (4.1) preserves the error correction capability

of the code. The same theorem applies to rotated surface codes as well, owing to their similar lattice structure.

## 4.3 Rapid-fire Decoder

We show now how to design an even faster decoder, the RFire decoder, that ensures the correction for all errors with weights $w \leqslant t$. Due to Corollary 2, each matching is equivalent to the minimum one in terms of (4.1). Hence, it is possible to compute $\mathcal{E}_i$ with $i = 1, 2$ without evaluating and matching the MST. Firstly, we compute the complete graph $\mathcal{G}$ on the defects. Then, we construct two graphs $\mathcal{S}_i$ by adding to $\mathcal{G}$ ghost ancillas on boundaries with the same strategy described in Sections 4.2.1. For each $\mathcal{S}_i$, we iteratively pair each defect with its closest one in a greedy fashion, we update the solution $\mathcal{E}_i$, and we remove both the defects from $\mathcal{S}_i$. This process results in two potential sets of faulty qubits, $\mathcal{E}_1$ and $\mathcal{E}_2$. Finally, we determine the error correction operator to adopt, following the procedure outlined in Section 4.2.3. In this way, we guarantee the correction of all errors of weight up to $w = t$ by Theorem 4.1.

## 4.4 Bubble Clustering Decoder

Although STM and the RFire decoders are able to guarantee the error correction capability $t$ of surface codes, due to some uncorrected error patterns of weight $t + 1$, a gap in performance between these decoders and the MWPM arises. We observe that several error patterns, among the ones causing the performance discrepancy, have errors located far away to each other in the lattice grid. For instance, the error patterns shown in Fig. 4.3 cannot be decoded by the STM or RFire decoders, whereas the MWPM decoder successfully decodes them. To address this issue, we clustered the defects into smaller subsets using an ad-hoc radius, enabling the decoding of multiple erroneous patterns and approaching the performance of MWPM. Moreover, to further reduce execution times, we developed an algorithm that, while performing the clustering, simultaneously generates a series of trees, i.e., connected graphs without cycles. In the following sections,

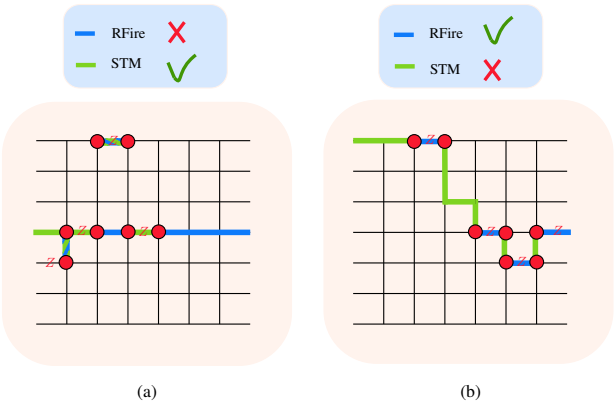


Figure 4.3: Examples of non corrected error patterns in a $[[85, 1, 7]]$ surface code for the RFire and STM decoders. Each faulty qubit is represented by a red $Z$ symbol, while defects are illustrated as red dots. The correction operators applied by the RFire decoder are represented by thick blue edges, whereas those applied by the STM decoder are represented by green edges. a) An error pattern of weight $t + 1$ that is not corrected by the RFire decoder but is corrected by the STM decoder. b) An error pattern of weight $t + 1$ that is not corrected by the STM decoder but is corrected by the RFire decoder. Both error patterns are corrected by the BC decoder.

we delve into the details of our BC decoder that starts from the error syndrome and returns an estimated error pattern. In particular, the four phases composing our solution are: radius evaluation phase, a bubble clustering phase, a peeling phase, and an error correction phase.

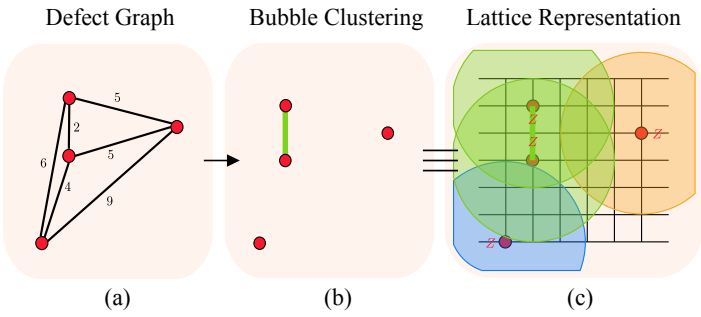


Figure 4.4: Example of bubble clustering phase for a $[[85, 1, 7]]$ surface code.

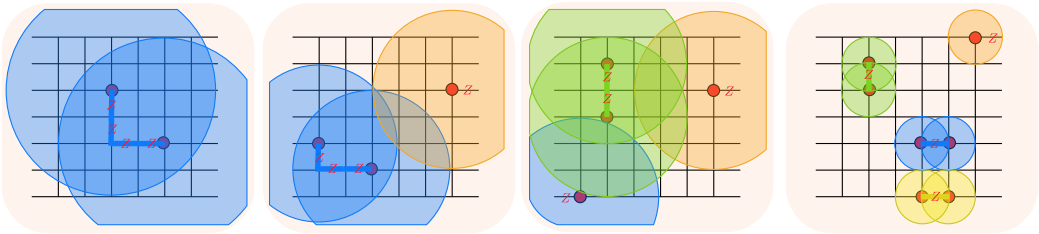


Figure 4.5: Example of bubble clustering phase for a $[[85, 1, 7]]$ surface code. A number of $\boldsymbol{Z}$ errors occurred on the lattice resulting in defects depicted as red circles. Different clusters are depicted with different colors, and adjacent defects are connected by edges of the corresponding color. a) Four $\boldsymbol{Z}$ errors with $n_{\mathrm{d}} = 2$ and $R_{\mathrm{sph}} = 4$, resulting in one single cluster. b) Four $\boldsymbol{Z}$ errors with $n_{\mathrm{d}} = 3$ and $R_{\mathrm{sph}} = 3$, resulting in two different clusters. c) Four $\boldsymbol{Z}$ errors with $n_{\mathrm{d}} = 4$ and $R_{\mathrm{sph}} = 3$, resulting in three different clusters. d) Four $\boldsymbol{Z}$ errors with $n_{\mathrm{d}} = 7$ and $R_{\mathrm{sph}} = 1$, resulting in four different clusters. Note that two defects can belong to the same cluster if they both lie within the region where their corresponding bubbles intersect, or if a third defect exists such that the distance between each defect and the third defect is less than or equal to the radius.

### 4.4.1 Bubble Radius Evaluation

The primary goal of this stage is to determine the optimal radius to cluster the defects on the lattice. Specifically, each defect is placed at the center of a bubble with uniform size. This step is critical because defects located in different bubbles will never be paired in the final solution.

From (2.12), we observe that the performance of an $[[n, k, d]]$ code, with $t = \lfloor (d-1)/2 \rfloor$, is asymptotically determined by the fraction of errors of weight $w = t + 1$ that the decoder can correct. The main idea is to exploit the number of defects to determine the smallest possible radius of the bubbles such that, given any error pattern of weight $\leqslant t + 1$, the clustering is carried out in such a way that it never compromises the error correction capability $t$ and aids as much as possible the decoding of $t + 1$ error patterns. For instance, in the case of a $t + 1$ chain of adjacent errors, resulting in only a pair of defects, we would like to have them inside the same cluster. For this reason, we have that the radius must be at least $R_{\mathrm{sph}} = t + 1$ when the number of defects is $n_{\mathrm{d}} = 2$. This ensures that the two defects belong to the same cluster, allowing them to be connected in subsequent steps of the decoding process. Therefore, a simple but suboptimal solution is to set this radius to $R_{\mathrm{sph}} = t + 1$. However, if there are more than two defects

in the lattice, this radius can be reduced to potentially improve the error correction capability of errors with weight $t + 1$. As a simple example, adopting this choice, both the error patterns in Fig. 4.3 would form a single cluster, leading to the performance of one of the decoders presented in [68].

In general, the radius should be determined by evaluating whether two defects need to be connected or not in the solution. For clarity, let us first consider a lattice with three defects. When focusing on two of these defects, they must be connected if their distance is at most $t + 1$ minus the weight of the edges connecting the remaining defect to a boundary. This ensures that the error correction capability is not compromised. Since there is a single unpaired defect, in the worst-case scenario, it could be connected to a boundary via an edge of weight one, resulting in a bubble radius of $t$. This scenario represents the most conservative case, where the radius is the largest possible, leading to fewer clusters overall. Indeed, if a single cluster is generated, the error correction capability becomes similar to that of the RFire decoder. Similarly, for the case with four defects, two defects must be connected if their distance is at most $t + 1$ minus the weight of the edges connecting the other two defects. In the worst-case scenario, these defects are connected to each other via an edge of weight one, which results again in a radius of $t$. Generalizing this trend, we observe that from the maximum value of the radius $t + 1$ it is possible to subtract an edge of weight one for each pair of defects, apart from the initial pair that is being analyzed, to determine if they should be connected. Therefore, the minimum value of the radius, while satisfying the conditions outlined above, will be $t + 1$ minus $\lceil n_{\mathrm{d}}/2 \rceil - 1$, leading to

$$R_{\text{sph}} = t + 2 - \left\lceil \frac{n_{\mathrm{d}}}{2} \right\rceil . \tag{4.2}$$

### 4.4.2 Bubble Clustering Phase

During this stage, utilizing the information provided by the syndrome $\boldsymbol{s}$ along with the calculated $R_{\text{sph}}$, all defects are organized into one or more clusters. In doing so, a subset of edges from the lattice is assigned to each cluster, ensuring the formation of trees, thus preventing the occurrence of any cycles. To formalize the algorithm, let us define $\boldsymbol{v}$ as the column vector of size $n_{\mathrm{d}}$ containing the indexes

**Algorithm 3:** Bubble Clustering Phase

**input :** $n_{\mathrm{d}}$, $\boldsymbol{v}$, $R_{\mathrm{sph}}$;
**output:** $\boldsymbol{A}$, adjacency matrix;
$\boldsymbol{p}$, indexes of the clusters for each defect;
$\boldsymbol{c}$, cardinality of each cluster;
$\boldsymbol{L}$, matrix of clusters;
$\boldsymbol{o}$, order of each defect;

```
init A, p, c to all zeros;
N_c ← 0;
totalDef ← 0;
while totalDef ⩽ n_d do
    contDefect ← 1;
    while p[contDefect] ≠ 0 do
        contDefect ← contDefect + 1;
    N_c ← N_c + 1;
    totalDef ← totalDef + 1;
    c[N_c] ← c[N_c] + 1;
    L[N_c][c[N_c]] ← v[contDefect];
    p[v[contDefect]] ← N_c;
    flagEndCluster ← 0;
    currentDef ← 1;
    while flagEndCluster = 0 do
        flagEndCluster ← 1;
        forall n_{d_i} ∈ v do
            if evalD(L[N_c][currentDef], n_{d_i}) ⩽ R_sph
                if p[n_{d_i}] = 0
                    A ← adjDef(currentDef, n_{d_i});
                    o[L[N_c][currentDef]] ← o[L[N_c][currentDef]] + 1;
                    o[n_{d_i}] ← o[n_{d_i}] + 1;
                    c[N_c] ← c[N_c] + 1;
                    L[N_c][c[N_c]] ← n_{d_i};
                    p[n_{d_i}] ← N_c;
                    totalDef ← totalDef + 1;
                    flagEndCluster ← 0;
        if flagEndCluster = 0
            currentDef ← currentDef + 1;
```

of each defect, and $\boldsymbol{L}$ as the matrix containing in the $i$-th row the list of the defects belonging to the $i$-th cluster. The size of $\boldsymbol{L}$, $N_c \times n_{\mathrm{d}}$, is determined during the clustering phase depending on the number of clusters needed.

The procedure unfolds as follows. During the initialization step, the first defect $v_1$ is added to the first row of $\boldsymbol{L}$, referred to as $\boldsymbol{l}_1$. Subsequently, the algorithm systematically examines whether any other defect lies within a distance less than or equal to $R_{\text{sph}}$ from $v_1$. If, for instance, this condition is verified for the $j$-th defect, then $v_j$ is included in $\boldsymbol{l}_1$. The proposed goal can be effectively achieved employing the function `evalD`, which computes the distances between two vertices in the grid, calculated based on their integer coordinate positions, enabling precise and efficient measurement of spatial relationships. In particular, defining as $q_i$ and $r_i$ as the quotient and the remainder of the integer division between $s_i$ and $2t$, the function $\texttt{evalD}(s_i, s_j)$ returns $|q_i - q_j| + |r_i - r_j|$. Additionally, we employ the function `adjDef` to record that defects $v_1$ and $v_j$ are adjacent to each other using an $n_{\mathrm{d}} \times n_{\mathrm{d}}$ adjacency matrix $\boldsymbol{A}$. Then, the same procedure is sequentially applied to all the other defects added to $\boldsymbol{l}_1$. Starting from the second defect of each cluster, we have to verify whether a particular defect is already present in the corresponding list before adding it. This can be performed by maintaining an array storing the cluster membership of each defect, thereby preventing cycles. Note that a defect that is already part of a cluster, meaning it is adjacent to at least one other defect, can only become adjacent to additional defects when considering the bubble centered around that defect. This is essential to ensure that the algorithm produces a tree structure. For each defect, the column vector $\boldsymbol{p}$ of size $n_{\mathrm{d}}$ stores the index of the cluster to which it belongs. Also, the cardinality of each cluster is stored in a column vector $\boldsymbol{c}$ of size $N_c$.

Once all defects in $\boldsymbol{l}_1$ have been processed, if there are any remaining defects not yet included in the list, the first of these defects is added to $\boldsymbol{l}_2$, and a new cluster is initialized. This process iterates until all defects have been assigned to a cluster, resulting in a total of $N_c$ clusters. An instance of the bubble clustering procedure, illustrating the defect graph, is shown in Fig. 4.4. For clarity, we will henceforth use the lattice representation notation. Some examples of bubble clustering procedures for the $[[85, 1, 7]]$ surface code are depicted in Fig. 4.5. This bubble clustering stage is outlined in Algorithm 3. During the algorithm execu-

tion, a vector $\boldsymbol{o}$ is initialized to record the order of each defect.

### 4.4.3 Peeling Phase

**Algorithm 4:** Peeling Phase

**input :** $\boldsymbol{s}$, $n_{\mathrm{d}}$, $\boldsymbol{A}$, $\boldsymbol{p}$, $\boldsymbol{c}$, $\boldsymbol{L}$;
**output:** $\{\mathcal{E}_1^1, \ldots, \mathcal{E}_{N_c}^1\}$, list of the matchings;
$\{w_1^1, \ldots, w_{N_c}^1\}$, list of the weights;

**forall** $i \in \{1, \ldots, N_c\}$ **do**
  clusterDim $\leftarrow \boldsymbol{c}[i]$;
  idx $\leftarrow$ `addGhost`$(i)$;
  `buildMatch`(idx);
  $\boldsymbol{s}$[idx] $\leftarrow 0$;
  **while** $\boldsymbol{c}[i] > 0$ **do**
    **forall** $j \in \{1, \ldots, \mathrm{clusterDim}\}$ **do**
      **if** $\boldsymbol{o}[\boldsymbol{L}[i][j]] = 1$
        idxAdj $\leftarrow$ `Adjacent`$(\boldsymbol{L}[i][j])$;
        **if** $\boldsymbol{s}[\boldsymbol{L}[i][j]] = 1$
          `buildMatch`$(\boldsymbol{L}[i][j], \mathrm{idxAdj})$;
          $\boldsymbol{s}[\boldsymbol{L}[i][j]] \leftarrow 0$;
          $\boldsymbol{s}$[idxAdj] $\leftarrow 1 - \boldsymbol{s}$[idxAdj];
        **if** $\boldsymbol{o}$[(idxAdj] $= 1$
          $\boldsymbol{c}[i] \leftarrow \boldsymbol{c}[i] - 1$;
        `peel`$(\boldsymbol{L}[i][j], \mathrm{idxAdj})$;
        $\boldsymbol{c}[i] \leftarrow \boldsymbol{c}[i] - 1$;

The goal of this step is to begin with the obtained tree and generate a suitable matching for each cluster. Then, in the final post-processing phase, each matching will be evaluated using specific metrics, after which it will either be selected as the final solution or set aside for further consideration. In the latter case, another solution is generated and an additional peeling phase will be required for that cluster (see Section 4.4.4). Specifically, the peeling phase, described in Algorithm 4, consists of two distinct sections: a pre-processing stage and a matching stage. Note that the processing described below is performed independently for each cluster, where a tree has been generated by the previous step if more than

one defect resides in the cluster. In the following, we will define the order of a defect as the number of its adjacent defects in the tree of its cluster. A defect of order one is called *boundary* defect.

