UNIVERSITY OF CALIFORNIA,
IRVINE

Design Optimization for Defect-Based Surface Codes

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Samira Sayedsalehi

Thesis Committee:
Professor Nader Bagherzadeh, Chair
Professor Jean-Luc Gaudiot
Professor Maxim Shcherbakov

2025

DEDICATION

To my parents and brothers
in recognition of their love, help and support

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

ACKNOWLEDGMENTS

First and foremost, I am deeply grateful to my advisor, Professor Nader Bagherzadeh, for his steadfast guidance, insightful feedback, and unwavering support throughout this thesis. His high standards and encouragement continually pushed me to clarify my thinking and strengthen my results.

I would also like to thank Professor Ratko Pilipović of the *Faculty of Computer and Information Science, University of Ljubljana, Slovenia* for his generous guidance and constructive discussions that helped shape the direction of this work.

My sincere thanks go to Alberto A. Del Barrio and Guillermo Botella of the *Facultad de Informatica, Universidad Complutense de Madrid, 28040 Madrid, Spain* for their helpful suggestions, technical insights, and many practical pointers during the development and evaluation phases.

I am profoundly indebted to my parents and my brothers for their love, patience, and encouragement. Their belief in me made this journey possible.

VITA

Samira Sayedsalehi

EDUCATION

Master of Science in Computer Engineering **2025**
University of California, Irvine *Irvine, California*

RESEARCH EXPERIENCE

Visiting Research Assistant **2022-2024**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2024**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Developing and Analyzing the Defect-Based Surface Codes Using Optimization Algorithms **2025**
Quantum Reports 7(2):25, https://doi.org/10.3390/quantum7020025

SOFTWARE

Squab `http://quantum-squab.com/`

Python

ABSTRACT OF THE THESIS

Design Optimization for Defect-Based Surface Codes

By

Samira Sayedsalehi

MASTER OF SCIENCE in Computer Engineering

University of California, Irvine, 2025

Professor Nader Bagherzadeh, Chair

Fault tolerance is essential for scalable quantum computation, and surface codes are the leading candidates due to high error thresholds and nearest-neighbor compatibility. Their chief limitation is the physical-qubit overhead per logical qubit. This thesis focuses on the defect-based approach, where logical information is encoded by introducing holes (defects) in a planar surface code. I formulate a distance-constrained design-optimization problem that trades off logical-qubit density against logical error rate, and employ stochastic search (e.g., simulated annealing and genetic algorithms) to explore large layout spaces. Through simulations, I compare closed versus partially open defects, quantify sensitivity to key hyperparameters, and identify regimes where increased density is achievable without violating target distance constraints. The results provide practical guidelines for distance-aware layout synthesis and clarify when defect geometry yields genuine density gains versus unacceptable logical-error penalties, informing the co-design of high-density, fault-tolerant quantum memories and processors.

Chapter 1

Introduction

1.1 Context and Problem Setting

Quantum computing (QC) exploits the principles of quantum mechanics—superposition, entanglement, and interference—to address problems that are intractable for classical computers. Following Preskill's framing of the Noisy Intermediate-Scale Quantum (NISQ) era, current devices feature tens to hundreds of imperfect qubits whose computations are limited by gate infidelity, measurement error, crosstalk, leakage, and environmental decoherence [1]. Quantum Error Correction (QEC) is therefore essential for scalable quantum computation [2, 3, 4, 5].

Among QEC families, topological surface codes have emerged as a leading architecture due to their high error thresholds and strictly local (nearest-neighbor) stabilizer measurements on two-dimensional lattices [6, 7]. In the planar variant, data and ancilla qubits are arranged on a grid; local X- and Z-type stabilizers are repeatedly measured to produce syndromes that a decoder maps to likely error configurations. While robust, surface codes are resource-intensive: encoding a single logical qubit typically requires on the order of $\mathcal{O}(d^2)$ physical

qubits, where d is the code distance. Consequently, raising the code distance to suppress logical error rates (LERs) imposes substantial area and measurement overheads.

Two broad strategies exist for encoding many logical qubits in a surface-code processor:

1. **Patch-based (tile) architectures**, in which each logical qubit occupies its own patch, and multi-qubit gates are enacted via lattice-surgery (merge/split) operations between patches; and

2. **Defect-based architectures**, in which multiple logical qubits are encoded within a *single* surface by introducing holes (defects) created by omitting selected stabilizers.

This thesis investigates the *defect-based* approach. Removing stabilizers (creating closed or partially open holes) increases the dimension of the logical subspace and thus the number of logical qubits that can be stored on a fixed lattice. However, the same omissions shorten non-trivial cycles and typically increase the LER. Hence, a central design question is how to place and shape defects to maximize logical-qubit density while respecting an LER target compatible with higher-level algorithmic requirements.

1.2 Motivation

As experimental quantum platforms progress, architectural efficiency becomes a first-order concern. Hardware budgets—number of qubits, couplers, readout lines, cryogenic bandwidth—remain scarce. For surface-code processors, every additional physical qubit contributes to calibration complexity, control cross-talk, and cryo I/O limits. Therefore, methods that **increase logical-qubit density** without catastrophic degradation of reliability are highly desirable.

Defect-based encoding offers two compelling promises:

- **Spatial efficiency:** By packing many logical qubits within a single lattice through holes, one can reduce the number of separate patches and the ancillary area typically required for lattice-surgery-mediated [8] interactions.

- **Flexible trade-offs:** The number, size, and boundary type of holes (closed vs. partially open) directly tune the code's topology, enabling designers to algorithmically trade logical-qubit count against LER.

Realizing these benefits is non-trivial. The design space is combinatorial: each candidate layout (hole positions, sizes, and boundary types) changes the code's homology, distance, and decoder workload. Analytical objective functions are difficult to formulate; empirical evaluation (e.g., via fast simulators under a chosen noise channel) is often required. This motivates optimization-driven methodologies—such as Simulated Annealing (SA) and Genetic Algorithms (GA)—that can efficiently search large discrete spaces while steering toward constraint-satisfying layouts.

At the same time, practical usage demands clarity on scope and limits. For instance, benchmarking under an erasure channel offers computational advantages and qualitative alignment with more complex noise models. However, it idealizes perfect erasure localization; real devices incur detection overheads and imperfections that will degrade performance. A rigorous study should therefore (i) quantify what is achievable under idealized but informative assumptions, and (ii) surface the implications for more realistic settings.

1.3 Thesis Scope and Contributions

Problem Statement. Given a fixed two-dimensional square lattice and a target logical error rate (LER) under an erasure channel, determine defect configurations—counts, placements, and boundary types (closed or partially open)—that maximize the number of encodable logical qubits subject to the LER constraint. Conversely, for a fixed number of logical

qubits, explore how code distance (via defect sizing/spacing) influences the attainable LER.

Assumptions and Evaluation Model.

- *Noise model:* quantum erasure channel with probability p (typical benchmarks around $p = 0.1$), evaluated via a fast induced-homology-based simulator.
- *Decoder knowledge:* the decoder is assumed to know locations of erasures (idealized, but common for scoping and rapid exploration).
- *Lattice geometry:* planar square lattices with open outer boundaries.
- *Metrics:* (i) number of encoded logical qubits; (ii) logical error rate (LER); (iii) implicit resource efficiency (physical-to-logical qubit ratio) as a derived quantity.

Methods.

1. Formulate defect placement as a discrete optimization task with a composite objective balancing *qubit count* and *LER deviation* from target.
2. Employ **Simulated Annealing (SA)** and **Genetic Algorithms (GA)** to explore the configuration space. SA uses temperature-controlled acceptance to escape local minima; GA evolves populations via crossover/mutation constrained by code-distance-preserving rules.
3. Compare **closed** versus **partially open** holes. The latter can raise the logical-qubit count more aggressively by altering boundary-component counts but may incur higher LER.
4. Study the influence of **code distance** (via hole size and spacing) on LER for fixed logical-qubit counts.

5. Perform sensitivity analyses on optimizer hyperparameters (e.g., SA temperature schedule, GA population size/generations) to assess robustness and practical tuning guidelines.

Key Findings (preview).

- For fixed target LERs on modest lattices, both SA and GA identify layouts approaching known packing limits while respecting error constraints.
- *Partially open* holes can increase logical-qubit density by $\sim$2–3$\times$ over closed-hole layouts at comparable target LERs, albeit with fewer holes overall due to their stronger impact on error behavior.
- The attainable LER improves with increasing *effective code distance*; in defect-based layouts this requires careful spacing to preserve non-trivial cycles during any dynamic operations (e.g., braiding).
- Optimizer hyperparameters moderately affect outcomes, but improvements saturate once layouts approach topological limits set by the target distance and lattice size.

Contributions.

1. An optimization-based framework for defect placement that jointly reasons about logical-qubit density and LER targets under an erasure channel.
2. A comparative study of *closed* vs. *partially open* holes, clarifying when and how partially open defects provide superior density at acceptable reliability.
3. Quantitative guidance on the interaction between **code distance, lattice size, and target LER**, including practical spacing rules.
4. Sensitivity analysis of SA/GA hyperparameters, providing actionable defaults for rapid

exploration.

5. A discussion of modeling assumptions versus hardware realities (erasure detection overheads, decoder complexity), charting directions for more realistic noise models and integrated braid-aware layout constraints.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 reviews background on stabilizer codes, planar surface codes, logical operators and distance, and noise models. Chapter 3 formulates the optimization problem, objective functions, and the simulated-annealing/genetic-algorithm procedures under explicit code-distance constraints. Chapter 4 presents experimental results comparing closed versus partially open holes, analyzes code-distance effects, and reports hyperparameter sensitivity studies, followed by a focused discussion. Chapter 5 concludes and outlines future directions, including braid-aware distance-preservation constraints and more realistic (e.g., twirled amplitude+phase-damping) noise modeling.

Chapter 2

Background and Related Work

2.1 Quantum Computing Fundamentals

Quantum computing is a new paradigm in information processing that leverages principles of quantum mechanics. The basic unit of information is the *qubit* (quantum bit). Like a classical bit, a qubit has computational basis states $|0\rangle$ and $|1\rangle$. Unlike a classical bit, it can be prepared in a *superposition*—a coherent combination of $|0\rangle$ and $|1\rangle$—which is a key source of quantum advantage. The state of a single qubit can be visualized on the *Bloch sphere* (Figure 2.1): the north pole corresponds to the definite state $|0\rangle$ and the south pole to $|1\rangle$. Whereas a classical bit occupies only one of these two poles, a pure qubit state can lie anywhere on the sphere's surface. Mathematically, a general single-qubit state is:

$$|\psi\rangle \;=\; \alpha\,|0\rangle \;+\; \beta\,|1\rangle, \qquad |\alpha|^2 + |\beta|^2 = 1 \tag{2.1}$$

where $\alpha, \beta \in \mathbb{C}$ are *probability amplitudes*. Upon measurement in the computational basis, the outcomes 0 and 1 occur with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. This superposition enables quantum computers to explore multiple computational paths simultaneously, yielding

speedups for certain problems [9]. A convenient parameterization of pure single-qubit states

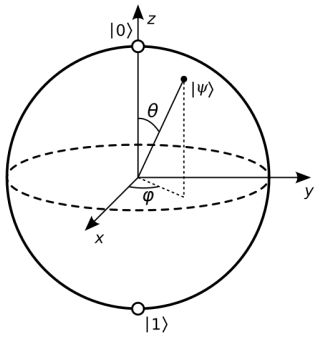

Figure 2.1: Bloch sphere representation of a single-qubit pure state.

uses Bloch-sphere angles θ and ϕ with $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{2.2}$$

On the Bloch sphere this corresponds to the point with Cartesian coordinates $(\sin\theta\cos\phi,\ \sin\theta\sin\phi,\ \cos\theta)$, where θ is the polar angle and ϕ is the azimuthal angle [9].

2.1.1 Single- and Two-Qubit Gates

Quantum computation proceeds by applying *quantum gates*, which are unitary operations that evolve qubit states reversibly and preserve total probability. Fundamental single-qubit gates include the Pauli operators:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.3}$$

implementing bit flip (X), combined bit/phase operation (Y), and phase flip (Z), respectively.

Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \equiv \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$. Then:

$$X|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = \beta|0\rangle + \alpha|1\rangle, \tag{2.4}$$

$$Y|\psi\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} -i\beta \\ i\alpha \end{pmatrix} = -i\beta|0\rangle + i\alpha|1\rangle, \tag{2.5}$$

$$Z|\psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ -\beta \end{pmatrix} = \alpha|0\rangle - \beta|1\rangle. \tag{2.6}$$

The *Hadamard* gate H creates equal superpositions:

$$H = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.7}$$

The *phase* gates apply controlled Z-rotations:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \tag{2.8}$$

A fundamental two-qubit gate shown in Figure 2.2 is the *Controlled-NOT* (CNOT), which flips the target qubit when the control is $|1\rangle$; it is key to creating entanglement—nonclassical correlations essential for quantum algorithms and error correction [10].

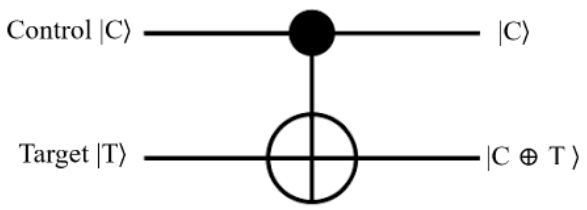

Figure 2.2: Controlled-NOT (CNOT)

2.2 Quantum Errors and Noise Models

Quantum information is inherently fragile, susceptible to noise from uncontrolled environmental coupling, imperfect controls, and device crosstalk. Any unintended interaction with the environment—such as stray magnetic fields, mechanical vibrations, or temperature fluctuations—can lead to *decoherence*, whereby information leaks to the environment and coherent quantum states become classical mixtures. Quantum systems thus face multiple error sources that corrupt quantum information, necessitating robust error-correction strategies, including mitigation of decoherence, control errors, leakage, and crosstalk.

A *quantum noise model* specifies how a physical process acts on states or computations. A standard and highly general description is the operator-sum (Kraus) representation of a completely positive trace-preserving (CPTP) map $\mathcal{E}$[11, 12]. Starting from a principal system with state ρ coupled to an environment prepared in $|e_0\rangle\langle e_0|$, a joint unitary U acts on system⊗environment and the environment is traced out[13]:

$$\mathcal{E}(\rho) = \mathrm{Tr}_{\mathrm{env}}\big[\, U\big(\rho \otimes |e_0\rangle\langle e_0|\big)U^\dagger\big] = \sum_i \langle e_i|\, U\, |e_0\rangle\, \rho\, \langle e_0|\, U^\dagger\, |e_i\rangle = \sum_i E_i\, \rho\, E_i^\dagger \tag{2.9}$$

where $\{|e_i\rangle\}$ is an orthonormal basis for the environment and the Kraus operators are $E_i = \langle e_i|U|e_0\rangle$. Trace preservation requires:

$$\sum_i E_i^\dagger E_i = I \tag{2.10}$$

which guarantees $\mathrm{Tr}\big(\mathcal{E}(\rho)\big) = \mathrm{Tr}(\rho)$.

2.2.1 Decoherence

Decoherence is the loss of phase coherence of quantum states due to unavoidable interactions with the environment. It converts delocalized superpositions into classical statistical mixtures, degrades entanglement, and corrupts computations, communications, and quantum memories. In practice, every platform—superconducting transmons, trapped ions, quantum dots, and photonics—experiences decoherence during storage, gates, and readout.