**Algorithm 5:** `addGhost`

**input :** $i$, cluster index;
**output:** $s_i$, $s_j$, vertices connected to the ghost ancillas;

**if** *first peeling phase*
  **if** *cluster* $i$ *has an odd number of defects*
    attach a ghost ancilla to the nearest defect $s_i$;
**else**
  **if** *cluster* $i$ *has an odd number of defects*
    attach a ghost ancilla to the nearest defect $s_i$ on the opposite boundary with respect to the boundary selected in peeling phase;
  **else**
    attach two ghost ancillas: one to the nearest defect $s_i$ on the left boundary, and one to the nearest defect $s_j$ on the right boundary;

*Ghost ancillas addition:* This step is required to assure that each cluster tree comprises an even number of defects and then is applied for all clusters with an odd number of defects. Indeed, since surface codes feature non-periodic boundaries, it is possible for an error chain to terminate at a boundary, resulting in the creation of just a single defect. In such cases, obtaining a suitable match is not feasible. Therefore, if the cardinality of a cluster is odd, a ghost ancilla is added to one of its defects according to the following criteria. The required ghost ancilla is connected to the defect closest to a boundary. If multiple defects are equidistant, the ghost ancilla is attached to the defect that is farthest from the other defects in the cluster. In particular, the defect that is farthest from the others in the cluster is the one with the maximum distance from its cluster nearest neighbor. Additionally, we retain in memory the specific boundary (i.e., left or right) to which the ghost ancilla is attached. In case of a second peeling solution, if the $i$-th cluster contains an even number of defects, we include a ghost ancilla on both the right and left boundaries. This is done to preserve the even parity of the defects. On the other hand, if the number of defects is odd, we add a ghost ancilla to the right (left) boundary if the cluster was attached to the left (right) boundary during the previ-

ous peeling phase. The criteria for determining which defects these ghost ancillas connect to remain the same. For conciseness, we utilize the function `addGhost` which takes as input the cluster index and returns the index of the adjacent defect as previously outlined (see Algorithm 5). We remark that the algorithm handles defects sequentially from left to right and top to bottom. As a result, when two defects are equidistant from the boundary and have the same distance from the nearest defect, the ghost ancilla is always assigned to the first one encountered in this order. Note that all the required information can be stored during the clustering phase. Here it has been presented separately for the sake of presentation clarity.

**Algorithm 6:** `buildMatch`

**input :** $s_i$, $s_j$, vertices;
$k$, index of the cluster;
**output:** $\mathcal{E}_k$, current solution (edge/qubit set);

Compute the vertical and horizontal coordinates of $s_i$ and $s_j$ using modular arithmetic;
Start from $s_i$;
Move step by step along the vertices in the vertical direction, until reaching a vertex $s_k$ that has the same vertical coordinate as $s_j$;
For each step, include in $\mathcal{E}_k$ any edge that connects two consecutively visited vertices;
Start from $s_k$;
Move step by step along the vertices in the horizontal direction, until reaching $s_j$;
For each step, include in $\mathcal{E}_k$ any edge that connects two consecutively visited vertices;

*Tree peeling:* In this stage, for each cluster $i$-th, the corresponding tree is iteratively peeled to achieve a first suitable matching $\mathcal{E}_i^1$. For this purpose, we define as vertices the defects present in the current cluster, all initially set to be switched on (i.e., set $s_i$ to one). For each cluster, there are two possibilities: it could contain either an even or an odd number of vertices. If the number is odd, we must add edges between the vertex selected by `addGhost` and the nearest boundary to complete the final matching. After this, we switch off (i.e., set $s_i$ to zero) the vertex selected by `addGhost`. In this way, we have ended up to an even number of vertices, allowing the algorithm to proceed identically for both possibilities. In Algorithm 4, we employ the function `buildMatch` to describe

this matching. This function takes as input the indices of two vertices and inserts all the qubit edges between them into the final solution (i.e., the qubit in the path composing the edge of the tree). On the other hand, if the function is called with only one single index, it adds the edges between the input vertex and the nearest boundary to the final matching. In this way, we connect it to the ghost ancilla in that cluster. This can be efficiently handled by using simple modular arithmetic operations, which exploit the regular structure of the lattice to directly compute qubit indices without requiring complex lookups or additional overhead.

After this pre-processing, the algorithm processes each boundary vertex (i.e., a vertex of order one) within the cluster as follows. To identify it, we employ the vector $\boldsymbol{o}$, prepared during the clustering phase, which contains the order of each defect. For each of the boundary vertices, we have one adjacent vertex indexed by `idxAdj`, retrieved using the function `Adjacent`. Then, if the current boundary vertex is switched on, we add its incident edge to the final matching $\mathcal{E}_i^1$, and its adjacent vertex is flipped (i.e., $s_i \leftarrow 1 - s_i$). Afterward, even if the current boundary vertex was switched off, we remove the incident edge from the graph and update the vertex orders accordingly. This is accomplished using the function `peel`. Finally, before proceeding with the next boundary vertex, we also update the cardinality of the cluster $\boldsymbol{c}[i]$. The desired matching among the initial defects is achieved as soon as the tree becomes an empty graph, which terminates the procedure. In addition, we keep track of the number of physical qubits associated with each matching. This count is referred to as the weight $w_i^1$ of the matching $\mathcal{E}_i^1$. The peeling procedure is detailed in Algorithm 4. The functions `addGhost` and `buildMatch` are detailed in Algorithm 5 and Algorithm 6, respectively. An instance of peeling phase procedure, for the error pattern of Fig 4.5b, is described in the following example.

**Example 4.1.** In Fig 4.6 we report an example of peeling phase procedure. The arrows indicate the currently processed defect, while switched-off defects are represented as white circles. Vertices are processed sequentially from left to right and top to bottom during the operation. If a ghost ancilla is present, it will always be processed first. Edges included in the solution are highlighted in green. a) Radius evaluation phase. b) Bubble clustering phase and addition of a ghost ancilla where necessary. c) A boundary vertex for the orange cluster is identified. Since this ver-

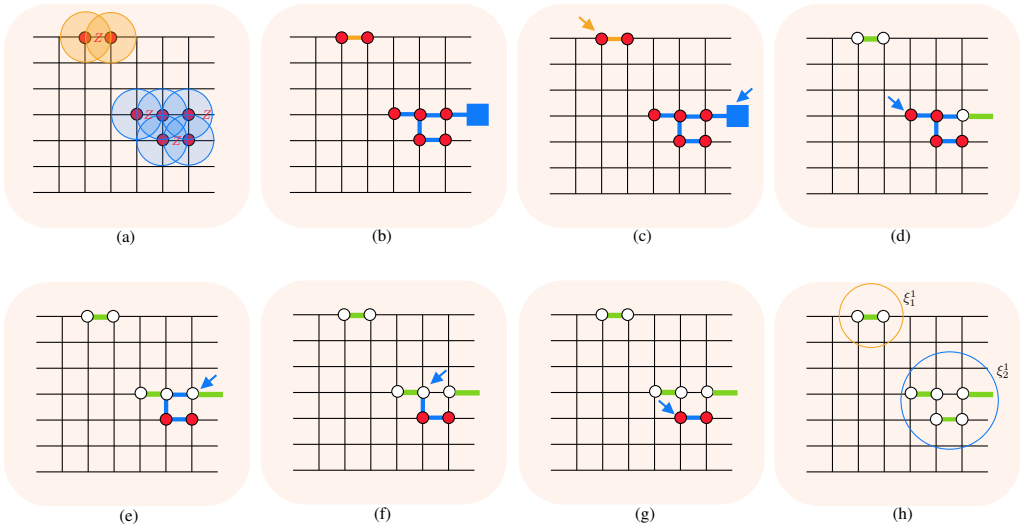


Figure 4.6: A detailed step-by-step example of the peeling phase procedure on a $[[85, 1, 7]]$ surface code with four errors present in the lattice.

tex is currently switched on, it will be switched off along with its adjacent defect. The incident edge will also be added to the solution. The edge connected to the ghost ancilla is added into the blue solution, and the adjacent vertex is switched off. d) A boundary vertex for the blue cluster is identified. Since this vertex is currently switched on, it will be switched off along with its adjacent defect. The incident edge will also be added to the solution. e, f) A boundary vertex is identified. Since this vertex is switched off, the incident edge will be discarded. g) A boundary vertex for the blue cluster is identified. Since this vertex is currently switched on, it will be switched off along with its adjacent defect. The incident edge will also be added to the solution. h) Resulting matching for both clusters.

### 4.4.4 Post-Processing Phase

The goal of this phase is to identify a suitable error correction pattern for each cluster, utilizing the matching solutions obtained from previous stages. For each cluster, if $w_i^1 \leqslant t$, the corresponding matching is selected as the error pattern and this phase is skipped. On the other hand, if the $i$-th cluster does not satisfy this condition, a post-processing is required (i.e., an additional peeling phase). The main idea is to construct a new solution differing from the previous one by a

logical operator. In this way, we can compare the two solutions using our tailored metric.

Then, the tree peeling is performed on this new graph to obtain the matching $\mathcal{E}_i^2$. The error pattern is chosen as follows:

- if $w_i^2 \leqslant t$, $\mathcal{E}_i^2$ is chosen as the current solution;
- else if $w_i^1 = t + 1$, $\mathcal{E}_i^1$ is chosen as the current solution;
- else if $w_i^2 = t + 1$, $\mathcal{E}_i^2$ is chosen as the current solution.

At this point it is very likely that the solution has been already determined. However, if both $w_i^1$ and $w_i^2$ have weight $> t + 1$, we apply the decision criteria presented in 4.2.4. Let us define a *column* as the set of horizontal edges aligned vertically on the lattice, and let us enumerate columns from left to right, ranging from 1 to $d$. Then, considering cluster $i$-th, we define two vectors $\boldsymbol{c}_i^j$ with entries $c_i^{j,k}$, where $j = 1, 2$ and $k = 1, \ldots, d$, representing the cardinality of the intersection between $\mathcal{E}_i^j$ and the $k$-th column. Also, we define as $\boldsymbol{u}_i^j$ a vector with entries $u_i^{j,k} = c_i^{j,k} \mod 2$. Hence, the final error pattern consists in the matching that minimize

$$f(\mathcal{E}_i^j) = \sum_{k=1}^{d} u_i^{j,k} \,. \tag{4.3}$$

### 4.4.5 Adjustments for High Physical Error Rates

Previously, we focused our attention on guaranteeing the error correction capability $t$ and obtaining a good performance in the low physical error region given by error patterns of weight $t + 1$. Here, we present some simple adjustments for the bubble clustering phase which mainly target error patterns of weight greater than $t + 1$, affecting the performance curve in the high physical error regime.

*Setting the minimum:* The BC decoder, as described in previous sections, achieves asymptotic performance that is very close to that of the standard MWPM decoder. However, some simple precautions are necessary when dealing with high physical error rates, specifically when the number of defects exceeds $2t + 2$. In particular, according to (4.2), the maximum radius of a cluster would be

zero, making it impossible to connect any pair of defects and always relying on boundary connections. Hence, to avoid failures in the decoding process caused by this issue, we adjust the bubble radius as

$$R_{\text{sph}} = \begin{cases} t + 2 - \left\lceil \frac{n_{\text{d}}}{2} \right\rceil & \text{if } n_{\text{d}} \leqslant 2t, \\ 2 & \text{otherwise.} \end{cases} \tag{4.4}$$

Note that, using (4.4), not only the radius cannot be zero, but also it cannot be one. This has been done to guarantee that error patterns with a double error occurring on two adjacent qubits in the actual lattice reside always in the same cluster if all other errors are not interfering with their defects. Beside this reasoning, we also fine-tune this adjustment testing other possibilities, such as constant radius one and non-constant shapes. The solution in (4.4) was the one having the best performance for error patterns of weight greater than $t+1$. In Fig. 4.8 we show the impact of this adjustment on a $[[85, 1, 7]]$ surface code. We observe that, for high physical error rates, applying these radius modifications provides an advantage in terms of the logical error rate, as demonstrated by comparing the black and yellow curves. This improvement arises from the enhanced ability to correct errors of weight $\geqslant t + 1$.

*Star-defects avoidance:* Recall that the bubble clustering stage, described in Section 4.4.2, is performed on the defect graph. This implies that there is no direct correspondence between the edges in the resulting clusters and the physical qubits in the lattice. Hence, since the defects are processed sequentially, due to $R_{\text{sph}}$ it is possible that several defects are connected to the defect under processing even if a different defect was the nearest neighbour, creating a *star*-like graph. These star defects do not harm the decoder ability to correct up to the code distance, however, they worsen the performance in the high physical error regime (capability to correct error pattern with more than $t$ errors). For instance, let us consider the bubble clustering stage in Fig. 4.7a. Since the error causes five defects, $R_{\text{sph}} = 2$. Then, according to the standard procedure and proceeding from the lower vertex index to the higher one, the defect $v_2$ results in a star defect, being adjacent to defects $v_1$, $v_3$ and $v_4$. In this way, the resulting tree is the one having as edges: $(v_1, v_2)$, $(v_2, v_3)$, $(v_2, v_4)$, and $(v_3, v_5)$. Indeed, during the peeling phase, this tree results in

a matching consisting of: a weight-two horizontal edge between the ghost ancilla and $v_1$; a weight-two edge between $v_2$ and $v_4$; and a weight-two edge from $v_3$ to $v_5$. In this particular configuration, the problem arises if the `buildMatch`($v_2$,$v_4$) does not share a qubit with `buildMatch`($v_3$,$v_5$) (e.g., the qubit between $v_3$ and $v_4$). In fact, if this qubit is not shared, the weight results in $w_1^1(\mathcal{E}_1^1) = 6$. Moreover, the alternative solution, obtained by attaching a ghost ancilla to $v_5$ instead of $v_1$ would have weight $w = 6$. In this case, both solutions are greater than $t + 1$. Furthermore, according to metric (4.3), the incorrect solution would be selected, causing an error in the correction process.

To avoid this, we introduce an improvement of the bubble clustering. Specifically, each time a defect is added to a cluster, we record the distance to its adjacent defect within the tree. Then, if the defect being processed is closer to another defect already in the tree, and both defects are adjacent to the same star defect, the second defect is detached from the star defect and attached to the defect being processed. This modification requires additional processing, which may increase the execution time. However, it should be noted that, in the presence of star defects, the information about the same physical qubits must be retrieved multiple times from memory during the peeling phase. Therefore, avoiding star defects helps recover execution time in this regard. Referring back to the previous example, with the proposed modification, when processing defect $v_3$, it becomes possible to detach $v_4$ from $v_2$, and attach $v_3$ to $v_4$. Moreover, when processing $v_5$ it is also possible to detach it from $v_3$ and attach it to $v_4$, obtaining the tree shown in Fig. 4.7b. In this way, the first matching has weight $w_1^1(\mathcal{E}_1^1) = 4$, and will be chosen as a final solution, correcting the error. Let us clarify this procedure with an example.

**Example 4.2.** In Fig. 4.7, we have four $\boldsymbol{Z}$ channel errors occurred on the lattice, resulting in defects depicted as red circles. Since $n_{\mathrm{d}} = 5$, the bubble radius $R_{\mathrm{sph}} = 2$, resulting in a single cluster with an odd number of defects, necessitating the insertion of a single ghost ancilla. a) Bubble clustering decoding without star-defects avoidance. The matching $\mathcal{E}_1^1$, evaluated during the peeling phase, has weight $w_1^1 = 6 > t$. Therefore, a post-processing phase is required. The second matching has weight $w_1^2 = 6 > t$. Both solutions have weight $w_1 > t + 1$. Therefore, the solution is selected based on (4.3), which results in choosing the

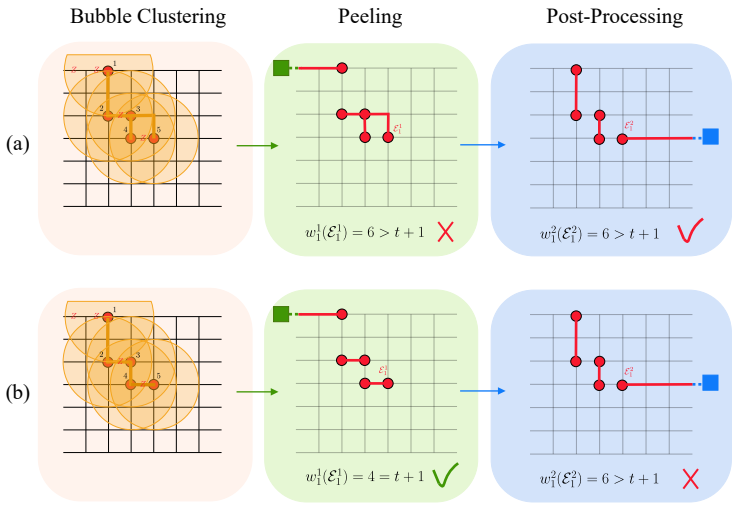


Figure 4.7: Examples of BC decoding for a $[[85, 1, 7]]$ surface code: a) without star-defects avoidance. b) with star-defects avoidance.

faulty error pattern. b) Bubble clustering decoding with star-defects avoidance. The matching $\mathcal{E}_1^1$, evaluated during the peeling phase, has weight $w_1^1 = 4 > t$. Therefore, a post-processing phase is required. The second matching has weight $w_1^2 = 6 > t$. Moreover, the weight of the first matching is equal to $t + 1$. Hence, $\mathcal{E}_1^2$ is chosen as final matching, and the error is actually corrected.