The two dominant single-qubit mechanisms are energy relaxation (*amplitude damping*) and pure dephasing (*phase damping*).

Amplitude damping (relaxation, T_1)

The amplitude-damping channel, $\mathcal{N}_{\mathrm{AD}}$, which models decoherence processes that cause energy loss (relaxation) in a qubit, acts on an arbitrary single-qubit state with density matrix ρ as:

$$\mathcal{N}_{\mathrm{AD}}(\rho) = E_0\,\rho\,E_0^\dagger + E_1\,\rho\,E_1^\dagger, \qquad E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{pmatrix}, \quad E_1 = \begin{pmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{pmatrix} \tag{2.11}$$

where E_0 and E_1 are the Kraus (error) operators of the channel, satisfying the completeness relation $E_0^\dagger E_0 + E_1^\dagger E_1 = I$. The damping probability has the time dependence:

$$\gamma(t) = 1 - e^{-t/T_1} \tag{2.12}$$

where T_1 is the experimentally determined relaxation time.

Phase damping (dephasing, T_2)

Phase damping preserves populations but randomizes relative phase (e.g., due to charge or magnetic-field fluctuations). One Kraus representation is:

$$\mathcal{N}_{\mathrm{PD}}(\rho) = E_0 \rho E_0^\dagger + E_1 \rho E_1^\dagger, \qquad E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\lambda} \end{pmatrix}, \quad E_1 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{\lambda} \end{pmatrix} \tag{2.13}$$

where λ is related to the coherence time T_2. In many devices:

$$\frac{1}{T_2} = \frac{1}{2T_1} + \frac{1}{T_\phi} \tag{2.14}$$

with T_ϕ the pure-dephasing time; consequently $T_2 \leq 2T_1$.

Pauli and depolarizing channels

It is often convenient to approximate generic noise by stochastic Pauli errors, which integrate seamlessly with stabilizer-based simulation and decoding. The single-qubit Pauli channel is:

$$\mathcal{N}_{\mathrm{P}}(\rho) = (1 - p_x - p_y - p_z)\,\rho \;+\; p_x\, X\rho X \;+\; p_y\, Y\rho Y \;+\; p_z\, Z\rho Z \tag{2.15}$$

with $p_x, p_y, p_z \geq 0$ and $p_x{+}p_y{+}p_z \leq 1$. In the symmetric case $p_x = p_y = p_z = p/3$ one obtains the depolarizing channel:

$$\mathcal{N}_{\mathrm{D}}(\rho) = (1-p)\,\rho + \frac{p}{3}\big(X\rho X + Y\rho Y + Z\rho Z\big) \tag{2.16}$$

a common worst-case surrogate. For n qubits:

$$\mathcal{N}_{\mathrm{P}}^{(n)}(\rho) = \sum_{P \in \mathcal{P}_n} p_P\, P\,\rho\, P, \qquad \mathcal{P}_n = \{I, X, Y, Z\}^{\otimes n}, \;\; \sum_P p_P = 1. \tag{2.17}$$

Although realistic channels such as combined amplitude+phase damping are not Pauli, *Pauli twirling* maps them to an effective Pauli (or depolarizing) form with parameters tied to T_1, T_2, preserving average behavior relevant to many protocols and enabling efficient classical studies and decoder benchmarking[12].

Gottesman–Knill and stabilizer simulability

The Gottesman–Knill theorem states that quantum circuits composed of Clifford unitaries, preparation of computational-basis states, and measurements of Pauli observables can be efficiently simulated on a classical computer. Because Pauli channels (and their n-qubit generalizations) apply Pauli operators stochastically, they fit naturally into this framework: error propagation remains within the stabilizer formalism, which underpins large-scale classical Monte Carlo evaluations of logical error rates for stabilizer codes[14].

2.2.2 Fault-tolerance threshold

The *threshold theorem* asserts that, for a given code family and decoding strategy, if the physical error rate is below a threshold p_{th}, then the logical error rate can be driven arbitrarily low by increasing code distance. The value of p_{th} depends on the noise model (e.g., Pauli vs. amplitude+phase damping), the error-correction circuit (with or without extraction faults), and the decoder; there is therefore no single threshold for a given code [15].

Common channel models for benchmarking. Two widely used channels are the depolarizing and erasure channels. For a qubit, the depolarizing channel can also be written as:

$$\mathcal{D}_p(\rho) = (1-p)\rho + p\,\frac{I}{2} \tag{2.18}$$

which is equivalent to (2.16). By contrast, the (flagged) *erasure channel* with probability ε replaces the qubit by an orthogonal erasure state $|e\rangle$ and records that location as erased:

$$\mathcal{E}_\varepsilon(\rho) = (1-\varepsilon)\rho \oplus \varepsilon\,|e\rangle\langle e| \tag{2.19}$$

Decoders can exploit erasure flags (known error locations), leading to higher thresholds for certain codes; for planar/surface codes with perfect syndrome extraction, the erasure threshold is approximately 50% [16], whereas depolarizing thresholds are notably lower and depend on the circuit and decoder.

Why I benchmark with the erasure channel at $p = 0.1$. I deliberately selected this noise model based on computational efficiency considerations. As described by Delfosse et al. [17], the erasure channel provides substantial computational savings compared to the depolarizing channel while still sharing many general trends and features with more complex noise models. The erasure probability of 0.1 was selected to be below the theoretical erasure threshold for surface codes under error-free syndrome extraction, which is approximately 0.5 [16], but high enough to meaningfully differentiate between different code configurations. This approach follows the strategy outlined in the Squab framework: quickly identifying promising code configurations that can later be subjected to more expensive and detailed simulations with more complex noise models.

2.3 Quantum Error Correction Codes

2.3.1 Stabilizer codes

Definition 2.1 (Stabilizer codes)**.** *A stabilizer code is a family of quantum error–correcting (QEC) codes that uses a stabilizer group S—an Abelian subgroup of the n-qubit Pauli group that does not contain $-I$—to define the codespace. The codespace is the simultaneous $+1$*

eigenspace of all elements of S:

$$\mathcal{C}(S) = \{\, |\psi\rangle : s\,|\psi\rangle = |\psi\rangle \;\forall\, s \in S \,\} \tag{2.20}$$

If S has $n-k$ independent generators, then k logical qubits are encoded into n physical qubits.

Error detection. An error E is detected when some generator $s \in S$ anticommutes with E. For any codeword $|\psi\rangle \in \mathcal{C}(S)$:

$$s\big(E|\psi\rangle\big) \;=\; -\,E\big(s|\psi\rangle\big) \;=\; -\,E|\psi\rangle \tag{2.21}$$

So $E|\psi\rangle$ is a -1 eigenstate of s, whereas valid codewords are $+1$ eigenstates. If E commutes with every element of S (i.e., $E \in N(S)$), the syndrome is trivial and the error is not detectable: when $E \in S$ it acts trivially on the codespace, and when $E \in N(S) \setminus S$ it acts as a logical operator. If all correctable errors yield distinct syndromes (patterns of stabilizer outcomes), the specific error can be identified and corrected.

Most quantum error–correcting codes are formulated as *stabilizer codes*. In quantum computing, constraints such as the no–cloning theorem and the destructive nature of measurement prevent directly copying or fully observing unknown states. Stabilizers enable indirect (syndrome) measurements that reveal errors without collapsing the encoded logical information, thereby making practical QEC possible. Figure 2.3 provides an overview of stabilizer-based quantum error correction codes. A quantum data register $|\psi\rangle_D$ is entangled with redundancy qubits $|0\rangle_R$ via an encoding operation to create a logical $|\psi\rangle_L$. The set of auxiliary qubits $|0\rangle_A$ is called the *syndrome*, and it tells us in which physical qubit the error occurred during transmission.