In Fig. 4.8 we show the impact of this adjustment on a $[[41, 1, 5]]$ surface code. Without star-defect avoidance, the logical error rate is represented by the yellow curve. However, by applying these techniques, we achieve the green curve, indicating a significant improvement. This gain shows that avoiding star defects enhances the correction of errors with weight $\geqslant t+1$, leading to improved overall performance. For completeness, we also include the case where the star-defect avoidance technique is employed without radius adjustments, shown by the red curve. The radius adjustment technique yields performance improvements at high physical error rates, whereas the star-defect avoidance technique provides an advantage at low physical error rates. Adopting the described techniques, we observe an overall improvement of the performance that increases with the code distance. As an example, in Fig. 4.8, for $p = 10^{-2}$, we observe an improvement of $90\%$ in error correction. For $p = 10^{-2}$, the $[[41, 1, 5]]$ surface code yields an improvement of $42\%$, while the $[[145, 1, 9]]$ surface code achieves a higher im-

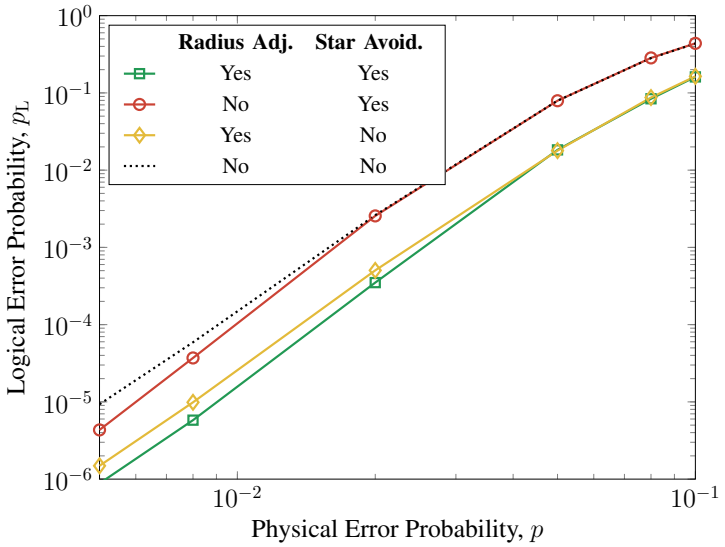


Figure 4.8: Logical error probability vs. physical error probability of the $[[85, 1, 7]]$ surface code. The curves illustrate the impact of radius adjustments and star-defect avoidance techniques on the error correction capability.

provement of $78\%$.

*High code distance:* As the number of qubits scales with the square of the code distance, in this error regime it is beneficial to implement additional low-complexity measures to refine the error correction capability. Specifically, for surface codes with a code distance of $d \geqslant 11$, we will adopt the following measures. During the sphere clustering phase, we record clusters that contain a single defect. If exactly two such clusters are found, and their defects are separated by a distance of $R'_{\text{sph}} + 1$, we merge them into a single cluster containing both defects. If a cluster contains a single defect, and its distance from the boundary equals the distance to a defect in an odd-cardinality cluster, we connect these defects in a single cluster.

### 4.4.6 Error Correction Capability Preservation

Before proceeding with the last technical details of the decoder, we demonstrate that the whole processing using the BC decoder ensures error correction capability up to the code distance.

**Theorem 4.2.** *For an $[[n, k, d]]$ surface code, an error pattern with weight $w \leqslant t$ is always correctly corrected using the BC decoder.*

*Proof.* Note that in case both $\mathcal{E}_i^1$ and $\mathcal{E}_i^2$ have weight $> t + 1$, (4.1) is employed, and the error is always corrected as proven in Section 4.2.4 . Regarding errors of weight $\leqslant t$, for which at least one matching between $\mathcal{E}_i^1$ and $\mathcal{E}_i^2$ has weight $\leqslant t + 1$, we want to show that these are always corrected. To this aim, we remark that one of the two possible solutions, $\mathcal{E}_i^1$ and $\mathcal{E}_i^2$, always corrects the error, while the other always realizes a logical operator. Indeed, the two solutions are related by the application of a logical operator since the surface codes encode $k = 1$ information qubit. Let us initially focus on a surface code with odd distance. As a worst case, we first consider $t$ Pauli errors in the same row of the lattice, possibly causing an horizontal logical operator of weight $2t+1$. In case of an odd number of defects, one of the two solutions $\mathcal{E}_i^1$ or $\mathcal{E}_i^2$ has weight $w = t$ due to the horizontal disposition, also corresponding to the actual error. The alternative solution has always weight $2t + 1 - t = t + 1$. Moreover, if an even number of defects is obtained, the first solution (with no ghost ancillas) always coincides with the error and has weight $t$. Hence, in both cases, the error is always corrected. Next, let us consider the case in which $t$ errors occur on two adjacent rows, possibly causing a logical operator of weight $2t + 2$. In this configuration, it is possible for the correct solution not to be of minimum weight, i.e., its weight can be greater than $t$. This happens when the final matching results in the correct solution plus an element of the stabilizer. However, given that the possible logical operator has a weight of $2t + 2$ and the actual error has a weight of $t$, the weight of the incorrect solution will be at least $2t + 2 - t = t + 2 > t + 1$. Hence, in the worst case, both solutions have weight greater than $t + 1$ and, using (4.1), the error is corrected. Moreover, if $t$ errors occur across multiple rows, the weight of the resulting logical operator would exceed $2t + 2$, further reinforcing the validity of the previous argument. The same arguments apply to a surface code with even distance, where the corresponding logical operators have a weight greater by one, i.e., the smallest logical operator has weight $2t + 2$. □

This theorem has important consequences on the decoding performance. Indeed, there is a great amount of errors of weight $t + 1$ that are not corrected by

(4.1) but can result in a solution of weight $t+1$. Hence, they are actually corrected by the BC decoder, according to Section 4.4.4. An instance of the BC decoder requiring a post-processing phase is depicted in Fig. 4.7 for the $[[85, 1, 7]]$ surface code.

**Corollary 3.** *Any error pattern of weight $t+\ell-1$, for which the bubble clustering procedure results in at least $\ell > 0$ distinct clusters, is always corrected.*

### 4.4.7 Complexity Analysis

In this section we analyze the complexity of the BC decoder. The first phase, i.e., the evaluation of the bubble radius, is performed with a complexity of $O(1)$. Next, for each defect the algorithm potentially compares it to every other defect to check the distance condition. This check is performed during the bubble clustering phase. Hence, the worst-case scenario involves $O(n_{\mathrm{d}}^2)$ operations. Also, before adding a defect to a cluster, the algorithm checks if it is already in the cluster. Each check is constant time $O(1)$ because it simply involves looking up the defect's membership in an array. Thus, the total complexity is dominated by the quadratic terms, resulting in an overall complexity of $O(n_{\mathrm{d}}^2)$. The star-defects avoidance technique involves checking whether two defects share any adjacent defects, and this check is performed during the bubble clustering phase whenever two defects are assigned to the same cluster. This verification can be performed efficiently by accessing a pre-stored element in an array that was created during the cluster construction specifically for this purpose. As a result, this procedure can be completed in constant time, with a complexity of $O(1)$. The tree peeling stage processes each cluster to iteratively remove defects and edges until a suitable matching is achieved. Since each defect in the lattice is part of exactly one cluster, and each operation (inverting parity, removing edges) is proportional to the number of defects, the overall complexity is proportional to the total number of defects $O(n_{\mathrm{d}})$. Moreover, in the post-processing phase, the weight of each cluster matching is compared to the $t$ parameter. This comparison is a simple $O(1)$ operation per cluster. If there are $N_{\mathrm{c}}$ clusters and recalling that $N_{\mathrm{c}} \leq n_{\mathrm{d}}$ the complexity becomes $O(n_{\mathrm{d}})$. Finally, the vector $\boldsymbol{u}_i^j$ can be computed simply by enumerating the horizontal qubits in the lattice row by row, from left to right. It

Table 4.1: Decoder Complexities.

| | MWPM [48] | UF [119] | BC | STM [68] | RFire [68] |
|---|---|---|---|---|---|
| complexity | $O(n_{\mathrm{d}}^3 \log(n_{\mathrm{d}}))$ | $O(n\alpha(n))$ | $O(n_{\mathrm{d}}^2)$ | $O(n_{\mathrm{d}}^2 \log(n_{\mathrm{d}}))$ | $O(n_{\mathrm{d}}^2)$ |

is sufficient to perform a single mod operation (which can be performed in $O(1)$) for each horizontal qubit in the solution. Thus, the worst-case complexity of this step is $O(w)$, where $w$ is the weight of the error. Note, however, that this step is required only in a small number of cases. Overall, the BC decoder complexity is primarily determined by the bubble clustering stage. Note that for an $[[n, k, d]]$ surface code, in the case of an error of weight $\leqslant t + 1$, the number of defects is $n_{\mathrm{d}} \leqslant 2t + 2$. Hence, the complexity of the algorithm in this regime can be written as $O(d^2)$ or $O(n)$, considering that in a surface code $n = d^2 + (d-1)^2$. Table 4.1 presents a comparison of the complexities of different decoders. Here, $n$ represents the number of qubits, $n_{\mathrm{d}}$ denotes the number of defects, and $\alpha$ is the inverse of Ackermann's function [119].

## 4.5 Numerical Results

In this section we compare the performance of surface codes using BC, MWPM, UF, and RFire decoding via Monte Carlo simulations. We included RFire in the comparison because it offers faster execution times than STM while providing similar error correction capability. All these decoders are implemented in C++, run with an Apple Silicon M2 processor and executed on a single core. The UF algorithm is a C++ implementation of the efficient weighted union find described in [119]. In the standard implementation, we exploit the LEMON C++ library for an efficient MWPM algorithm [100]. Also, we employ the PyMatching 2 library for the sparse blossom implementation [52]. Additionally, when using PyMatching, we submit shots in batches of 1000 to minimize the overhead of Python-to-C++ method calls. Also, for an efficient implementation of LEMON version of the MWPM, and RFire decoders, we use Manhattan distance to assign weights and construct the graph of defects, avoiding the usage of Dijkstra's algorithm. Regarding the BC decoder, the numerical evaluation is performed by including all

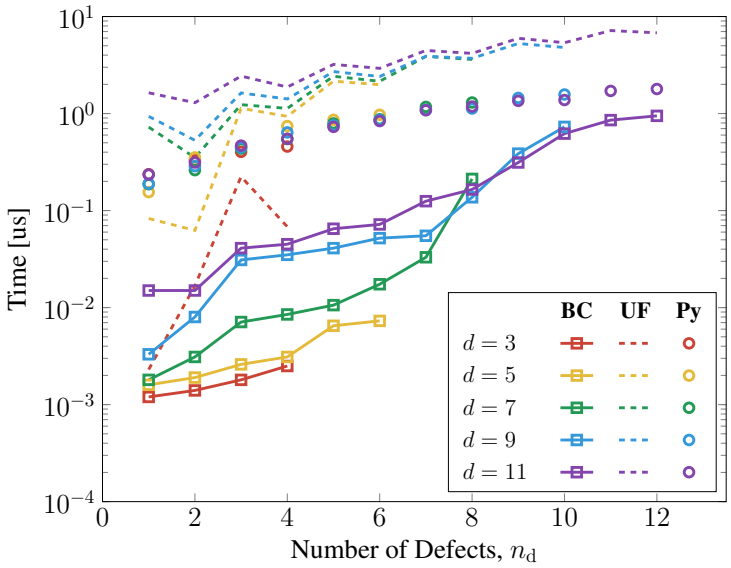


Figure 4.9: Average execution times per single decoding vs. number of defects. The abbreviation Py stands for PyMatching [52] version of the MWPM.

adjustments discussed in Sec. 4.4.5. For each simulation point in the numerical analysis, 100 error realizations were tested to ensure sufficient statistical accuracy.

*Average execution times:* To assess the complexity of the decoders, we measure the average execution time of the complete decoding procedure: from the syndrome measurement to the final solution. Fig. 4.9 and Fig. 4.10 show the average execution times per single decoding over the number of defects in the lattice. The evaluation is carried out when varying the number of defects and the lattice size, from the $[[13, 1, 3]]$ to the $[[221, 1, 11]]$ surface codes. Specifically, the decoders are provided with batches of 1000 instances, each containing $n_{\mathrm{d}}$ defects. From Fig. 4.9, it is evident that the BC decoder outperforms both PyMatching and UF algorithms in terms of execution speed across all code instances. From Fig. 4.10, we observe that the BC decoder achieves a time savings of over an order of magnitude compared to the RFire decoder for code distances greater than three. Moreover, Fig. 4.11 presents a comparison of the average execution times of the BC, UF, and Pymatching algorithms for high code distances, i.e., surface codes up to the $[[685, 1, 19]]$. The results indicate that the performance advantage of the BC decoder remains consistent as the code distance increases.

*Logical error rate:* In Fig. 4.12 and Fig. 4.13, we show the logical error rate

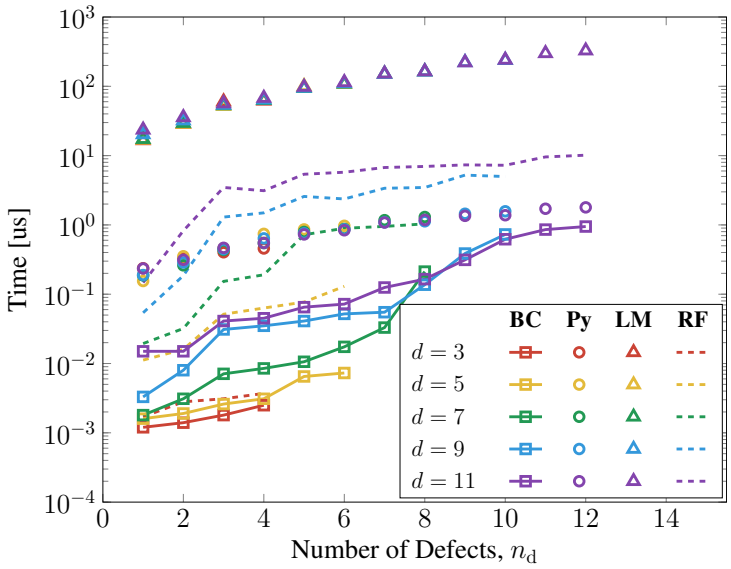


Figure 4.10: Average execution times per single decoding vs. number of defects. The abbreviation Py stands for PyMatching [52] version of the MWPM.

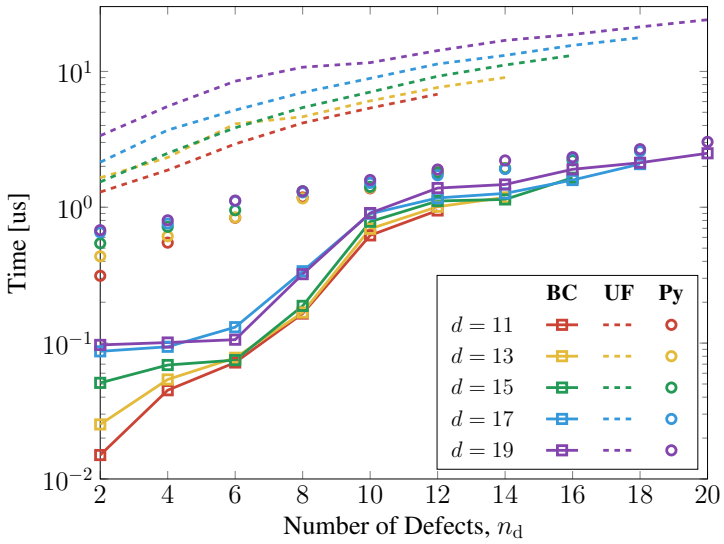


Figure 4.11: Average execution times per single decoding vs. number of defects. The abbreviation Py stands for PyMatching [52] version of the MWPM.

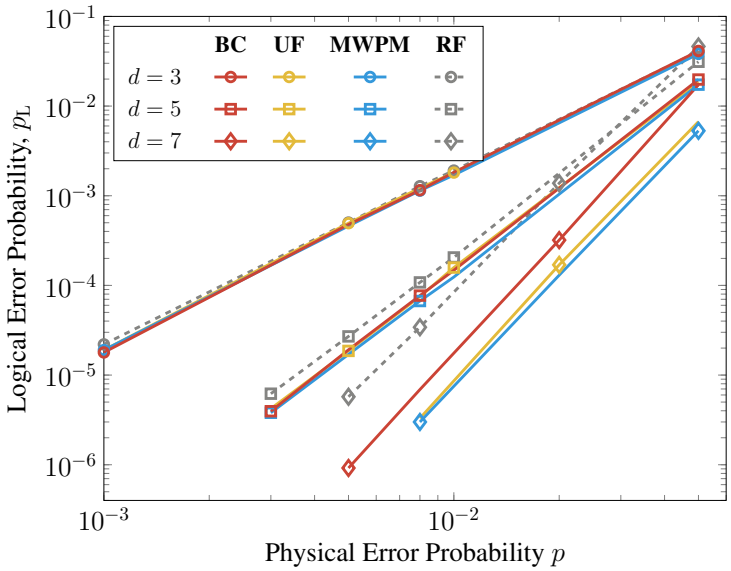


Figure 4.12: Logical error probability vs. physical error probability over depolarizing channel.

as a function of the physical error rate of some surface codes over depolarizing channel. We can observe that, employing the clusterization procedure, we have a large gain in error correction capability for the BC decoder when compared to the RFire. From these results we observe that, for shorter code distances, the performance of the BC and the MWPM decoders are quite comparable. In Fig. 4.13 we observe that the performance gap between the MWPM and BC decoders widens when comparing distances $d = 7$ and $d = 9$, but shows only a slight increase between $d = 9$ and $d = 11$.