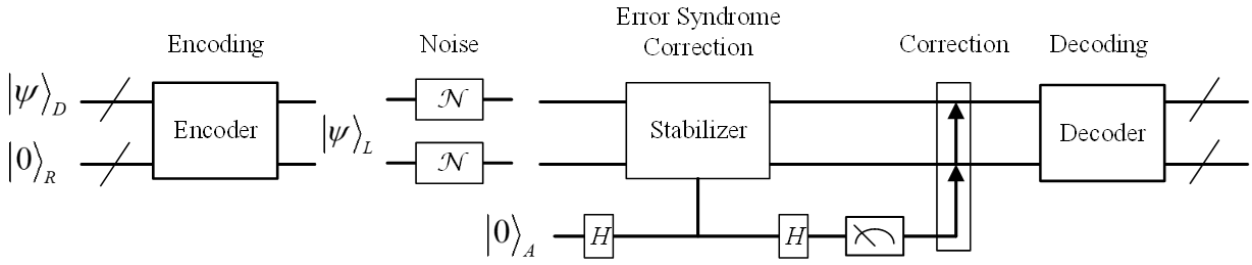

Figure 2.3: Schematic of a stabilizer quantum error-correction cycle: encoding, noisy channel $\mathcal{N}$, ancilla-assisted stabilizer measurements, classical decoding, and recovery. Adapted from [15].

2.4 Surface Code

2.4.1 Patch-based Surface Code

Surface codes are quantum error–correcting codes that place qubits on the edges of a 2D lattice and use local stabilizer measurements to protect logical qubits. The (planar) surface code introduced by Kitaev [6] is a stabilizer code: it detects errors via syndrome measurements that do not directly measure (and thus do not destroy) the encoded data qubits.

There are two types of stabilizer generators: a Z-type stabilizer that detects X errors and an X-type stabilizer that detects Z errors. In the planar surface code with a finite lattice, X stabilizers are placed at the vertices, while Z stabilizers correspond to faces (plaquettes). A planar surface code with two different boundary conditions is illustrated in Figure 2.4, where the dots represent physical qubits. The boundary of a face, marked by four blue-colored edges, defines a Z stabilizer (Z_p), and the red-colored edges depict an X stabilizer (X_v); both are defined by Equation 2.22 :

$$X_v = \prod_{i \in \text{vertex}(v)} X_i, \qquad Z_p = \prod_{i \in \text{boundary}(p)} Z_i \tag{2.22}$$

where X_v is the product of Pauli-X operators acting on the qubits and Z_p is the product of Pauli-Z operators. A vertex v and a plaquette p share either 0 or 2 common edges; hence

the operators X_v and Z_p commute.

If the lattice is $n \times m$ in size, it has nm vertical edges and $(n+1)(m+1)$ horizontal edges, totaling $2nm + n + m + 1$ physical qubits [18]. In this lattice, there are $n(m+1)$ face (Z) stabilizer operators and $(n+1)m$ vertex (X) stabilizer operators. With $2nm + n + m + 1$ qubits and $2nm + n + m$ independent stabilizer operators, the lattice has:

$$(2nm + n + m + 1) - (2nm + n + m) = 1 \tag{2.23}$$

degree of freedom, meaning that only one logical qubit may be encoded.

As mentioned above, X_v and Z_p detect Z and X errors, respectively. Quantum errors are detected by performing stabilizer measurements on groups of four qubits that form plaquettes and vertices on the lattice. These measurements can reveal whether a bit-flip or a phase-flip error has occurred on any of the qubits. The measurement outcomes form a *syndrome*, which is a classical bit string that indicates the type and location of errors. By repeating the stabilizer measurements over time, one can track the errors and correct them using appropriate decoder algorithms [19]. The number of quantum errors that can be identified and corrected is related to the minimum distance. The minimum distance in the surface code is the size of the smallest non-trivial loop on the lattice. Non-trivial loops are related to the logical operators that act on the logical qubits. In the surface code, a logical operator (a set of Pauli operators) commutes with all the stabilizers of the code while acting non-trivially on the logical qubits. The logical operators depend on the geometry and boundary conditions of the lattice. For example, in a surface code defined on a planar lattice, the logical Z operators, shown as red lines in Figure 2.5, form a subset of edges. The logical X operators, depicted as blue lines, are defined on the dual lattice, which is illustrated by the dashed lines in Figure 2.5. The distance of the surface code is the minimum size of a logical operator. In a lattice of dimensions $n \times m$ with an open boundary, the minimum distance is

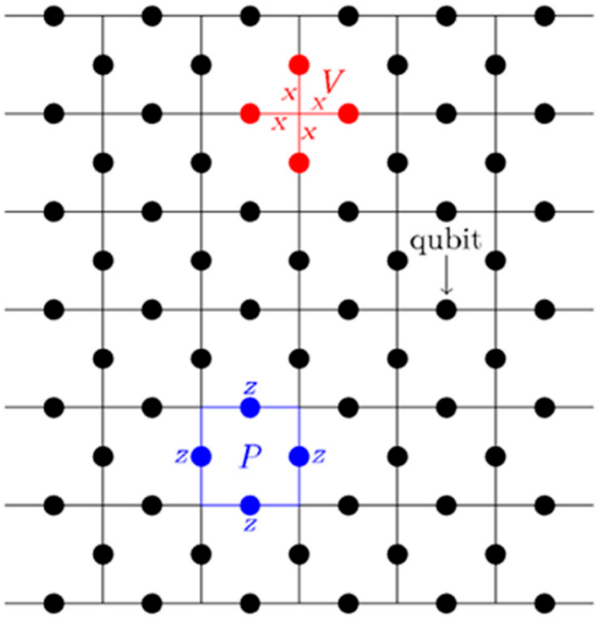

Figure 2.4: A planar surface code is illustrated. A plaquette (Z stabilizer) is marked in blue, and an X stabilizer (vertices) is marked in red. Physical qubits are located on edges (black dots).

given by:

$$d = \min\{\, n+1,\, m+1 \,\} \tag{2.24}$$

The code is capable of protecting against $(d-1)/2$ errors.

2.4.2 Defect-based Surface Code

In the surface code, a *defect* is a deliberately created hole in the stabilizer lattice: one stops measuring a chosen set of stabilizers so that an internal boundary appears. This introduces additional degrees of freedom that can encode a logical qubit. As noted by Fowler et al. [7], the literature commonly refers to these holes as "defects," created by turning off internal measure-X and/or measure-Z qubits (i.e., not running their CNOT-and-measurement cycles).

There are two types of defects: Z-cut and X-cut. Z-cut (smooth) defects arise by turning

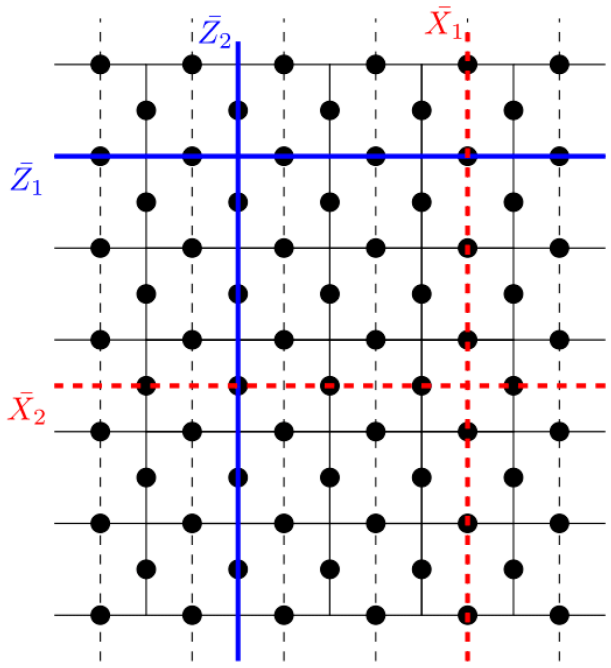

Figure 2.5: A planar lattice is shown in solid black lines, and its dual is depicted in dashed black lines. Logical operators Z (blue solid lines) and X (dotted red lines) for two logical qubits are illustrated as non-trivial cycles on the primal and dual lattices, respectively [20].

off measure-Z stabilizers, creating a hole whose internal boundary is X-type. A single Z-cut hole near an X boundary defines the logical operators X_L (a chain of X operators from the outer X boundary to the hole's internal X boundary) and Z_L (a loop of Z operators encircling the hole). Both commute with all remaining stabilizers, but anticommute with each other because the chain and loop share exactly one data qubit. Hence the hole encodes a single logical qubit (with $X_L^2 = Z_L^2 = I$ and $Y_L = Z_L X_L$). This construction requires at least one X boundary; with only Z boundaries, Z_L would be fixed and no new qubit appears. The distance is $d = 3$ (or $d = 4$ if the hole is one cell farther from the edge) and cannot exceed 4 without enlarging the hole. In Ref. [7], a single Z-cut qubit is also called a smooth/dual defect.

Figure 2.6 shows how a single Z-cut logical qubit is formed by turning off one measure-Z stabilizer to make a small hole placed near an outer X boundary. Turning that stabilizer off creates an internal X boundary around the hole and introduces two degrees of freedom [7].

The logical X operator X_L is a short chain of Xs, specifically $X_1X_2X_3$, that runs from the outer X boundary to the hole's internal X boundary. Because the chain's Xs are paired across each Z stabilizer, it commutes with all Z stabilizers (and trivially with the X ones) [7]. The logical Z operator Z_L is a loop of Zs encircling the hole, $Z_3Z_4Z_5Z_6$. The loop's Zs are paired across each X stabilizer, so it commutes with all X stabilizers and—crucially—because the Z-stabilizer inside the loop was turned off to make the hole, this loop is no longer fixed by a stabilizer and can act nontrivially on the code state [7].

The figure highlights that the chain and loop share exactly one data qubit (qubit 3), so they anticommute:

$$X_L Z_L = (X_1X_2X_3)\,(Z_3Z_4Z_5Z_6) \;=\; -\,Z_L X_L, \tag{2.25}$$

This establishes that the hole encodes a logical qubit (with $X_L^2 = Z_L^2 = I$ and $Y_L = Z_L X_L$) [7]. Figure 2.6 also conveys two design constraints: (i) you must have at least one outer X boundary (with only Z boundaries, Z_L would be fixed and no new qubit appears), and (ii) the distance is $d = 3$ for the depicted placement; moving the hole one cell farther from the edge gives $d = 4$, but you cannot exceed 4 without enlarging the hole [7].

A closely related construction yields a single X-cut qubit: turn off a measure-X qubit in an array that includes at least one Z boundary. The resulting logical operators are defined by a Z_L chain running from the outer Z boundary to the newly formed internal Z boundary, and an X_L loop encircling the X-cut hole. These operators obey the required anticommutation relations. In [7], a single X-cut qubit is commonly called a *rough* or *primal* defect.

A *double Z-cut* qubit is formed by turning off two measure-Z stabilizers in the surface-code lattice, is shown in Figure 2.7. This creates two Z-cut holes (defects), each surrounded by measure-X qubits and thus endowed with an internal X-type boundary. The two holes introduce four additional degrees of freedom. Each hole can be manipulated independently

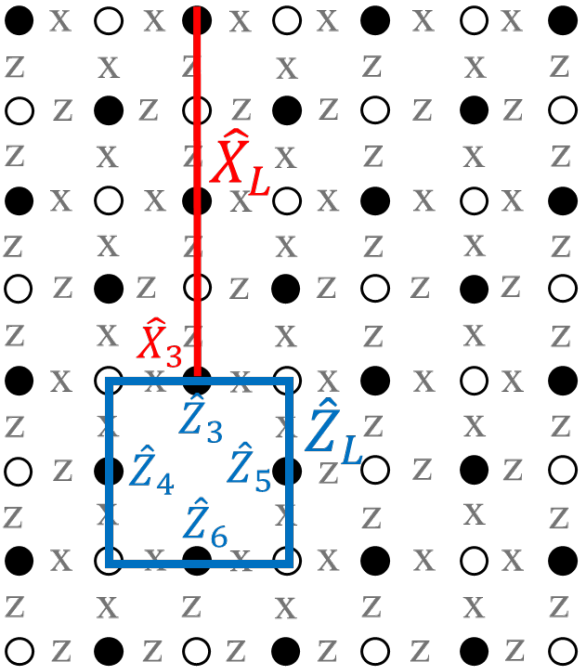

Figure 2.6: Defect-based surface code with a Z-cut: data qubits are shown as solid circles and measurement qubits as open circles. Two logical operators are indicated: $X_L = X_1X_2X_3$, running from the array's outer X boundary to the hole's internal X boundary, and $Z_L = Z_3Z_4Z_5Z_6$, forming a loop around the Z-cut hole. Note that the operator chains for X_L and Z_L share one physical data qubit (qubit 3) [7].

by defining $(\hat{X}_{L1}, \hat{Z}_{L1})$ for the upper hole and $(\hat{X}_{L2}, \hat{Z}_{L2})$ for the lower hole, in direct analogy with the single Z-cut construction. These four linearly independent operators span all four degrees of freedom: within each pair $(\hat{X}_{Lj}, \hat{Z}_{Lj})$ the operators anticommute, while operators from different holes commute.

To manipulate the two holes in a correlated fashion, replace $\hat{X}_{L1}$ with the product $\hat{X}_{L1}\hat{X}_{L2}$. This composite operator anticommutes with both $\hat{Z}_{L1}$ and $\hat{Z}_{L2}$ and, together with $\hat{X}_{L2}$, provides functionality equivalent to $\{\hat{X}_{L1}, \hat{X}_{L2}\}$ (since $\hat{X}_{L2}^2 = \hat{I}$ implies $\hat{X}_{L1} = (\hat{X}_{L1}\hat{X}_{L2})\hat{X}_{L2}$). Moreover, by multiplying $\hat{X}_{L1}\hat{X}_{L2}$ by the product of the enclosed X-type stabilizers (which changes only an overall sign), we obtain an equivalent *linked* operator $\hat{X}_L$ that directly

connects the internal X-boundaries of the two holes (solid red line in Figiure 2.7). Thus:

$$\hat{X}_L \equiv \hat{X}_{L1}\hat{X}_{L2} \quad \text{(up to stabilizers and a global sign).} \tag{2.26}$$

Then a convenient operator set is $\{\hat{X}_L, \hat{X}_{L2}, \hat{Z}_{L1}, \hat{Z}_{L2}\}$. The logical X-type operators commute with one another, as do the logical Z-type operators. The linked operator $\hat{X}_L$ anticommutes with both $\hat{Z}_{L1}$ and $\hat{Z}_{L2}$ (each shares an odd overlap with $\hat{X}_L$), while $\hat{X}_{L2}$ anticommutes only with $\hat{Z}_{L2}$.

If we wish to encode a *single* logical qubit with the pair of holes, we restrict to two of the four degrees of freedom—for example, choose:

$$\hat{X}_L, \qquad \hat{Z}_{L2} \equiv \hat{Z}_L, \qquad \hat{Y}_L = \hat{Z}_L\hat{X}_L \tag{2.27}$$

which obey $\hat{X}_L^2 = \hat{Z}_L^2 = \hat{I}$ and $\hat{X}_L\hat{Z}_L = -\hat{Z}_L\hat{X}_L$. Under this restriction, operations act on the two holes in a correlated manner, effectively forming a single two-level logical qubit with state$\alpha\,|g_L\rangle + \beta\,|e_L\rangle$.