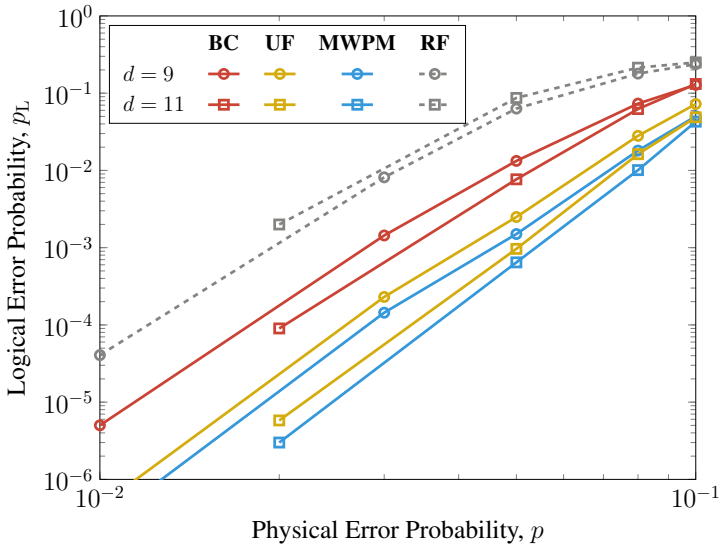


Figure 4.13: Logical error probability vs. physical error probability over depolarizing channel.

# Chapter 5

# Flag at Origin. A Modular Fault-Tolerant Preparation for CSS Codes

Stabilizer states play a central role in universal FT quantum computation [72, 131]. Logical states such as $\left|\overline{0}\right\rangle$ and $|\overline{+}\rangle$ frequently serve as initial qubits or ancillary resources in quantum algorithms [132]. In case of QLDPC codes with multiple encoded qubits [133], a broad set of stabilizer states must be prepared fault-tolerantly to support teleportation-based protocols for the universal Clifford group. [134–136]. Moreover, FT error correction schemes such as those by Steane [137, 138] or Knill [139] rely on additional logical stabilizer states to extract the error syndrome. Beyond these roles, stabilizer states serve other purposes as well, including the mediation of interactions between distant logical qubit blocks in architectures with restricted connectivity, thereby enabling long-range logical operations [34, 140].

The overarching goal is therefore to devise FT, resource-efficient initialization protocols for QEC codes across arbitrary code distances $d$. Specifically, a FT circuit guarantees that up to $t = \lfloor d/2 \rfloor$ faults lead to logical errors only with probability scaling as $\mathcal{O}(p^{t+1})$ [141, 142], where $p$ denotes the physical error rate. In the case of CSS codes, the design of FT circuits is simplified, since $\boldsymbol{X}$ and $\boldsymbol{Z}$ stabilizers can be treated independently. A common strategy first employs a non-

FT Clifford circuit to prepare the target CSS state, followed by a verification stage that ensures fault-tolerance. The non-FT component can be engineered via partial Latin rectangle completions [143, 144] or Clifford synthesis methods [145–147]. Verification then relies on redundant ancilla states together with either deterministic [148, 149] or probabilistic [150–152] filtering techniques. Although effective, these approaches often incur substantial resource overheads, though for certain codes optimizations are possible [153]. Recent work has introduced heuristic and automated approaches to identify compact FT circuits, including direct inspection for small codes [154, 155], reinforcement learning [156], and SAT-based solvers [157]. Nevertheless, scaling such methods to large code distances remains computationally demanding.

In this chapter we propose a modular and scalable framework for FT initialization of CSS codes that significantly reduces resource overhead, especially for the largest codes considered. We show that CSS states can be prepared using bipartite circuits composed of $\boldsymbol{CX}$ and Hadamard gates, with the bipartition naturally determined by the control and target qubits of the $\boldsymbol{CX}$ gates. The proposed construction exploits the asymmetric propagation of Pauli errors inherent to such circuits: Pauli-$\boldsymbol{X}$ errors propagate exclusively from control to target qubits, while Pauli-$\boldsymbol{Z}$ errors propagate in the opposite direction, from target to control qubits. By appending $\boldsymbol{X}$- and $\boldsymbol{Z}$-detecting flag gadgets to the corresponding qubit partitions, the scheme achieves fault-tolerance in a structured and resource-efficient manner.

## 5.1 Bipartite Preparation Circuit for CSS Codes

A *CSS state* is a stabilizer state whose stabilizer generators consist exclusively of tensor products of either Pauli $\boldsymbol{X}$ operators or Pauli $\boldsymbol{Z}$ operators. Any CSS code, prepared in a logical state stabilized by logical operators of this form, yields a CSS state. For instance, CSS states include $\left|\overline{0}\right\rangle$ or $|\overline{+}\rangle$ in CSS codes with $k = 1$, logical GHZ states of the form $(\left|\overline{0\cdots0}\right\rangle + |\overline{+\cdots+}\rangle)/\sqrt{2}$ in CSS codes with $k > 1$, or any tensor product of such states. In this section, we prove that CSS states can be prepared using a bipartite circuit composed only of $\boldsymbol{CX}$ gates and qubit initializations in the $|0\rangle$ or $|+\rangle$ states. The proof follows the approach in [158], tailored to

the specific case of CSS states. Specifically, a CSS state can be represented by the $2n \times n$ binary matrix

$$\boldsymbol{S} = \left[\begin{array}{c|c} \overbrace{0}^{r} & \overbrace{\boldsymbol{B}}^{n-r} \\ \hline \boldsymbol{A} & 0 \end{array}\right]$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times r$ and $n \times (n-r)$ matrices representing the $\boldsymbol{X}$-type and $\boldsymbol{Z}$-type generators, respectively. Since no product of generators yields the identity, $\boldsymbol{A}$ and $\boldsymbol{B}$ must be full-rank matrices with ranks $r$ and $n-r$, respectively. Hence, by applying gaussian elimination, the stabilizer representation can be brought into the form

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{Z}_1 \\ \boldsymbol{Z}_2 \end{bmatrix}$$

such that $\boldsymbol{X}_1$ and $\boldsymbol{Z}_2$ are invertible $r \times r$ and $(n-r) \times (n-r)$ matrices, respectively. Applying Hadamard gates to the last $n-r$ qubits transforms the stabilizer state into the graph state

$$\boldsymbol{G} = \left[\begin{array}{c|c} 0 & \boldsymbol{Z}_1 \\ \boldsymbol{X}_2 & 0 \\ \hline \boldsymbol{X}_1 & 0 \\ 0 & \boldsymbol{Z}_2 \end{array}\right]. \tag{5.1}$$

Recombining the columns from the right with the invertible matrix $\boldsymbol{R}$ yields identity matrices in the lower block of the graph state representation:

$$\boldsymbol{GR} = \left[\begin{array}{c|c} 0 & \boldsymbol{Z}_1\boldsymbol{Z}_2^{-1} \\ \boldsymbol{X}_2\boldsymbol{X}_1^{-1} & 0 \\ \hline \boldsymbol{I} & 0 \\ 0 & \boldsymbol{I} \end{array}\right], \qquad \boldsymbol{R} = \left[\begin{array}{cc} \boldsymbol{X}_1^{-1} & 0 \\ \hline 0 & \boldsymbol{Z}_2^{-1} \end{array}\right]. \tag{5.2}$$

This step consists in a change of the generator basis. From the commutation condition $\boldsymbol{Z}^T\boldsymbol{X} = 0$, we find that the top block of the graph state matrix is symmetric

with zeros on the diagonal, satisfying

$$\boldsymbol{Z}_1\boldsymbol{Z}_2^{-1} = \left(\boldsymbol{X}_2\boldsymbol{X}_1^{-1}\right)^T.$$

The matrix $\boldsymbol{GR}$ represents the adjacency matrix of a bipartite graph in which the control qubits (first $r$ columns) are connected only to the target qubits (last $n-r$ columns).

This shows that the generator matrix of CSS stabilizer states can be brought into the form of a bipartite graph state, up to Hadamard operators on one partition. Graph states can be prepared by initializing all $n$ qubits in $|+\rangle$ and applying a $\boldsymbol{CZ}$ gate between every pair of qubits that are connected in the underlying graph. When Hadamard gates are applied to one partition of the qubits, all $\boldsymbol{CZ}$ gates are transformed into $\boldsymbol{CX}$ gates, and the target qubits become initialized in the $|0\rangle$ state instead. This yields a bipartite $\boldsymbol{CX}$ circuit, in which all $\boldsymbol{CX}$ gates act from control qubits belonging to one partition (initialized in $|+\rangle$) onto target qubits belonging exclusively to the complementary partition (initialized in $|0\rangle$).

For any input CSS state, the Python library StabGraph [159] can generate bipartite $\boldsymbol{CX}$ circuits with a reduced $\boldsymbol{CX}$ count.

## 5.2 Fault-Tolerant Circuit Construction

In realistic quantum hardware, imperfections cause circuit components to fail with some probability, thereby introducing errors into the computation. Such faults can be modeled by replacing the faulty component with its ideal counterpart, followed by the application of a random Pauli error. Specifically, after a faulty qubit preparation, single-qubit gate, or idling period, a random Pauli operator from the set $\{\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}$ is applied prior to measurement. For a faulty two-qubit gate, a random Pauli operator from $\{\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}^{\otimes 2} \setminus \{\boldsymbol{I} \otimes \boldsymbol{I}\}$ is applied. These errors may propagate through the circuit, potentially leading to high-weight correlated errors. Controlling such error propagation is therefore essential to ensure fault-tolerance.

**Definition 5.1.** A preparation circuit for a CSS state is FT [142] if, for any number $f \leqslant t = \lfloor d/2 \rfloor$ of faulty components, the resulting propagated error is either of minimum weight $w \leqslant f$ and therefore correctable by an ideal decoder, or is

detected by the ancillas and flags in the circuit, causing the preparation attempt to be restarted. The minimum weight of a Pauli error is the minimum support of the error under the multiplication by all stabilizer operators of the CSS state.

As shown in Section 5.1, every CSS state can be prepared using a bipartite $\boldsymbol{CX}$ circuit, such as the one in Fig. 5.1a, where each qubit serves as either a control or a target for $\boldsymbol{CX}$ gates. Notably, in bipartite $\boldsymbol{CX}$ circuits, $\boldsymbol{Z}$ errors on control qubits and $\boldsymbol{X}$ errors on target qubits do not propagate, making the construction inherently compliant with the fault-tolerance criterion for state preparation. The only propagating errors that must be detected are $\boldsymbol{X}$ errors on control qubits and $\boldsymbol{Z}$ errors on target qubits. These errors are caught at their origin by appending $\boldsymbol{X}$-detecting flag gadgets to each control qubit, and $\boldsymbol{Z}$-detecting flag gadgets to each target qubit, as shown in Fig. 5.1b and in Fig. 5.1c, respectively. Whenever a flag gadget detects an error, the preparation attempt is restarted. The resulting circuit, shown in Fig. 5.1d, is FT. The two types of flag gadgets must be carefully coordinated to preserve the internal gate ordering within each gadget, as this ordering is essential to ensuring their fault-tolerance. Importantly, this construction is general for CSS codes and, thanks to its simplicity and modular design, can be scaled to large code distances. An example of an $\boldsymbol{X}$-detecting flag gadget for five $\boldsymbol{CX}$ gates and code distances $d = 5$ is given in Fig. 5.2j.

Since the number of gates in a bipartite circuit scales as $O(n^2)$ in the worst case, the overall construction requires only a polynomial amount of resources. In practice, we observe that the flag gadgets require relatively few additional $\boldsymbol{CX}$ gates and flag qubits, with their size growing only linearly in both the code distance and the number of gates acting on each code qubit. Table 5.1 provides a comparison of the $\boldsymbol{CX}$ gate count achieved by our construction against state-of-the-art optimized implementations for specific codes (when available), or against baseline methods. These consist of initializing all code qubits in the $|0\rangle$ state and fault-tolerantly measuring all $\boldsymbol{X}$ stabilizer generators [132]. The measurement is repeated $\lfloor d/2 \rfloor + 1$ times, with the process restarted upon detection of any error. This approach is modular and applicable to any CSS code at any distance, making it a suitable benchmark for comparison.

For most CSS codes, achieving FT measurements of the stabilizer generators requires either attaching flag gadgets to the ancillary qubits used for syndrome

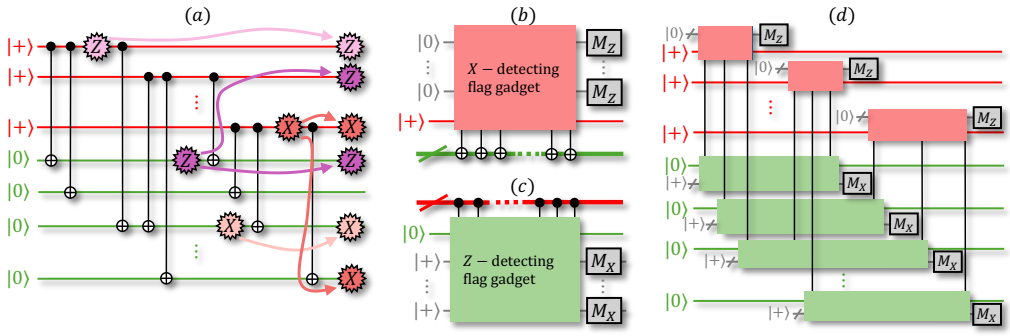


Figure 5.1: Flag at origin construction of a CSS state. $(a)$ Such a state can be prepared non-fault-tolerantly by a bipartite $\boldsymbol{CX}$ circuit, where $\boldsymbol{CX}$ gates control only control qubits initialized in $|+\rangle$ and target only target qubits initialized in $|0\rangle$. $\boldsymbol{X}$ errors propagate only from control to target qubits, while $\boldsymbol{Z}$ errors propagate only from target to control qubits. $(b)$ An $\boldsymbol{X}$-detecting flag gadget appended to a control qubit can detect any $\boldsymbol{X}$ error occurring on it before it propagates. $(c)$ A $\boldsymbol{Z}$-detecting flag gadget appended to a target qubit can detect any $\boldsymbol{Z}$ error occurring on it before it propagates. $(d)$ Appending $\boldsymbol{X}$-detecting flag gadgets to all control qubits and $\boldsymbol{Z}$-detecting flag gadgets to all target qubits results in a FT preparation circuit.

extraction [96] or employing cat states [21]. Designing flag gadgets that maintain the fault-tolerance of the protocol remains a nontrivial task. While several general constructive methods exist, they often produce gadgets that demand a large number of flag qubits and additional gates [160, 161]. However, for surface codes, whose stabilizer generators are of low weight, FT measurements can be achieved through carefully designed gate schedules [27, 162]. In the present analysis, for non-surface codes, the number of $\boldsymbol{CX}$ gates in the baseline techniques is calculated by considering a FT flag-based stabilizers measurement, where the flag gadgets are constructed as described in Section 5.3. Indeed, since these gadgets are designed to detect any combination of $f \leqslant t$ faults that propagate to an error of weight $w > f$ in a GHZ state preparation circuit, they are well suited for FT syndrome extraction based on post-selection.

For all medium- to large-scale codes, the proposed CSS state preparation outperforms both state-of-the-art circuits and baseline constructions. Even for individually optimized QEC codes, such as the $[[31, 1, 7]]$ color code and the $[[23, 1, 7]]$ Golay code, our construction produces more efficient circuits, notably with a reduced number of $\boldsymbol{CX}$ gates. In addition, the table reports the maximum number of

| QEC code and logical state prepared | Baseline CXs | CXs | Sim. Qubits | Flags | Depth | Log. Error Rate | Acceptance Rate |
|---|---|---|---|---|---|---|---|
| $[[7,1,3]]$ Steane $\|\overline{0}\rangle$ | **11** [154] | 15 | 8 | 3 | 10 | $[2.7,\ 2.9]\times10^{-5}$ | $[0.9783,\ 0.9784]$ |
| $[[9,1,3]]$ rot. surface $\|\overline{0}\rangle$ | **8** [155] | 26 | 12 | 9 | 9 | $[2.4,\ 2.6]\times10^{-5}$ | $[0.97150,\ 0.97158]$ |
| $[[17,1,5]]$ color code $\|\overline{0}\rangle$ | **71** [157] | 74 | 23 | 21 | 25 | $[7.7,\ 18.2]\times10^{-7}$ | $[0.8945,\ 0.8948]$ |
| $[[25,1,5]]$ rot. surface $\|\overline{0}\rangle$ | 120 [163] | **92** | 32 | 28 | 23 | $[6.7,\ 24.2]\times10^{-7}$ | $[0.8980,\ 0.8984]$ |
| $[[49,1,5]]$ triorthogonal $\|\overline{+}\rangle$ | 936 | **361** | 95 | 105 | 59 | $[4.3,\ 5.4]\times10^{-5}$ | $[0.585,\ 0.584]$ |
| $[[20,2,6]]$ self-dual $\|\overline{00}\rangle$ | 376 | **145** | 36 | 47 | 54 | $[2.3,\ 9.7]\times10^{-8}$ | $[0.8234,\ 0.8235]$ |
| $[[23,1,7]]$ Golay $\|\overline{0}\rangle$ | 297 [153] | **237** | 44 | 80 | 33 | $[1.8,\ 3.1]\times10^{-7}$ | $[0.7095,\ 0.7099]$ |
| $[[31,1,7]]$ color code $\|\overline{0}\rangle$ | 421 [157] | **211** | 55 | 69 | 58 | $[8.0,\ 21.8]\times10^{-7}$ | $[0.750,\ 0.751]$ |
| $[[49,1,7]]$ rot. surface $\|\overline{0}\rangle$ | 336 [163] | **262** | 64 | 85 | 46 | $[1.4,\ 2.1]\times10^{-6}$ | $[0.702,\ 0.703]$ |
| $[[95,1,7]]$ triorthogonal $\|\overline{+}\rangle$ | 4792 | **1175** | 258 | 380 | 389 | $[8.5,\ 9.8]\times10^{-5}$ | $[0.240,\ 0.241]$ |
| $[[49,1,9]]$ color code $\|\overline{0}\rangle$ | 1020 | **408** | 93 | 136 | 123 | $[1.5,\ 9.8]\times10^{-7}$ | $[0.531,\ 0.532]$ |
| $[[81,1,9]]$ rot. surface $\|\overline{0}\rangle$ | 720 [163] | **614** | 141 | 206 | 129 | $[3.8,\ 4.1]\times10^{-5}$ | $[0.355,\ 0.356]$ |
| $[[47,1,11]]$ self-dual $\|\overline{0}\rangle$ | 4140 | **1033** | 186 | 388 | 292 | $[2.4,\ 3.2]\times10^{-5}$ | $[0.122,\ 0.123]$ |
| $[[71,1,11]]$ color code $\|\overline{0}\rangle$ | 1860 | **829** | 177 | 268 | 282 | $[2.7,\ 2.8]\times10^{-5}$ | $[0.214,\ 0.215]$ |

Table 5.1: Size and performance of the FT preparation circuits obtained with the flag at origin construction. From left to right, the first two columns indicate: the code and the initial logical state prepared, the number of $\boldsymbol{CX}$ gates required by the best of the baseline and known optimized constructions. Then, for the proposed construction: the maximum number of simultaneous qubits required, the number of flag qubits, the depth in terms of two-qubit gates, the logical error rate, and the acceptance rate in circuit-level simulations (using the noise model described in Section 5.4) at a physical error rate of $10^{-3}$, using Wilson confidence intervals of $95\%$.

qubits used simultaneously during execution (assuming fast qubit reset), the circuit depth, and the logical error and acceptance rates obtained from circuit-level simulations under depolarizing noise (on gates and measurements) and memory noise (on idle qubits).