A double X-cut qubit is constructed analogously to the double Z-cut case by turning off two measure-X qubits, thereby creating two X-cut holes.

2.4.3 Decoder

A *decoder* uses measured syndrome data to infer and correct physical errors so that the encoded logical state is restored. It is a cornerstone of any error-correction stack: decoding must be accurate and fast enough that errors do not accumulate while the correction is being computed, and it should be implementable with modest hardware resources to enable scaling. Two standard metrics characterize decoder performance. (i) The *threshold* is the **maximum** physical error rate p_{th} below which increasing the code distance d (i.e., using more physical

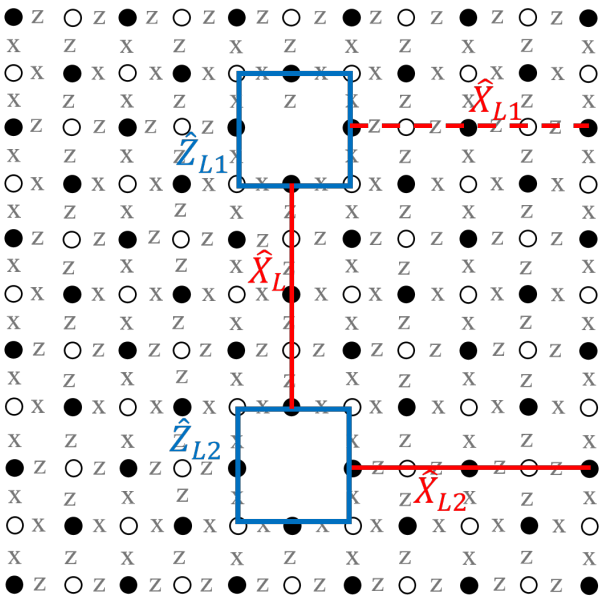

Figure 2.7: Double Z-cut logical qubit created by disabling two measure-Z qubits. For each Z-cut hole, logical operators $\hat{X}_{L1}, \hat{Z}_{L1}$ (upper) and $\hat{X}_{L2}, \hat{Z}_{L2}$ (lower) are defined: $\hat{X}_{L1}$ and $\hat{X}_{L2}$ run from the array's outer X boundary (right) to the holes' internal X boundaries, while $\hat{Z}_{L1}$ and $\hat{Z}_{L2}$ form loops encircling the holes. The dashed red path ($\hat{X}_{L1}$) may be replaced by the product $\hat{X}_{L1}\hat{X}_{L2}$, which flips both holes and, together with $\hat{X}_{L2}$, is equivalent to using $\hat{X}_{L1}$ and $\hat{X}_{L2}$ separately. Multiplying $\hat{X}_{L1}\hat{X}_{L2}$ by the outlined X-type stabilizers (changing only a global sign) yields the linked operator $\hat{X}_L$, shown as the solid blue path, which directly connects the two holes[7].

qubits per logical qubit) *reduces* the logical error rate. Thresholds depend on the code family, the noise model, and the decoding method, and are established theoretically or via numerical simulation. (ii) *Runtime/latency*: decoding must complete within (or faster than) the QEC cycle time so that corrections can be applied (or tracked in software) without stalling the quantum program. Practical assessments therefore consider algorithmic time complexity versus system size, constant factors, parallelism, and hardware acceleration, not just asymptotics.

A variety of methods are used for topological codes, including classical graph-based algorithms, renormalization approaches, machine-learning decoders, maximum-likelihood variants, and circuit-level decoders that explicitly model faults in syndrome extraction. Representative examples for the surface code include Minimum-Weight Perfect Matching (MWPM)[21],

neural-network decoders[19], renormalization-group (RG) decoders[22], maximum-likelihood decoders (MLD)[23], circuit-level decoders[24], and transformer-based neural network decoders [25]. Each offers a different trade-off between accuracy, robustness to realistic noise, and implementation complexity (CPU/GPU/FPGA/ASIC).

Chapter 3

Proposed Method

In surface codes, a standard planar lattice with open outer boundaries and no holes encodes a single logical qubit. Introducing holes (defects) into the lattice can increase the number of encoded logical qubits and reduce physical-qubit overhead; however, this gain comes with a trade-off: the logical error rate typically rises because the code distance decreases. Before addressing how to balance qubit density and error protection, I first explain how to determine the number of logical qubits in defect-based surface codes with closed and open boundaries.

3.1 Determining the Number of Logical Qubits with Holes (defects)

In this section, I provide more details about the planar lattice and its dual lattice concepts to define the general surface code with open and closed boundaries. In the paper [26], a compact 2-manifold S is cellulated to create a surface code. The surface code is represented as a triple (V, E, F), and within this triple is a finite graph known as $G = (V, E)$, where each connected component of $S \backslash G$ is a disk corresponding to a face $f \in F$ of the cellulation. A face is considered a subset of $f \subset E$ because it is associated with the collection of edges on its

boundary. If an edge of G is incident to just one face of G, it is defined as a boundary edge. In the generalized surface code, each edge on the boundary can be either open or closed. A face with a boundary edge is called a boundary face, denoted as ∂F. The boundary vertices, ∂V, are the endpoints of boundary edges, ∂E. The indices C and O stand for non-open (closed) and open components of the surface code. For instance, the sets of non-open (closed) vertices, edges, and faces are denoted by V_C, E_C, and F_C, respectively. The number of connected components of G containing no closed boundary edge is denoted by $\kappa_{\overline{\partial_C E}}(G)$, and the number of connected components of G containing no open vertices is denoted by $\kappa_{\overline{\partial_O V}}(G)$.

In Ref. [26], the dual surface code $G^* = (V^*, E^*, F^*)$ with open and closed boundaries is defined from $G = (V, E, F)$ by replacing each face $f \in F$ of G with a non-open vertex $V \setminus \partial V \in V^*$. Each closed edge, $e \in \partial_C E$, of G is substituted with an open vertex, $\partial_O V^*$. Non-open edges, $E \setminus \partial E \in E^*$, are obtained by connecting two vertices of G^* if they correspond to distinct faces of G that share an edge. In addition, each edge, $\partial_O E \in E^*$, is acquired by connecting $V \setminus \partial V \in V^*$ and $\partial_O V^*$ if $V \setminus \partial V \in V^*$ corresponds to $f \in F$ that is the unique face of G containing the boundary edge e. Open edges, $\partial_O E^*$, are defined for every pair of distinct closed boundary edges, $\partial_C E$, sharing a closed boundary vertex, $v \in \partial_C V$, in G. A face from the cycle F_v in G^* for each $v \in V \setminus \partial V$ is defined as a non-open face, and a face from the cycle $\overline{F}_v \in \partial_O F^*$ for each $v \in \partial_C V$ is defined as an open face.

It can be concluded that the following bijection exists between the surface code $G = (V, E, F)$ and its duality, $G^* = (V^*, E^*, F^*)$:

$$\begin{aligned} &F \leftrightarrow V^* \setminus \partial V^*, \qquad \partial_C E \leftrightarrow \partial_O V^*, \qquad E \setminus \partial E \leftrightarrow E^* \setminus \partial E^*, \\ &\partial_C V \leftrightarrow \partial_O E^*, \qquad V \setminus \partial V \leftrightarrow F^* \setminus \partial F^*, \qquad \partial_C V \leftrightarrow \partial_O F^*. \end{aligned} \tag{3.1}$$

The generalized surface code $G = (V, E, F)$ with a block size n, representing the number of

closed edges, $E \backslash \partial_O E$ (i.e., the number of physical qubits); k, the number of logical qubits encoded; and d, the minimum distance, is indicated as $[n, k, d]$. The following formula determines the number of logical qubits:

$$k = -|V_C| + |E_C| - |F| + \kappa_{\overline{\partial_C E}}(G) + \kappa_{\overline{\partial_O V}}(G) \tag{3.2}$$

The minimum code distance is the minimum length of a non-trivial relative cycle of G and G^*. The non-trivial relative cycle of G that is denoted as d_Z is the logical Pauli Z operator. Moreover, the logical Pauli X operator, d_X, is a non-trivial relative cycle of G^*.

Figure 3.1 shows a surface code consisting of two closed holes (Ref. [26] considered the external boundary as a closed hole), one open hole, and one partially open hole. The number of closed edges in the surface code is 203, the number of closed vertices is 108, and the total number of faces is 91. Therefore, the number of logical qubits encoded is 4 by using Equation 3.2.

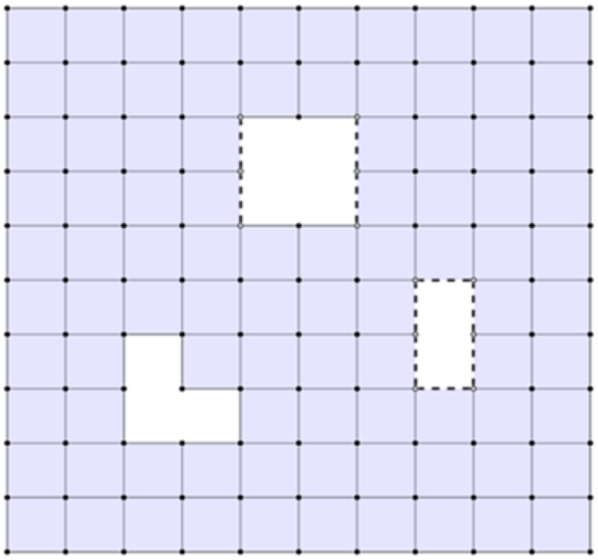

Figure 3.1: A surface code with four holes encodes four logical qubits. The dotted lines represent open edges, while the solid lines indicate closed edges.

3.1.1 Maximum Possible Number of Logical Qubits with Closed Holes on a Square Lattice Surface Code of Dimensions n by n

In [26], the method for calculating the minimum distance of a code in a square lattice surface code punctured with square holes is discussed. Each qubit corresponds to a closed hole of size $t \times t$, and these holes are separated from each other and from the boundary by $4t - 1$. This separation ensures that the minimum distance of the code is $d = 4t$. This lattice is depicted in Figure 3.2. For instance, if a hole is 1×1 and the separation between holes and from the boundary is three, then the minimum distance of the code is four. If the distance between holes and hole and boundary are less than $4t-1$, then the minimum distance is less than $4t$ and is equal to this distance. As shown in Figure 3.3, if the holes are separated from each other and from the boundary by two, then the minimum distance of the code is three. The number of encoded qubits is determined by the number of closed holes. Therefore, the number of logical qubits achievable within a tile depends on the minimum distance of the code and the size of the hole. It can be said that if the minimum code distance is three, a hole consisting of one face is the optimal choice for maximizing space to create additional logical qubits.

To find the maximum number of holes that can be placed in a lattice while maintaining the minimum code distance of three, we consider a 3×3 square around each hole. Additionally, each hole should be at least two faces away from the boundary, which means that each 3×3 square should be placed within an $(n - 2) \times (n - 2)$ area, as shown in Figure 3.3. Assuming that the minimum code distance is three and the closed hole consists of one face, the maximum number of logical qubits can be calculated using the following equation:

$$\text{maximum number of logical qubits} = \left\lfloor \frac{n-2}{3} \right\rfloor \times \left\lfloor \frac{n-2}{3} \right\rfloor \tag{3.3}$$

This formula can be used in any size of lattice. Additionally, if two boundaries of the lattice consist of open edges, one more logical qubit is added to Equation 3.3.

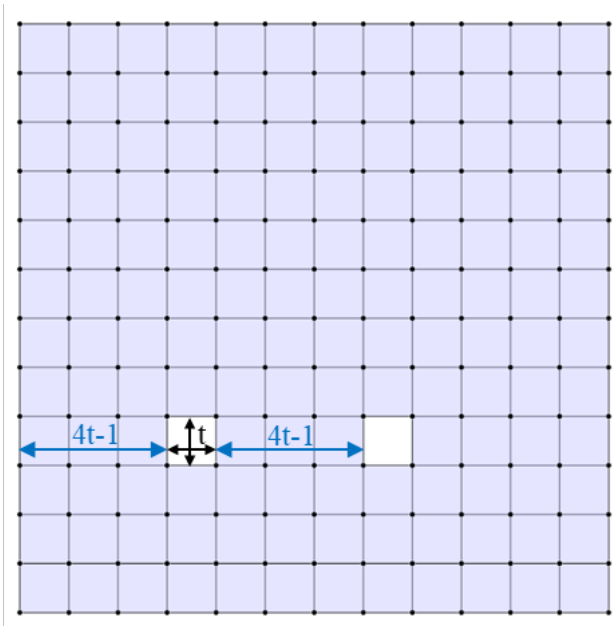

Figure 3.2: A surface code with 2 closed holes of size $t \times t$ encoding 2 logical qubits.

3.1.2 The Number of Logical Qubits with Partially Open Holes on a Square Lattice Surface Code of Dimensions n by n

If a hole contains both open and closed edges, it is termed a partially open hole or mixed hole. As noted previously, a closed hole of any size increases the number of logical qubits by one. Therefore, the number of closed holes is directly proportional to the number of encoded qubits. In contrast, a partially open hole increases the count by more than one logical qubit. Consider a square lattice surface code punctured with a square hole having two open boundaries on each side; according to Equation 3.2, this results in two logical qubits. When two boundaries are open, the number of closed vertices decreases more compared to closed edges, thereby increasing the number of logical qubits, as determined by Equation 3.2. For instance, a square lattice $n \times n$ with a closed edge and closed boundary has one logical qubit because the lattice comprises $n^2 - 1$ faces, $(n + 1) \times (n + 1)$ closed vertices, and $2n(n + 1)$ closed edges. Furthermore, the number of connected components of G containing no closed

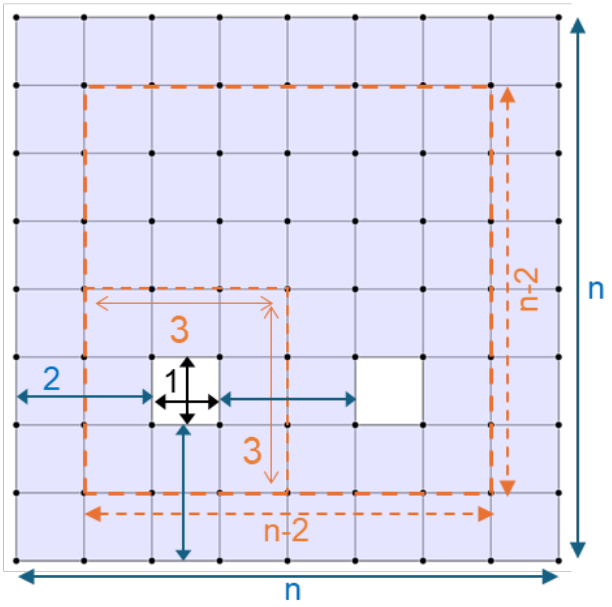

Figure 3.3: A surface code with two closed holes and a minimum code distance of 3.

boundary edge is zero, and the number of connected components of G containing no open vertices is one. Therefore, the number of logical qubits is as follows:

$$\begin{aligned} k &= -|V_C| + |E_C| - |F| + \kappa_{\overline{\partial_C E}}(G) + \kappa_{\overline{\partial_O V}}(G) \\ k &= -(n+1)^2 + 2n^2 + 2n - n^2 + 1 + 1 \\ k &= -n^2 - 2n - 1 + 2n^2 + 2n - n^2 + 1 + 1 = 1 \end{aligned} \tag{3.4}$$