The proposed construction admits several optimizations to reduce circuit depth, the number of $\boldsymbol{CX}$ gates, and the maximum number of simultaneously active qubits. First, for each CSS code, we exploit the freedom to choose among different bipartite circuit representations, selecting the configuration that minimizes the number of required $\boldsymbol{CX}$ gates by exploring many random instances. Second, a further degree of freedom, applicable to all CSS codes, is the ordering of commuting $\boldsymbol{CX}$ gates in bipartite circuits. Since any ordering remains FT once flag gadgets are appended, randomly shuffling the gates and selecting the optimal order allows minimization of circuit depth or the maximum number of simultaneously active qubits. Finally, the Golay code admits an additional simplification: any $\boldsymbol{Z}$

($\boldsymbol{X}$) error is equivalent, up to multiplication by a $\boldsymbol{Z}$ ($\boldsymbol{X}$) logical operator, to an error of weight at most three. As a result, when preparing the logical $|0\rangle$, the flag gadgets need not detect combinations of three $\boldsymbol{Z}$-type faults, allowing the use of smaller $\boldsymbol{Z}$-detecting flag gadgets designed for distances $d = 4$ and $d = 5$, thereby reducing both the number of flag qubits and the $\boldsymbol{CX}$ count. By employing these techniques, we obtain a preparation circuit for the Golay code that fits within the 56-qubit limit of the Quantinuum H2-1 device, enabling the experiment described in Section 5.5.

## 5.3 Discovery of Flag Gadgets

This section presents the algorithm proposed to discover flag gadgets. The goal of the gadget is to entangle the target qubits to the control qubit with a FT circuit. Hence, any combination of $f \leqslant t$ faults must propagate to a weight $w \leqslant f$ error, up to multiplication with stabilizer operators. Specifically, a non-trivial measurement outcome on any of the flag qubits signals the occurrence of a potentially uncorrectable error. In such cases, the preparation attempt is aborted and restarted.

The algorithm takes as input the number $t$ of correctable faults, the number of target qubits, and an estimate of the number of flag qubits required. The flag gadget is then constructed gate by gate, starting from the end of the circuit. This is necessary because, at each iteration, the fault-tolerance of the gadget must be tested, which is only possible once the end of the circuit is defined. Some of the $\boldsymbol{CX}$ gates in the gadget entangle the target qubits to the control qubit. The rest entangle and disentangle flag qubits to allow the detection of potentially non-correctable fault combinations. All flag qubits must be disentangled before the circuit terminates. We remark that the components comprising the flag gadget are themselves subject to failure. Algorithm 7 presents the pseudo-code for the discovery of flag gadgets.

Fig. 5.2 describes the steps leading to the discovery of an $\boldsymbol{X}$-detecting flag gadget with two flag qubits, $f_1$ and $f_2$, that protects a control qubit $c$ connected to five target qubits $t_1, \ldots, t_5$ against up to $t = 2$ $\boldsymbol{X}$ faults. Analogously, to protect a target qubit connected to five control qubits against $\boldsymbol{Z}$ faults, the $\boldsymbol{Z}$-detecting flag

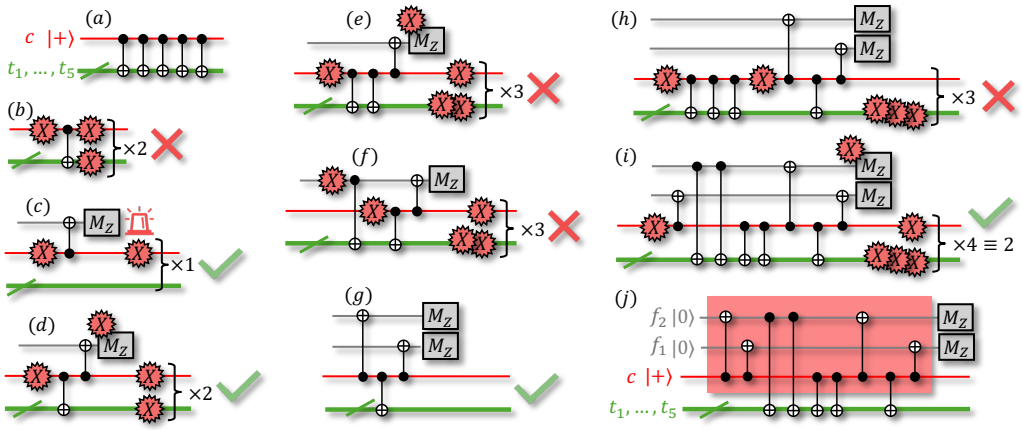


Figure 5.2: Algorithm to discover flag gadgets. $(a)$ The inputs are $t = 2$, representing the number of correctable faults; the number of target qubits; and the number of flags believed to be sufficient (2 in this example). $(b)$ The first attempted gate results in a non-FT circuit because a single fault can propagate into two faults. $(c)$ Adding flag qubit $f_1$ makes the gadget FT because no single fault propagates to an undetected error of weight greater than 1. $(d)$ Re-adding the first attempted gate now preserves fault-tolerance because even two faults do not propagate to an undetected error of weight greater than 2. $(e)$ Adding the next $\boldsymbol{CX}$ gate from $c$ to $t_2$ is not FT because two faults propagate to an undetected weight $w = 3$ error. $(f)$ The next $\boldsymbol{CX}$ in the pool controls $f_1$ instead. This is possible because at this point $f_1$ shares a GHZ-like entanglement with the control qubit, but this attempt is still not FT. $(g)$ Adding a new flag $f_2$ makes the gadget FT. $(h)$ After several successful steps, adding the last $\boldsymbol{CX}$ from $c$ to $t_4$ is not FT because two faults propagate undetected to a weight $w = 3$ error. $(i)$ Moving the control of the previous attempted gate to $f_2$ and disentangling $f_1$ makes the gadget FT despite two faults propagating to weight $w = 4$. This is because this error reduces to weight $w = 2$ up to the weight-6 stabilizer operator created by the circuit for the CSS code. $(j)$ The algorithm outputs the FT flag gadget.

gadget consists in a Hadamard-conjugated version of the one in the figure. This example serves as a basis for the explanation that follows.

To manage the construction of the circuit, three pools of $\boldsymbol{CX}$ gates are initialized and updated as the construction proceeds. The first pool, responsible for entangling target qubits, is initialized to entangle the first target qubit: $\mathcal{T}_0 = [\boldsymbol{CX}(c, t_1)]$. The second pool, responsible for entangling flag qubits, is initialized to entangle the first flag qubit: $\mathcal{E}_0 = [\boldsymbol{CX}(c, f_1)]$. The third pool, for disentangling flag qubits, is initialized as an empty list: $\mathcal{D}_0 = \varnothing$. Additionally, the set of entangled target qubits $T_0$, the set of entangled flags $E_0$, and the output circuit $\mathcal{C}_0$ are initialized as empty lists.

First, the algorithm adds the gate from $\mathcal{T}_0$ to a temporary circuit $\mathcal{C}_{\text{temp}} = [\boldsymbol{CX}(c, t_1)]$, as shown in Fig. 5.2b. This circuit is then used to test fault-tolerance. In Algorithm 7, the test is carried out through the function `IsFaultTolerant`. However, in this example, the FT test fails, as a single fault propagates into an error of weight $w = 2$. The algorithm then updates $\mathcal{T}_1 = \varnothing$, and leaves the other lists unchanged: $\mathcal{E}_1 = \mathcal{E}_0$, $\mathcal{D}_1 = \mathcal{D}_0$, $T_1 = T_0$, $E_1 = E_0$, and $\mathcal{C}_1 = \mathcal{C}_0$. Specifically, in Algorithm 7, we employ the function `EntangledQubits` to update the lists of entangled targets and flags based on the current circuit, and the function `AvailableGates` to update the three pools of available $\boldsymbol{CX}$ gates accordingly. Next, as the entangling target qubit pool is empty, the algorithm proceeds by adding the gate from $\mathcal{E}_1$ to $\mathcal{C}_{\text{temp}} = [\boldsymbol{CX}(c, f_1)]$, as shown in Fig. 5.2c. In this case, the FT test is successfully passed. Moreover, $f_1$ shares a GHZ-like entanglement with $c$, meaning it can be used interchangeably with $c$. The pools are updated accordingly: $\mathcal{T}_2 = [\boldsymbol{CX}(c, t_1), \boldsymbol{CX}(f_1, t_1)]$ to reattempt entangling $t_1$, $\mathcal{E}_2 = [\boldsymbol{CX}(c, f_2), \boldsymbol{CX}(f_1, f_2)]$ to entangle $f_2$, and $\mathcal{D}_2 = [\boldsymbol{CX}(c, f_1), \boldsymbol{CX}(f_1, c)]$ to disentangle $f_1$. The sets are updated to $T_2 = T_1$, $E_2 = \{f_1\}$, and the circuit becomes $\mathcal{C}_2 = \mathcal{C}_{\text{temp}}$. At this stage, the algorithm adds the first gate from $\mathcal{T}_2$ to the circuit: $\mathcal{C}_{\text{temp}} = [\boldsymbol{CX}(c, f_1), \boldsymbol{CX}(c, t_1)]$. As shown in Fig. 5.2d, the weight of the undetected error remains less than or equal to the number of faults. The pools and sets are updated to $\mathcal{T}_3 = [\boldsymbol{CX}(c, t_2), \boldsymbol{CX}(f_1, t_2)]$, $\mathcal{E}_3 = \mathcal{E}_2$, $\mathcal{D}_3 = \mathcal{D}_2$, $T_3 = \{t_1\}$, $E_3 = E_2$, and $\mathcal{C}_3 = \mathcal{C}_{\text{temp}}$. Fig. 5.1e–i illustrate additional successful and unsuccessful steps. Regarding Fig. 5.1i, note that target qubits $t_4$ and $t_5$ are entangled using $\boldsymbol{CX}$ gates whose control is a flag qubit rather than the designated

control qubit $c$. This is permissible because $c$ and $f_2$ become indistinguishable once they are entangled. Indeed, it is possible for the final preparation circuit to include a $\boldsymbol{CX}$ gate in which the control is a flag from an $\boldsymbol{X}$-detecting gadget and the target is a flag from an $\boldsymbol{Z}$-detecting gadget. Moreover, the specific two-fault combination illustrated propagates to an undetected error of weight $w = 4$. However, the stabilizer operator $\boldsymbol{X}_c\boldsymbol{X}_{t_1}\boldsymbol{X}_{t_2}\boldsymbol{X}_{t_3}\boldsymbol{X}_{t_4}\boldsymbol{X}_{t_5}$, obtained by propagating the stabilizer $\boldsymbol{X}_c$ of the control qubit's initial state $|+\rangle$ through the circuit, reduces this error to weight $w = 2$ under multiplication.

The algorithm terminates once the following conditions are met: $i)$ $T_s$ contains all target qubits; $ii)$ $E_s$ is empty for some step $s$; $iii)$ the fault-tolerance test is satisfied. In the example, these conditions are achieved with the gadget shown in Fig. 5.2j. If, at any step $s$, all three pools are exhausted before the termination conditions are met, the algorithm backtracks to the most recent successful step, selects the next unused gate from the pools, and proceeds to construct a new branch. If no valid configuration can be found after all gates across previous steps have been explored, the algorithm terminates without generating a FT gadget for the given inputs. By gradually increasing the number of flag qubits provided as input, the algorithm is able to identify optimal gadgets, as the first successful configuration found corresponds to the minimal number of flags required. Table 5.2 shows the number of flag qubits consumed by the flag gadgets discovered. Note that, for some large values of $t$, many target qubits, and a small number of flag qubits, no gadget may be found within reasonable time. In such cases, a larger number of flag qubits than initially expected is provided, enabling the algorithm to find a solution within seconds or minutes. These suboptimal flag counts are indicated in Table 5.2 using the $\leqslant$ symbol.

## 5.4 Decoding

In this section, we describe the noise model used in the simulations and the decoding strategy adopted to estimate the logical error rate of the prepared circuits. In the noise model, each qubit initialization is followed by a single-qubit depolarizing channel with error rate $p$, and every two-qubit gate is followed by a two-qubit depolarizing channel with the same error rate $p$. In addition, each flag-qubit mea-

**Algorithm 7:** Flag Gadget Discovery Algorithm

**input :** number of non-problematic faults $t$, targets $r$, and flags $m$
**output:** flag gadget $\mathcal{C}$

initialize step $s \leftarrow 0$;
initialize gadget $\mathcal{C}_s \leftarrow \varnothing$;
initialize disentangled targets $T_s \leftarrow [t_1, \ldots, t_r]$;
initialize disentangled flags $E_s \leftarrow [f_1, \ldots, f_r]$;
initialize gates to entangle targets $\mathcal{T}_s \leftarrow [\boldsymbol{CX}(c, t_1)]$;
initialize gates to entangle the flags $\mathcal{E}_s \leftarrow [\boldsymbol{CX}(c, f_1)]$;
initialize gates to disentangle the flags $\mathcal{D}_s \leftarrow \varnothing$;
initialize available gates $\mathcal{G}_s \leftarrow \mathcal{T}_s + \mathcal{E}_s + \mathcal{D}_s$;
**while** $T_s \neq \varnothing$ *and* $E_s \neq \varnothing$ **do**
  **if** *there are available gates* $\mathcal{G}_s \neq \varnothing$
    propose the next gate $G_s \leftarrow \mathcal{G}_s[0]$;
    $\mathcal{C}_{\text{temp}} \leftarrow \mathcal{C}_s + [G_s]$;
    **if** $\mathit{IsFaultTolerant}(\mathcal{C}_{\text{temp}}, t, r) =$ *True*
      remove the proposed gate $\mathcal{G}_s \leftarrow \mathcal{G}_s[1:]$;
      increase step $s \leftarrow s + 1$;
      add gadget to history $\mathcal{C}_s \leftarrow \mathcal{C}_{\text{temp}}$;
      $T_s, E_s \leftarrow \texttt{EntangledQubits}(\mathcal{C}_s, r, m)$;
      $\mathcal{T}_s, \mathcal{E}_s, \mathcal{D}_s \leftarrow \texttt{AvailableGates}(T_s, E_s, r, m)$;
      $\mathcal{G}_s \leftarrow \mathcal{T}_s + \mathcal{E}_s + \mathcal{D}_s$;
  **else**
    **if** $\mathit{IsFaultTolerant}(\mathcal{C}_s, t, r) =$ *True*
      **return** $\mathcal{C}_s$;
    **if** $\mathcal{G}_s = \varnothing$ *for all steps* $s$
      **return** "No FT gadget. Increase $m$";
    **else**
      remove last history step $\mathcal{G}_s = \mathcal{G}_{s-1}$;
      update gadget $\mathcal{C}_s \leftarrow \mathcal{C}_{s-1}$;
  propose new gate $G_s \leftarrow \mathcal{G}_s[0]$;

| tar. qbts. | 1–4 | 5 | 6 | 7–8 | 9–10 | 11 | 12–13 | 14 | 15–16 | 17 | 18 | 19 | 20–21 | 22 | 23 | 24 | 25 | 26–27 | 28–29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t=1$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $t=2$ | 1 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 |
| $t=3$ | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 6 | $\leqslant 7$ | $\leqslant 7$ | $\leqslant 8$ | $\leqslant 8$ | $\leqslant 9$ | $\leqslant 9$ | $\leqslant 9$ | $\leqslant 10$ | $\leqslant 10$ | $\leqslant 11$ | $\leqslant 11$ |
| $t=4$ | 1 | 2 | 3 | 4 | 6 | 7 | 7 | $\leqslant 8$ | $\leqslant 9$ | $\leqslant 10$ | $\leqslant 10$ | $\leqslant 11$ | $\leqslant 11$ | $\leqslant 12$ | $\leqslant 13$ | $\leqslant 13$ | $\leqslant 14$ | $\leqslant 14$ | $\leqslant 15$ |
| $t=5$ | 1 | 2 | 3 | 4 | 6 | $\leqslant 9$ | $\leqslant 9$ | $\leqslant 10$ | $\leqslant 11$ | $\leqslant 13$ | $\leqslant 13$ | $\leqslant 13$ | $\leqslant 14$ | $\leqslant 15$ | $\leqslant 15$ | $\leqslant 16$ | – | – | – |

Table 5.2: Size of FT flag gadgets discovered to protect one control qubit connected to several target qubits via $\boldsymbol{CX}$ gates at distance $d$. Column headers indicate the number of target qubits. The gadget requires two $\boldsymbol{CX}$ gates for every flag qubit. The symbol $\leqslant$ indicates that the number of flags may not be optimal.

surement is preceded by a single-qubit bit-flip channel with error rate $p$. Measurements of code qubits at the end of the circuit are assumed to be noiseless, since in practice such measurements are typically performed only after state preparation. Memory noise is modeled by applying a single-qubit depolarizing channel with error rate $p/100$ for every active qubit during each $\boldsymbol{CX}$ gate application. Qubits are initialized as late as possible, immediately before the first $\boldsymbol{CX}$ gate that acts on them, and flag qubits are measured as early as possible, immediately after their final $\boldsymbol{CX}$ gate, to limit the accumulation of memory noise. All code qubits are measured simultaneously only after the last flag qubit has been measured.

As regards decoding, performing simulations for large code distances at low physical error rates poses considerable numerical challenges. To obtain sufficient statistical accuracy, we perform high-speed circuit-level (CL) Pauli propagation simulations, generating up to $10^9$ random Pauli errors drawn from the noise model for each circuit and physical error rate under study. The obtained dataset is further expanded to as many as $10^{10}$ errors through subset sampling. Given the high computational cost of algorithmic decoders in this regime, we instead construct maximum-likelihood (ML) look-up tables (LUTs) from the simulated data. We use half of the error set as a training set to build the CL-ML-LUT, while the remaining half serves as a test set for estimating logical error rates. As the number of code qubits increases, it becomes more likely that syndromes in the test set are absent from the CL-ML-LUT. To address this, we supplement it with a code-capacity (CC) minimum-weight (MW) LUT, ensuring that any previously unseen syndromes can still be decoded.

### 5.4.1 Subset-sampling

In Monte Carlo simulations of Pauli error channels, at moderately low physical error rates $p$, it is highly likely that no errors occur, producing the trivial syndrome. To avoid expending computational resources on such instances, subset-sampling techniques are employed [164, 165]. We first count the number of circuit locations, $L_p$ and $L_q$, where faults can occur at rates $p$ and $q = p/100$, the two error rates considered in the noise model. From these, we compute the normalized probability distribution $\mathcal{P}_{p,q}(f_p, f_q)$ over the number of faults $f_p$ and $f_q$. Excluding the case with no faults, we obtain a normalized distribution $\mathcal{Q}_{p,q}(f_p, f_q)$ with $\mathcal{Q}_{p,q}(0, 0) = 0$. To reduce memory usage, we discard extremely unlikely fault events by removing $(f_p, f_q)$ pairs with probabilities $\mathcal{P}_{p,q}(f_p, f_q) \leq 1/S^2$, where $S$ denotes the number of Monte Carlo samples. Subsequently, we draw $S$ fault events from $\mathcal{Q}$ and, for each, generate Pauli errors at random according to the noise model. Finally, we account for the expected number of missing trivial fault events by restoring their known trivial syndromes to the final error set, thereby effectively increasing the total sample count.

### 5.4.2 Look-Up Table-Based Decoding Strategy

Once the final error set is obtained, each error instance is propagated through the circuit using Clifford simulations. During this process, errors that trigger a flag measurement are recorded to estimate the acceptance rate. When all flag measurements are trivial, the corresponding syndromes and equivalence classes are stored. Equivalence classes are determined by checking whether an error commutes or anticommutes with a chosen logical operator representative that stabilizes the logical state: logical $\boldsymbol{Z}_L$ for $\left|\overline{0}\right\rangle$ and logical $\boldsymbol{X}_L$ for $|\overline{+}\rangle$. For the $[[20, 2, 6]]$ code, logical qubits are prepared in the state $\left|\overline{00}\right\rangle$, resulting in four equivalence classes corresponding to whether the error commutes or anticommutes with each of the two logical $\boldsymbol{Z}_L$ operators. To prevent overfitting, the dataset of (syndrome, equivalence) class pairs is randomly split into two equal subsets: one used as the training set to generate the CL-ML-LUT, and the other as the test set to evaluate logical error rates. Specifically, the CL-ML-LUT stores, for each syndrome observed in the training set, the most likely equivalence class.

The CC-MW-LUT for preparing the logical state $|\overline{0}\rangle$ is built by considering all $\boldsymbol{X}$ errors of weight $1$ up to $t$ on the ideally prepared state, and recording their corresponding (syndrome, equivalence class) pairs. By definition, all such errors are correctable; therefore, any errors that produce the same syndrome must belong to the same equivalence class. Consequently, the CC-MW-LUT can be implemented as a dictionary that maps each syndrome to a unique equivalence class.

After both LUTs have been populated, they are used to decode the shots in the test error set. Firstly, the CL-ML-LUT is used to decode every (syndrome, equivalence class) pair in the test set. An instance is considered uncorrected if its equivalence class differs from the class predicted by the CL-ML-LUT. If the syndrome is not present in the CL-ML-LUT, the CC-MW-LUT is consulted. In case the syndrome is absent from both tables, a logical error is declared whenever the equivalence class corresponds to an error that anticommutes with the logical operator. Although such cases are rare, when they occur the applied correction is likely to result in a logical error. Despite the strong performance of the CL-ML-LUT and CC-MW-LUT decoders, the suboptimal behavior of the final decoding layer explains the unexpectedly high logical error rates observed for large codes in the simulations (see Table 5.1). As the number of possible syndromes grows, the likelihood of encountering them in the LUTs decreases. Consequently, whenever an error anticommutes with the logical operator, it is treated as a logical error. A possible solution to this issue is to append an algorithmic decoder at the final stage, used only when the syndrome is absent from both LUTs. Decoders such as BP+OSD, which are well suited for QLDPC codes, have cubic complexity in the number of qubits; however, this step would only be required for the small fraction of shots not already decoded by the LUTs. By replacing the final decoding layer with an algorithmic decoder capable of handling any syndrome, one could significantly improve logical error rates while still maintaining a fast and accurate decoding procedure in practice.

Finally, for codes with even distance, such as the $[[20, 2, 6]]$ code, errors of weight $t = d/2$ are not expected to be corrected. In these cases, the entire computation is discarded whenever such errors are detected.

## 5.5 Numerical and Hardware Results

This section presents the numerical and hardware results obtained employing the decoding pipeline described in Section 5.4.

*Performance of preparation circuits:* The logical error rate and the discard rate (1 minus acceptance rate) as a function of the physical error rate are plotted in Fig. 5.3 for some CSS codes of interest. In the regime of low error rates, the simulations match the expected decays $\mathcal{O}(p^{t+1})$. This provides numerical confirmation that the preparation circuits are indeed FT up to $t$ faults. In the case of even-distance codes, some errors of weight $t = d/2$ can be detected but not corrected. Therefore, if a syndrome is compatible with a weight-$t$ error but not with lower-weight errors, instead of correcting the state, the entire computation is discarded during decoding, even after a successful state preparation. The resulting post-discard rate for the [[20,2,6]] code is plotted in the subplot at the bottom of Fig. 5.3, sowing the expected trend of $\mathcal{O}(p^t)$.

*Performance of Steane-QEC :* Since the logical error rate in the preparation of a logical $|\overline{0}\rangle$ state is sensitive only to $\boldsymbol{X}$ errors, performance against $\boldsymbol{Z}$ errors is studied using a Steane-QEC gadget. In this gadget, a logical $|\overline{0}\rangle$ resource state is fault-tolerantly prepared, a transversal $\boldsymbol{CX}$ is applied from the resource block to the computational block to be corrected, and the resource block is subsequently measured destructively in the $\boldsymbol{X}$ basis. Decoding the output provides information about the joint $\boldsymbol{Z}$ errors on the blocks and allows the application of a suitable correction to the computational block. In realistic settings, logical operations introduce more noise than freshly prepared resource states. To emulate this, we consider the following experiment: a computational block is noiselessly prepared in the logical $|\overline{+}\rangle$ state, single-qubit depolarizing channels of rate $10p$ are applied to each qubit, Steane-QEC is performed, and the same depolarizing channel is applied again on the computational block. The logical resource state $|\overline{0}\rangle$ for Steane-QEC is prepared using our construction and subjected to the noise model defined in Sec. 5.4 with error rate $p$. The same noise model is applied to the subsequent transversal $\boldsymbol{CX}$ gate, additional idle locations in the gadget, and the measurement of the resource state. The logical error rate of this experiment is compared to that obtained without Steane-QEC. Figure 5.4 shows the logical error

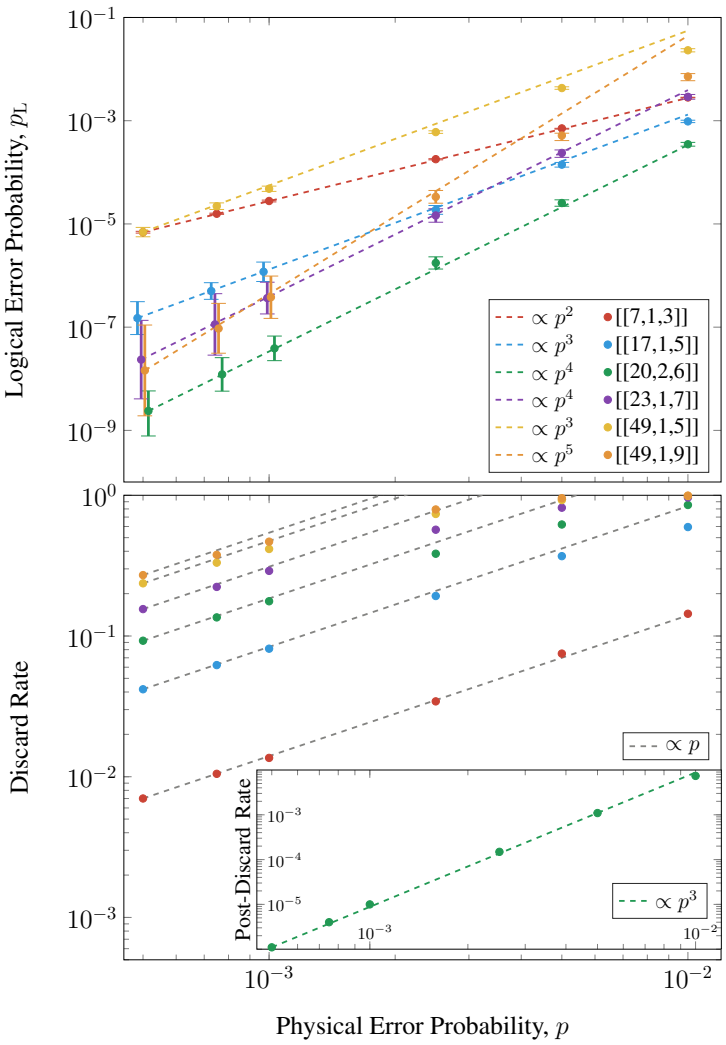


Figure 5.3: Logical error probability (top) and discard rate (bottom) vs. physical error probability for various CSS codes. Dashed lines visually indicate the expected $\mathcal{O}(p^{t+1})$ scaling with the code distance $d$. The subplot below shows the post-discard rate of the [[20,2,6]] code during decoding. Error bars indicate $95\%$ confidence intervals.

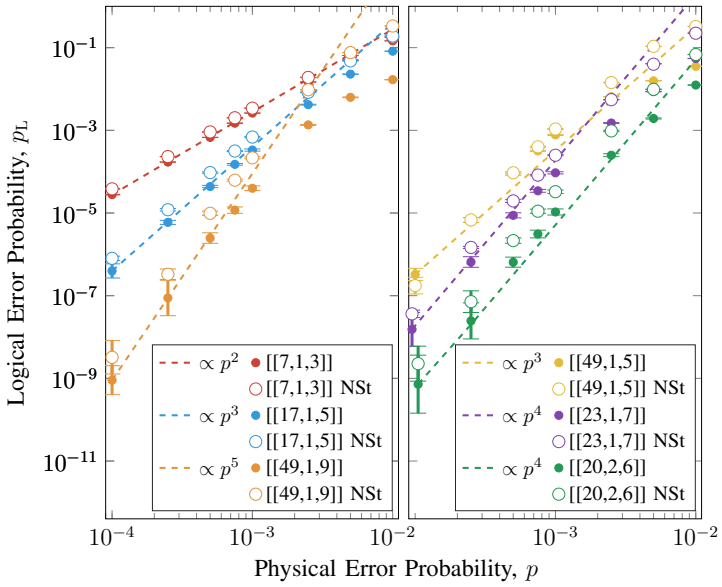


Figure 5.4: Logical error probability with and without (indicated by NSt in the legend) a Steane-QEC gadget vs. physical error probability for various CSS codes. Dashed lines indicate the expected $\mathcal{O}(p^{t+1})$ scaling with code distance $d$, where $t = \lfloor d/2 \rfloor$ is the number of non-problematic faults. Error bars represent 95% confidence intervals.

rate as a function of the physical error rate $p$. For five of the six CSS codes studied, Steane-QEC reduces the logical error rate across the simulated range by up to a factor of two. The $[[49, 1, 5]]$ triorthogonal code is an exception, showing no clear improvement, likely due to the substantial circuit overhead required for FT preparation. Dashed lines indicate the expected $\mathcal{O}(p^{t+1})$ scaling; apart from the $[[49, 1, 5]]$ case, the results follow the predicted scaling in the low-error regime.

*Fault-tolerance of both $\boldsymbol{X}$ and $\boldsymbol{Z}$ Pauli errors:* Protecting against $\boldsymbol{Z}$ errors during the preparation of $|\overline{0}\rangle$ may seem unnecessary, since these errors do not immediately cause a logical fault. However, $\boldsymbol{Z}$ errors can persist in the system, later propagating through logical gates or Steane-QEC gadgets. As they accumulate, they increase the risk of logical failure. To illustrate this risk, we present a numerical demonstration in the setting of Steane-QEC. Specifically, when the $|\overline{0}\rangle$ state serves as the control of a transversal $\boldsymbol{CX}$ in a Steane-QEC gadget, any $\boldsymbol{Z}$ errors in the $|\overline{0}\rangle$ state can combine with $\boldsymbol{Z}$ errors propagated from the data qubit under correction, resulting in an incorrect recovery operation. Figure 5.5 shows the logical error rate of the Steane-QEC gadget when the $|\overline{0}\rangle$ state of the $[[17, 1, 5]]$ color

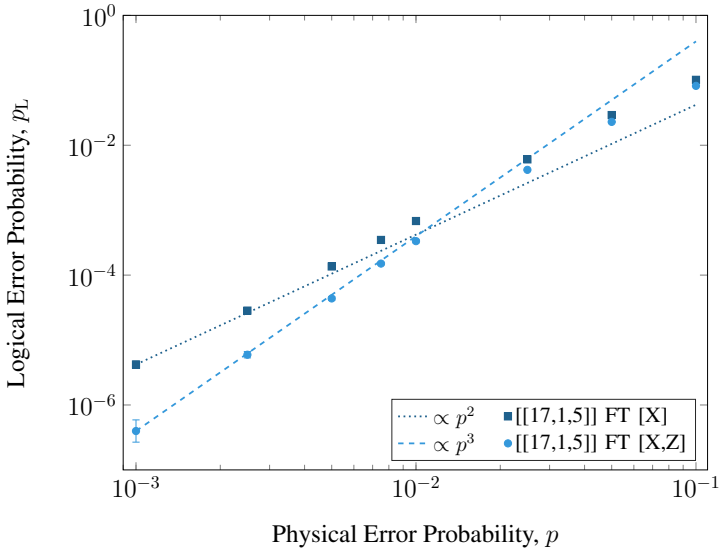


Figure 5.5: Logical error probability vs. physical error probability for a Steane-QEC gadget in which the resource state $|\overline{0}\rangle$ is prepared either with or without FT protection against $Z$ errors. The light dashed line shows the expected $\mathcal{O}(p^3)$ scaling for a distance-$d = 5$ code, while the dark dashed line shows the $\mathcal{O}(p^2)$ scaling expected for an effective distance-$d = 3$ code. Error bars indicate $95\%$ confidence intervals.

code is prepared fault-tolerantly against $\boldsymbol{X}$ errors but not against $\boldsymbol{Z}$ errors. The figure demonstrates that the scheme fails to achieve the expected $O(p^3)$ logical error scaling characteristic of a fully FT gadget, as illustrated in Fig. 5.4. This result highlights the importance of preparing ancillary states such as $|\overline{0}\rangle$ and $|\overline{+}\rangle$ fault-tolerantly with respect to both $\boldsymbol{X}$ and $\boldsymbol{Z}$ errors.

*Hardware results:* We implement the FT preparation of the $|\overline{0}\rangle$ state for the $[[23, 1, 7]]$ Golay code on Quantinuum's H2-1 and H2-2 trapped-ion quantum computers. The results are summarized in Table 5.3. Decoding is performed using LUTs generated from numerical simulations at an error rate of $p = 10^{-3}$, as described in Sec. 5.4. Memory noise in Quantinuum devices can be partially approximated by single-qubit coherent rotations $\exp(-i\delta Z)$ on every qubit at every time step, with the same small angle $\delta$ [166]. If unmitigated, these rotations accumulate over time and propagate to other qubits through $\boldsymbol{CX}$ gates. To optimize performance, we explore four different settings for mitigating such memory noise, combining compilations in terms of either $\boldsymbol{CX}$ or $\boldsymbol{CZ}$ gates with three variants of dynamical decoupling (DD) [167]. Specifically, compiling the circuit in terms

of $\boldsymbol{CX}$ gates with $\boldsymbol{CZ}$ gates prevents this propagation, since $\boldsymbol{Z}$ Pauli noise commutes with the unwanted rotations. Moreover, DD inserts Pauli-$\boldsymbol{X}$ and Pauli-$\boldsymbol{Y}$ operators at regular intervals without affecting the final logical state. These operators reverse the direction of the coherent rotations, effectively canceling them. We test three DD configurations: i) no DD, ii) the default software DD in Quantinuum devices [166], which accounts for precise timing and ion positioning, and iii) a custom DD that simply inserts an $\boldsymbol{XX}$ Pauli operator after every two-qubit gate. Similarly to the experimental setting employed in the previous numerical simulations, all qubits are initialized as late as possible, all flag qubits are measured as early as possible, and all code qubits are measured simultaneously after the last measured flag qubit.

Across all experiments, the acceptance rate ranges from $38\%$ to $51\%$. The best performance is obtained when using $\boldsymbol{CZ}$ gates in combination with DD. Aggregating the four experiments E3–E6 on H2-1, which use $\boldsymbol{CZ}$ gates with varying DD schemes, yields 6,140 accepted runs out of 13,000 preparation attempts—an acceptance rate of $47.23\%$. Among these accepted attempts, only two fail, corresponding to an average logical error rate of $3.3 \times 10^{-4}$ with a $95\%$ Wilson confidence interval of $[0.9, 11.9] \times 10^{-4}$. H2-2 shows a slightly higher average logical error rate of $5.18 \times 10^{-4}$ with a confidence interval of $[1.4, 18.9] \times 10^{-4}$. For context, the logical error rate without additional post-discarding is better (within $95\%$ confidence intervals) than the minimum hardware SPAM error rate of $6.0(1.6) \times 10^{-4}$ of a physical $|0\rangle$ in Quantinuum H2-1 and H2-2 [168]. It is also consistent with the state-of-the-art logical error rates reported in QEC experiments with distances greater than two and with moderate post-selection: $8^{+16}_{-6} \times 10^{-4}$ with the tesseract code [169], and with the Steane code, $5.1(2.7) \times 10^{-4}$ [56], $9(2) \times 10^{-4}$ [170], $5^{+4}_{-3} \times 10^{-4}$ [171], and $4.1(1.3) \times 10^{-4}$ [172].

| | E1 | E2 | E3 | E4 | E5 | E6 | E7 |
|---|---|---|---|---|---|---|---|
| Machine | H2-1 | H2-1 | H2-1 | H2-1 | H2-1 | H2-1 | H2-2 |
| 2q gates | CX | CX | CZ | CZ | CZ | CZ | CZ |
| DD | None | Def. | None | Def. | Cust. | Def. | Def. |
| Attempts | 1000 | 1000 | 1000 | 1000 | 1000 | 10000 | 10000 |
| Acc. Att. | 411 | 509 | 389 | 493 | 426 | 4832 | 3859 |
| LE post. | 10 | 1 | 0 | 0 | 0 | 2 | 2 |

Table 5.3: Hardware experiment settings and results. In descending order by row: Quantinuum hardware, compilation type ($\boldsymbol{CX}$ or $\boldsymbol{CZ}$ gates), dynamical decoupling (DD) strategy used (none, default, or custom), number of initialization attempts, number of accepted attempts, number of logical errors within them.

# Conclusion

The first part of this thesis focused on the analysis and design of quantum error-correcting codes. Chapter 2 introduced the undetectable errors WE polynomial, derived from the quantum MacWilliams identities. Each coefficient of this polynomial corresponds to the number of logical operators of a given quantum code and provides a way to evaluate upper bounds on its error-correcting capability. We employed this tool to analyze the logical error rates of topological codes of various distances, highlighting their strengths and limitations under both depolarizing and biased physical noise.

Motivated by the prevalence of noise asymmetry in experimental platforms, Chapter 3 presented the ZZZY codes, a new family of codes tailored for asymmetric channels. In these constructions, selected $\boldsymbol{Z}$ operators in the stabilizers of the surface code are replaced by $\boldsymbol{Y}$ operators, yielding improved performance under biased noise. Furthermore, building on the homological product construction, we introduced cylindrical codes, and by applying structural twists to the resulting stabilizers, we designed the Möbius codes. We demonstrated that these new codes outperform standard surface codes when deployed over asymmetric noise channels.

In Chapter 4, we turned to the problem of real-time decoding, where sub-microsecond processing is essential for FT operation. We proved that, under a suitably chosen metric, all distinct matchings in the defect graph of a surface code are equivalent to the MW matching. This implies that decoding with respect to this metric guarantees correction up to the code distance. Building on this insight, we proposed three new low-complexity decoders. In particular, the BC decoder achieves a complexity that scales quadratically in the number of defects, which corresponds to linear scaling in the number of data qubits for moderate

error rates. Numerical simulations show that these decoders significantly reduce decoding latency compared to standard surface code decoders, at the expense of a modest trade-off in logical performance.

The second part of the thesis addressed the development of FT state preparation protocols. Stabilizer states occupy a central position in universal quantum computation, providing indispensable resources for quantum algorithms, error-correcting procedures, and a broad spectrum of applications. We showed that CSS states can be prepared using bipartite circuits composed of $\boldsymbol{CX}$ and Hadamard gates, where the bipartition is naturally defined by the control and target qubits of the $\boldsymbol{CX}$ gates. The proposed scheme then exploits the asymmetric propagation of Pauli errors in such circuits: Pauli-$\boldsymbol{X}$ errors propagate exclusively from control to target qubits, whereas Pauli-$\boldsymbol{Z}$ errors propagate from target to control qubits. By appending $\boldsymbol{X}$- and $\boldsymbol{Z}$-detecting flag gadgets to the respective partitions, the scheme achieves fault-tolerance in a scalable and modular manner. Using this framework, we constructed preparation circuits for surface, color, and triorthogonal codes, obtaining significantly more resource-efficient designs than previous methods, particularly for large codes. As a final demonstration, a logical zero state of the $[[23, 1, 7]]$ Golay code was successfully implemented on the Quantinuum H2 trapped-ion quantum computer using a circuit of $237$ $\boldsymbol{CX}$ gates, providing concrete experimental evidence of scalable FT state preparation.

# List of Figures

# Bibliography


[1] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, Jun. 1982.

[2] I. M. Georgescu, S. Ashhab, and F. Nori, "Quantum simulation," *Rev. Mod. Phys.*, vol. 86, pp. 153–185, Mar 2014.

[3] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[4] A. Das and B. K. Chakrabarti, "Colloquium: Quantum annealing and analog quantum computation," *Rev. Mod. Phys.*, vol. 80, pp. 1061–1081, Sep 2008.

[5] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[6] C. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theoretical Computer Science*, vol. 560, pp. 175–179, 1984.

[7] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[8] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, p. eaam9288, 2018.

[9] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti, "Towards a distributed quantum computing ecosystem," *IET Quantum Communication*, vol. 1, no. 1, pp. 3–8, 2020.

[10] J. Illiano, M. Caleffi, A. Manzalini, and A. S. Cacciapuoti, "Quantum internet protocol stack: A comprehensive survey," *Computer Networks*, vol. 213, p. 109092, 2022.

[11] M. Chiani, A. Conti, and M. Z. Win, "Piggybacking on quantum streams," *Phys. Rev. A*, vol. 102, no. 1, jul 2020.

[12] M. Sena, M. Flament, S. Andrewski *et al.*, "Robust high-fidelity quantum entanglement distribution over large-scale metropolitan fiber networks with co-propagating classical signals," *arXiv preprint arXiv:2504.08927*, 2025.

[13] M. Kjaergaard, M. E. Schwartz, J. Braumüller *et al.*, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020.

[14] C. Monroe, W. C. Campbell, L.-M. Duan *et al.*, "Programmable quantum simulations of spin systems with trapped ions," *Reviews of Modern Physics*, vol. 93, no. 2, p. 025001, 2021.

[15] M. Saffman, T. G. Walker, and K. Mølmer, "Quantum information with Rydberg atoms," *Reviews of modern physics*, vol. 82, no. 3, pp. 2313–2363, 2010.

[16] J. L. O'brien, A. Furusawa, and J. Vučković, "Photonic quantum technologies," *Nature photonics*, vol. 3, no. 12, pp. 687–695, 2009.

[17] D. Dieks, "Communication by EPR devices," *Physics Letters A*, vol. 92, no. 6, pp. 271–272, 1982.

[18] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, 1982.

[19] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum Hamming bound," *Phys. Rev. A*, vol. 54, pp. 1862–1868, Sep 1996.

[20] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, pp. R2493–R2496, Oct 1995.

[21] ——, "Fault-tolerant quantum computation," in *Proceedings of 37th conference on foundations of computer science*. IEEE, 1996, pp. 56–65.

[22] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[23] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[24] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Phys. Rev. A*, vol. 54, no. 2, p. 1098, 1996.

[25] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[26] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," *arXiv preprint quant-ph/9811052*, 1998.

[27] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, sep 2002.

[28] H. Bombin and M. A. Martin-Delgado, "Topological quantum distillation," *Phys. Rev. Lett.*, vol. 97, no. 18, p. 180501, 2006.

[29] N. P. Breuckmann, C. Vuillot, E. Campbell, A. Krishna, and B. M. Terhal, "Hyperbolic and semi-hyperbolic surface codes for quantum storage," *Quantum Science and Technology*, vol. 2, no. 3, p. 035007, aug 2017.

[30] J. P. B. Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, "The XZZX surface code," *Nature Communications*, vol. 12, no. 1, apr 2021.

[31] D. Gottesman, "The heisenberg representation of quantum computers," *arXiv preprint arXiv:quant-ph/9807006*, 1998.

[32] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, no. 2, p. 022316, 2005.

[33] J. T. Anderson, G. Duclos-Cianci, and D. Poulin, "Fault-tolerant conversion between the steane and reed-muller quantum codes," *Phys. Rev. Lett.*, vol. 113, p. 080501, Aug 2014.

[34] D. Litinski, "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery," *Quantum*, vol. 3, p. 128, Mar. 2019.

[35] L. Z. Cohen, I. H. Kim, S. D. Bartlett, and B. J. Brown, "Low-overhead fault-tolerant quantum computing using long-range connectivity," *Science Advances*, vol. 8, no. 20, May 2022.

[36] D. J. Williamson and T. J. Yoder, "Low-overhead fault-tolerant quantum computation by gauging logical operators," 2024.

[37] A. Cross, Z. He, P. Rall, and T. Yoder, "Improved QLDPC surgery: Logical measurements and bridging codes," 2024.

[38] D. J. MacKay, G. Mitchison, and P. L. McFadden, "Sparse-graph codes for quantum error correction," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2315–2330, 2004.

[39] A. A. Kovalev and L. P. Pryadko, "Quantum Kronecker sum-product low-density parity-check codes with finite rate," *Physical Review A*, vol. 88, no. 1, Jul. 2013.

[40] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, "High-threshold and low-overhead fault-tolerant quantum memory," *Nature*, vol. 627, no. 8005, pp. 778–782, 2024.

[41] J.-P. Tillich and G. Zémor, "Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength," *IEEE Trans. Inf. Theory*, vol. 60, no. 2, pp. 1193–1202, 2013.

[42] S. Bravyi and M. B. Hastings, "Homological product codes," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, pp. 273–282.

[43] M. B. Hastings, J. Haah, and R. O'Donnell, "Fiber bundle codes: breaking the $n^{1/2}$polylog(n) barrier for quantum LDPC codes," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021, pp. 1276–1288.

[44] P. Panteleev and G. Kalachev, "Quantum LDPC codes with almost linear minimum distance," *IEEE Trans. Inf. Theory*, vol. 68, no. 1, pp. 213–229, 2022.

[45] S. Bravyi and J. Haah, "Magic-state distillation with low overhead," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 5, p. 052329, 2012.

[46] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann *et al.*, "Realizing repeated quantum error correction in a distance-three surface code," *Nature*, vol. 605, no. 7911, pp. 669–674, 2022.

[47] Y. Zhao, Y. Ye, H.-L. Huang *et al.*, "Realization of an error-correcting surface code with superconducting qubits," *Phys. Rev. Lett.*, vol. 129, p. 030501, Jul 2022.

[48] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, "Improved decoding of circuit noise and fragile boundaries of tailored surface codes," *Phys. Rev. X*, vol. 13, p. 031007, Jul 2023.

[49] Google Quantum AI, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[50] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, "Improved decoding of circuit noise and fragile boundaries of tailored surface codes," *Phys. Rev. X*, vol. 13, p. 031007, Jul 2023.

[51] Google Quantum AI, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, pp. 920–926, 2025.

[52] O. Higgott and C. Gidney, "Sparse Blossom: correcting a million errors per core second with minimum-weight matching," *Quantum*, vol. 9, p. 1600, Jan. 2025.

[53] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, p. 58–65, 2023.

[54] P. Sales Rodriguez, J. M. Robinson, P. N. Jepsen, Z. He, C. Duckering, C. Zhao, K.-H. Wu, J. Campo, K. Bagnall, M. Kwon *et al.*, "Experimental demonstration of logical magic state distillation," *Nature*, pp. 1–3, 2025.

[55] S. Dasu, S. Burton, K. Mayer, D. Amaro, J. A. Gerber, K. Gilmore, D. Gresh, D. DelVento, A. C. Potter, and D. Hayes, "Breaking even with magic: demonstration of a high-fidelity logical non-clifford gate," *arXiv 2506.14688*, 2025.

[56] L. Daguerre, R. Blume-Kohout, N. C. Brown, D. Hayes, and I. H. Kim, "Experimental demonstration of high-fidelity logical magic states from code switching," *arXiv preprint arXiv:2506.14169*, 2025.

[57] Y. Hong, E. Durso-Sabina, D. Hayes, and A. Lucas, "Entangling four logical qubits beyond break-even in a nonlocal code," *Phys. Rev. Lett.*, vol. 133, no. 18, p. 180601, 2024.

[58] P. Panteleev and G. Kalachev, "Degenerate quantum LDPC codes with good finite length performance," *Quantum*, vol. 5, p. 585, 2021.

[59] N. Berthusen, J. Dreiling, C. Foltz *et al.*, "Experiments with the four-dimensional surface code on a quantum charge-coupled device quantum computer," *Phys. Rev. A*, vol. 110, no. 6, p. 062413, 2024.

[60] K. Wang, Z. Lu, C. Zhang, G. Liu, J. Chen, Y. Wang, Y. Wu, S. Xu, X. Zhu, F. Jin *et al.*, "Demonstration of low-overhead quantum error correction codes," *arXiv preprint arXiv:2505.09684*, 2025.

[61] A. Robertson, C. Granade, S. D. Bartlett, and S. T. Flammia, "Tailored codes for small quantum memories," *Physical Review Applied*, vol. 8, no. 6, p. 064004, 2017.

[62] Y. Seis, B. J. Brown, A. S. Sørensen, and J. F. Goodwin, "Improving trapped-ion-qubit memories via code-mediated error-channel balancing," *Phys. Rev. A*, vol. 107, no. 5, p. 052417, 2023.

[63] D. Forlivesi, L. Valentini, and M. Chiani, "Performance analysis of quantum CSS error-correcting codes via MacWilliams identities," *Quantum*, vol. 9, p. 1950, 2025.

[64] ——, "Logical error rates of XZZX and rotated quantum surface codes," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 7, pp. 1808–1817, 2024.

[65] L. Valentini, D. Forlivesi, and M. Chiani, "Performance analysis of quantum error-correcting surface codes over asymmetric channels," in *ICC 2023*. IEEE, 2023, pp. 4175–4181.

[66] D. Forlivesi, L. Valentini, and M. Chiani, "Quantum codes for asymmetric channels: ZZZY surface codes," *IEEE Commun. Lett.*, vol. 28, no. 10, pp. 2233–2237, 2024.

[67] L. Valentini, D. Forlivesi, and M. Chiani, "Cylindrical and Möbius quantum codes for asymmetric Pauli errors," *IEEE Trans. Inf. Theory*, vol. 71, no. 5, pp. 3766–3778, 2025.

[68] D. Forlivesi, L. Valentini, and M. Chiani, "Spanning tree matching decoder for quantum surface codes," *IEEE Commun. Lett.*, vol. 28, no. 7, pp. 1509–1513, 2024.

[69] ——, "Bubble clustering decoder for quantum topological codes," *IEEE Trans. Commun.*, vol. 73, no. 10, pp. 8470–8483, 2025.

[70] L. Valentini, D. Forlivesi, and M. Chiani, "Impact of decoding latency in the assessment of quantum surface codes performance," in *2025 IEEE QCNC*. IEEE, 2025, pp. 554–559.

[71] D. Forlivesi and D. Amaro, "Flag at origin: a modular fault-tolerant preparation for CSS codes," *arXiv preprint arXiv:2508.14200*, 2025.

[72] D. Gottesman, "An introduction to quantum error correction and fault-tolerant quantum computation," in *Quantum information science and its contributions to mathematics, PSAM*, vol. 68, 2010, pp. 13–58.

[73] M. Chiani and L. Valentini, "Short codes for quantum channels with one prevalent Pauli error type," *IEEE J. on Selected Areas in Information Theory*, vol. 1, no. 2, pp. 480–486, 2020.

[74] P. Shor and R. Laflamme, "Quantum analog of the macwilliams identities for classical coding theory," *Phys. Rev. Lett.*, vol. 78, no. 8, p. 1600, 1997.

[75] E. M. Rains, "Quantum weight enumerators," *IEEE Trans. Inf. Theory*, vol. 44, no. 4, pp. 1388–1394, 1998.

[76] C. Cao and B. Lackey, "Quantum weight enumerators and tensor networks," *IEEE Trans. Inf. Theory*, pp. 1–1, 2023.

[77] F. MacWilliams and N. Sloane, *The theory of error-correcting codes*. Amsterdam, the Netherlands: North Holland Mathematical Libray, 1977, vol. 16.

[78] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, "Quantum error correction via codes over GF(4)," *IEEE Trans. Inf. Theory*, vol. 44, no. 4, pp. 1369–1387, 1998.

[79] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Trans. Inf. Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[80] T. A. Gulliver, V. K. Bhargava, and J. M. Stein, "Q-ary gray codes and weight distributions," *Applied mathematics and computation*, vol. 103, no. 1, pp. 97–109, 1999.

[81] I. Bouyukliev, S. Bouyuklieva, T. Maruta, and P. Piperkov, "Characteristic vector and weight distribution of a linear code," *Cryptography and Communications*, vol. 13, no. 2, pp. 263–282, 2021.

[82] K. Zimmermann, *Integral Hecke Modules, Integral Generalized Reed-Muller Codes, and Linear Codes*. Techn. Univ. Hamburg-Harburg, 1996.

[83] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, and A. Wassermann, *Error-correcting linear codes: Classification by isometry and applications*. Springer Science & Business Media, 2006, vol. 18.

[84] A. Canteaut and F. Chabaud, "A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense bch codes of length 511," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 367–378, 1998.

[85] M. Grassl, "Searching for linear codes with large minimum distance," in *Discovering Mathematics with Magma: Reducing the Abstract to the Concrete*. Springer, 2006, pp. 287–313.

[86] P. Lisonek and L. Trummer, "Algorithms for the minimum weight of linear codes." *Adv. Math. Commun.*, vol. 10, no. 1, pp. 195–207, 2016.

[87] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system I: The user language," *Journal of Symbolic Computation*, vol. 24, no. 3-4, pp. 235–265, 1997.

[88] C. Cao, M. J. Gullans, B. Lackey, and Z. Wang, "Quantum lego expansion pack: Enumerators from tensor networks," 2023.

[89] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, no. 5, p. 793, 1996.

[90] J. Lee, J. Park, and J. Heo, "Rectangular surface code under biased noise," *Quantum Information Processing*, vol. 20, pp. 1–16, 2021.

[91] F. H. Watson and S. D. Barrett, "Logical error rate scaling of the toric code," *New Journal of Physics*, vol. 16, no. 9, p. 093045, 2014.

[92] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, "Perfect quantum error correcting code," *Phys. Rev. Lett.*, vol. 77, no. 1, p. 198, 1996.

[93] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, no. 3, sep 2012.

[94] A. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time," *Quantum Information and Computation*, vol. 15, no. 1-2, pp. 145–158, 2014.

[95] C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, dec 2012.

[96] R. Chao and B. W. Reichardt, "Fault-tolerant quantum computation with few qubits," *npj Quantum Information*, vol. 4, no. 1, p. 42, 2018.

[97] C. Chamberland and M. E. Beverland, "Flag fault-tolerant error correction with arbitrary distance codes," *Quantum*, vol. 2, p. 53, Feb. 2018.

[98] A. M. Steane, "Active stabilization, quantum computation, and quantum state synthesis," *Physical Review Letters*, vol. 78, no. 11, p. 2252–2255, Mar. 1997.

[99] P. Prabhu and B. W. Reichardt, "Fault-tolerant syndrome extraction and cat state preparation with fewer qubits," *Quantum*, vol. 7, p. 1154, Oct. 2023.

[100] B. Dezső, A. Jüttner, and P. Kovács, "LEMON–an open source C++ graph template library," *Electronic notes in theoretical computer science*, vol. 264, no. 5, pp. 23–45, 2011.

[101] H. V. Nguyen, Z. Babar, D. Alanis *et al.*, "Exit-chart aided quantum code design improves the normalised throughput of realistic quantum devices," *IEEE Access*, vol. 4, pp. 10 194–10 209, 2016.

[102] R. Lescanne, M. Villiers, T. Peronnin *et al.*, "Exponential suppression of bit-flips in a qubit encoded in an oscillator," *Nature Physics*, vol. 16, no. 5, pp. 509–513, 2020.

[103] P. K. Sarvepalli, A. Klappenecker, and M. Rötteler, "Asymmetric quantum codes: constructions, bounds and performance," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 465, no. 2105, pp. 1645–1672, 2009.

[104] M. Chiani and L. Valentini, "Design of short codes for quantum channels with asymmetric Pauli errors," in *Computational Science – ICCS 2020*. Cham: Springer Int. Publishing, 2020, pp. 638–649.

[105] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison *et al.*, "Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays," *Nature Physics*, pp. 1–7, 2024.

[106] H. Bombin and M. A. Martin-Delgado, "Quantum measurements and gates by code deformation," *Journal of Physics A: Mathematical and Theoretical*, vol. 42, no. 9, p. 095302, feb 2009.

[107] C. Vuillot, L. Lao, B. Criger, C. G. Almudéver, K. Bertels, and B. M. Terhal, "Code deformation and lattice surgery are gauge fixing," *New Journal of Physics*, vol. 21, no. 3, p. 033028, 2019.

[108] G. E. Bredon, *Topology and geometry*. Springer Science & Business Media, 2013, vol. 139.

[109] A. Hatcher, *Algebraic topology*. Cambridge Univ Pr, 2005.

[110] S. Bravyi and M. B. Hastings, "Homological product codes," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, pp. 273–282.

[111] A. Strikis and L. Berent, "Quantum LDPC codes for modular architectures," *arXiv preprint arXiv:2209.14329*, 2022.

[112] W. Zeng and L. P. Pryadko, "Higher-dimensional quantum hypergraph-product codes with finite rates," *Phys. Rev. Lett.*, vol. 122, no. 23, p. 230501, 2019.

[113] N. P. Breuckmann and J. N. Eberhardt, "Balanced product quantum codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 10, pp. 6653–6674, 2021.

[114] ——, "Quantum low-density parity-check codes," *PRX Quantum*, vol. 2, no. 4, p. 040101, 2021.

[115] O. Higgott, "Pymatching: A Python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–16, 2022.

[116] B. J. Brown, "Conservation laws and quantum error correction: towards a generalised matching decoder," *IEEE BITS*, 2023.

[117] V. Kolmogorov, "Blossom V: a new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, vol. 1, pp. 43–67, 2009.

[118] B. M. Terhal, "Quantum error correction for quantum memories," *Rev. Mod. Phys.*, vol. 87, pp. 307–346, Apr 2015.

[119] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, vol. 5, p. 595, 2021.

[120] N. Delfosse and G. Zémor, "Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel," *Physical Review Research*, vol. 2, no. 3, p. 033042, 2020.

[121] Y. Wu, N. Liyanage, and L. Zhong, "An interpretation of union-find decoder on weighted graphs," *arXiv preprint arXiv:2211.03288*, 2022.

[122] S. Gicev, L. C. Hollenberg, and M. Usman, "A scalable and fast artificial neural network syndrome decoder for surface codes," *Quantum*, vol. 7, p. 1058, 2023.

[123] K. Meinerz, C.-Y. Park, and S. Trebst, "Scalable neural decoder for topological surface codes," *Phys. Rev. Lett.*, vol. 128, no. 8, p. 080505, 2022.

[124] M. Pacenti, M. F. Flanagan, D. Chytas, and B. Vasić, "Progressive-proximity bit-flipping for decoding surface codes," *IEEE Trans. Commun.*, 2024.

[125] S. Bravyi, M. Suchara, and A. Vargo, "Efficient algorithms for maximum likelihood decoding in the surface code," *Phys. Rev. A*, vol. 90, no. 3, p. 032326, 2014.

[126] J. Roffe, D. R. White, S. Burton, and E. Campbell, "Decoding across the quantum low-density parity-check code landscape," vol. 2, no. 4, p. 043423, 2020.

[127] B. Barber, K. M. Barnes, T. Bialas *et al.*, "A real-time, scalable, fast and resource-efficient decoder for a quantum computer," *Nature Electronics*, vol. 8, no. 1, p. 84–91, Jan. 2025.

[128] R. Diestel, A. Schrijver, and P. Seymour, "Graph theory," *Oberwolfach Reports*, vol. 4, no. 2, pp. 887–944, 2008.

[129] S. Hadfield, "Error rate thresholds for physical implementations of the surface code," Bachelor thesis, School of Physics, University of Melbourne. 2008.

[130] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[131] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, pp. R2493–R2496, Oct 1995.

[132] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[133] N. P. Breuckmann and J. N. Eberhardt, "Quantum low-density parity-check codes," *PRX Quantum*, vol. 2, p. 040101, Oct 2021.

[134] D. Gottesman and I. L. Chuang, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature*, vol. 402, no. 6760, pp. 390–393, Nov 1999.

[135] T. A. Brun, Y.-C. Zheng, K.-C. Hsu, J. Job, and C.-Y. Lai, "Teleportation-based fault-tolerant quantum computation in multi-qubit large block codes," *arXiv preprint arXiv:1504.03913*, 2015.

[136] H. Goto, "High-performance fault-tolerant quantum computing with many-hypercube codes," *Science Advances*, vol. 10, no. 36, p. eadp6388, 2024.

[137] A. M. Steane, "Active stabilization, quantum computation, and quantum state synthesis," *Phys. Rev. Lett.*, vol. 78, pp. 2252–2255, Mar 1997.

[138] S. Huang and K. R. Brown, "Between shor and steane: A unifying construction for measuring error syndromes," *Phys. Rev. Lett.*, vol. 127, p. 090505, Aug 2021.

[139] E. Knill, "Scalable quantum computing in the presence of large detected-error rates," *Phys. Rev. A*, vol. 71, p. 042322, Apr 2005.

[140] T. Yoder, E. Schoute, P. Rall *et al.*, "Tour de gross: A modular quantum computer based on bivariate bicycle codes," Jun 2025, arXiv preprint arXiv:2506.03094.

[141] J. Preskill, "Fault-tolerant quantum computation," in *Introduction to quantum computation and information*. World Scientific, 1998, pp. 213–269.

[142] D. Gottesman, "An introduction to quantum error correction," in *Proceedings of Symposia in Applied Mathematics*, vol. 58, 2002, pp. 221–236.

[143] A. M. Steane, "Fast fault-tolerant filtering of quantum codewords," *Quantum Physics*, 2002.

[144] P. J. Salas, "Simple fault-tolerant encoding over q-ary css quantum codes," *International Journal of Quantum Information*, vol. 5, no. 05, pp. 705–716, 2007.

[145] K. N. Patel, I. L. Markov, and J. P. Hayes, "Efficient synthesis of linear reversible circuits," *arXiv preprint arXiv:quant-ph/0302002*, 2003.

[146] R. Duncan, A. Kissinger, S. Perdrix, and J. van de Wetering, "Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus," *Quantum*, vol. 4, p. 279, Jun. 2020.

[147] M. Webster, S. Koutsioumpas, and D. E. Browne, "Heuristic and optimal synthesis of cnot and clifford circuits," *arXiv preprint arXiv:2503.14660*, 2025.

[148] A. M. Steane, "Overhead and noise threshold of fault-tolerant quantum error correction," *Phys. Rev. A*, vol. 68, p. 042322, Oct 2003.

[149] ——, "Fast fault-tolerant filtering of quantum codewords," *arXiv preprint quant-ph/0202036*, 2004.

[150] B. W. Reichardt, "Improved ancilla preparation scheme increases fault-tolerant threshold," *arXiv preprint quant-ph/0406025*, 2004.

[151] A. W. Cross, D. P. Divincenzo, and B. M. Terhal, "A comparative code study for quantum fault tolerance," *Quantum Info. Comput.*, vol. 9, no. 7, p. 541–572, Jul. 2009.

[152] Y.-C. Zheng, C.-Y. Lai, and T. A. Brun, "Efficient preparation of large-block-code ancilla states for fault-tolerant quantum computation," *Phys. Rev. A*, vol. 97, p. 032331, Mar 2018.

[153] A. Paetznick and B. W. Reichardt, "Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit golay code," *Quantum Inf. Comput.*, vol. 12, pp. 1034–1080, 2011.

[154] H. Goto, "Minimizing resource overheads for fault-tolerant preparation of encoded states of the steane code," *Scientific reports*, vol. 6, no. 1, p. 19578, 2016.

[155] H. Goto, Y. Ho, and T. Kanao, "Measurement-free fault-tolerant logical-zero-state encoding of the distance-three nine-qubit surface code in a one-dimensional qubit array," *Phys. Rev. Res.*, vol. 5, p. 043137, Nov 2023.

[156] R. Zen, J. Olle, L. Colmenarez, M. Puviani, M. Müller, and F. Marquardt, "Quantum circuit discovery for fault-tolerant logical state preparation with reinforcement learning," *Phys. Rev. X*, Sep 2025.

[157] T. Peham, L. Schmid, L. Berent, M. Müller, and R. Wille, "Automated synthesis of fault-tolerant state preparation circuits for quantum error-correction codes," *PRX Quantum*, vol. 6, p. 020330, May 2025.

[158] M. Van den Nest, J. Dehaene, and B. De Moor, "Graphical description of the action of local clifford transformations on graph states," *Phys. Rev. A*, vol. 69, p. 022316, Feb 2004.

[159] D. Amaro, "Stabgraph," July 2019.

[160] C. Chamberland and M. E. Beverland, "Flag fault-tolerant error correction with arbitrary distance codes," *Quantum*, vol. 2, p. 53, 2018.

[161] R. Chao and B. W. Reichardt, "Flag fault-tolerant error correction for any stabilizer code," *PRX Quantum*, vol. 1, no. 1, p. 010302, 2020.

[162] Y. Tomita and K. M. Svore, "Low-distance surface codes under realistic quantum noise," *Phys. Rev. A*, vol. 90, p. 062320, Dec 2014.

[163] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[164] M. Gutiérrez, M. Müller, and A. Bermúdez, "Transversality and lattice surgery: Exploring realistic routes toward coupled logical qubits with

trapped-ion quantum processors," *Phys. Rev. A*, vol. 99, p. 022330, Feb 2019.

[165] C. J. Trout, M. Li, M. Gutiérrez, Y. Wu, S.-T. Wang, L. Duan, and K. R. Brown, "Simulating the performance of a distance-3 surface code in a linear ion trap," *New Journal of Physics*, vol. 20, no. 4, p. 043038, apr 2018.

[166] M. DeCross, R. Haghshenas, M. Liu, E. Rinaldi *et al.*, "Computational power of random quantum circuits in arbitrary geometries," *Phys. Rev. X*, vol. 15, p. 021052, May 2025.

[167] L. Viola, E. Knill, and S. Lloyd, "Dynamical decoupling of open quantum systems," *Phys. Rev. Lett.*, vol. 82, pp. 2417–2421, Mar 1999.

[168] Quantinuum, "H2 spec sheet," June 2025.

[169] B. W. Reichardt, D. Aasen, R. Chao *et al.*, "Demonstration of quantum computation and error correction with a tesseract code," *arXiv preprint arXiv:2409.04628*, Sep 2024.

[170] K. Mayer, C. Ryan-Anderson, N. Brown *et al.*, "Benchmarking logical three-qubit quantum fourier transform encoded in the steane code on a trapped-ion quantum computer," *arXiv preprint arXiv:2404.08616*, 2024.

[171] A. Paetznick, M. P. da Silva, C. Ryan-Anderson, J. M. Bello-Rivas *et al.*, "Demonstration of logical qubits and repeated error correction with better-than-physical error rates," *arXiv preprint arXiv:2404.02280*, 2024.

[172] C. Ryan-Anderson, N. C. Brown, M. S. Allman *et al.*, "Implementing fault-tolerant entangling gates on the five-qubit code and the color code," *arXiv preprint arXiv:2208.01863*, Aug 2022.