If two boundaries of the hole are opened, as illustrated in Figure 3.4, the surface code consists of n^2-1 faces, $2n^2+2n-3$ closed vertices, $2n^2+2n-2$ closed edges, and $\kappa_{\overline{\partial_C E}}(G) = \kappa_{\overline{\partial_O V}}(G) = 0$, resulting in the following count of logical qubits:

$$k = -n^2 - 2n + 3 + 2n^2 + 2n - 2 - n^2 + 1 = 2 \tag{3.5}$$

When two sides of a square hole are opened, this results in an increase in the number of open vertices by two compared to the number of open edges. Therefore, it can be concluded that two logical qubits are added when there is only one partially open hole. The first square

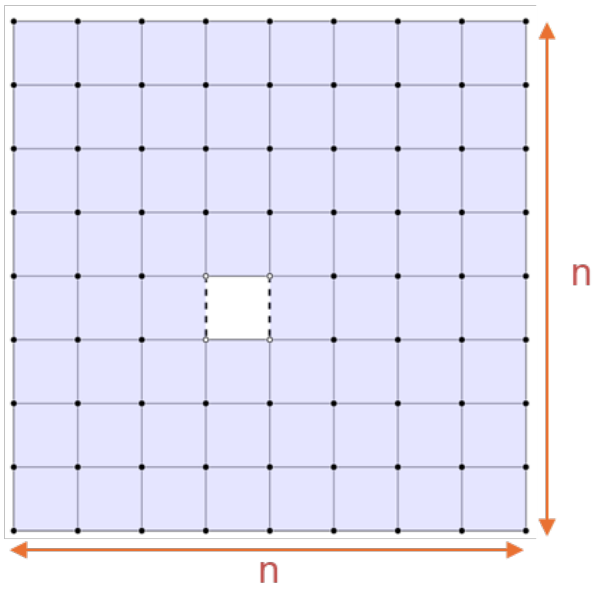

Figure 3.4: A surface code with a partially open hole.

partially open hole contributes two logical qubits, and each additional square partially open hole adds three more logical qubits, due to the reduction of one face, four closed vertices, and two closed edges. Consequently, the number of logical qubits with h partially open holes on a square lattice surface code of dimensions $n \times m$ is as follows:

$$k = 2 + 3 \times h \tag{3.6}$$

If the number of faces in a hole is more than one, it yields the same result as a hole with one face because, as the number of faces decreases, the number of edges also decreases. Thus, a surface code with partially open holes leads to a substantial increase in the number of logical qubits.

3.1.3 Preserving Code Distance During Braiding

Logical operations in defect-based surface codes are often performed by braiding, which involves moving defects (holes) around each other to enact entangling gates such as CNOT [7, 27]. One key advantage of this approach is that it does not require ancillary qubit patches as

in patch-based lattice surgery. However, the movement of defects can dynamically alter their spacing and thus change the effective code distance. If defects are brought too close together during braiding, the minimum distance of logical operators may shrink, increasing the risk of logical errors [7, 27]. Brown et al. [27] describe conceptual frameworks for preserving code distance during logical operations via careful control of defect trajectories and separation.

3.2 Proposed Method

As noted above, the efficiency gained by introducing holes comes with a trade-off: shrinking code distance increases the logical error rate. To balance maximizing the number of encoded logical qubits with maintaining acceptable error-correction performance, I cast hole placement as a knapsack optimization problem [28].

The Knapsack problem is well-known in computer science and mathematics for its optimization challenge, particularly in combinatorial optimization. It has applications across various domains, like finance, logistics, and resource allocation, and serves as a fundamental problem in algorithm design and optimization studies. Finding an optimal solution is deemed NP-hard since there is no known polynomial-time algorithm for solving it, especially for large instances; however, several algorithms offer approximate solutions, such as SA (simulated annealing), genetic programming, and various heuristics suitable for real-world applications. In my scenario, I look at the surface code as a knapsack: The knapsack items symbolize the holes, I aim to introduce them in the code structure, and the weights of knapsack items represent the logical error rate that arises from presenting holes in the code. If the inset of holes outweighs the desired logical error rate, we need to find another set of holes. Unlike traditional Knapsack problems, though, expressing the relationship between hole count/placement and logical error rate is challenging. Therefore, I rely on an empirical assessment to determine the impact of the holes and their placement on the logical error rate, a task facilitated by the Squab framework [17].

The core idea of our method is illustrated in Figure 3.5. It employs a discrete optimization algorithm and the Squab framework. The discrete optimization algorithm predicts the best positioning of holes in the surface code, considering a specified logical error rate and selected constraints, which will be detailed later in this paper. The optimization algorithm interacts with the Squab process, which acts as an environment offering feedback through a logical error rate. If the error rate threshold is unmet, depending on the nature of the optimization algorithm, it will seek another subset of holes that meet the requirements for logical errors. Through an iterative process, our method explores potential solutions to maximize qubit numbers based on specific logical error demands. Another viable approach to improve code

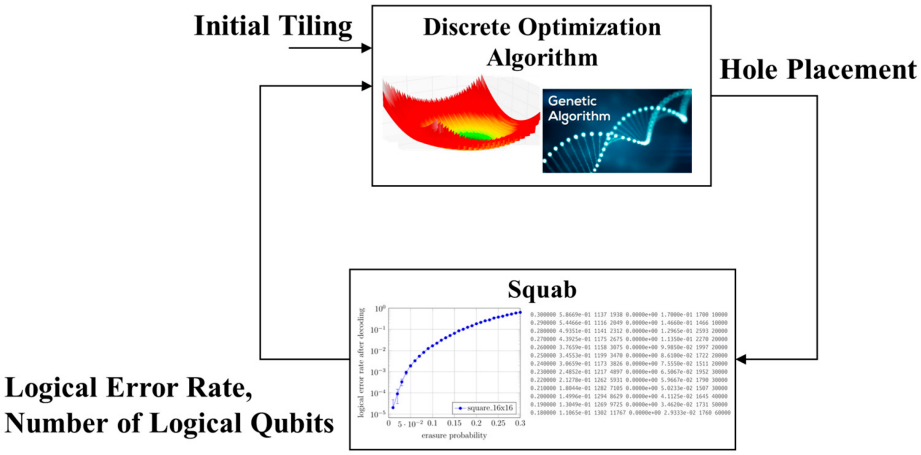

Figure 3.5: Proposed methodology

density is patch-based or lattice concatenation. When surface codes are arranged adjacently, multiple codes can be combined to form a larger quantum computation [29, 30]. However, incorporating holes in a surface code to encode more logical qubits in a lattice proves advantageous, enhancing the overall efficiency of the quantum computation process by reducing resource overhead, particularly in terms of fewer physical qubits, and optimizing error correction mechanisms. The strategic use of holes in the lattice allows for a more streamlined and resource-efficient implementation of large-scale quantum computing.

3.2.1 Maximizing the Number of Logical Qubits Using Optimization Algorithms

Expressing the loss function to determine the maximum number of logical qubits within a given logical error is difficult due to the intricate connection between the hole placement and the rate of logical errors. Therefore, I rely on approximate algorithms to discover the optimal placement of holes in the lattice for a given logical error rate. Approximation algorithms play a crucial role in solving optimization problems, especially when finding an exact solution is computationally infeasible or too time-consuming. These algorithms provide solutions that are close to optimal solutions, often within a provable bound, and are widely used in real-world applications where efficiency is a primary concern.

In the literature, a plethora of approximation algorithms exists to find near-optimal solutions for a given problem. Among the existing approaches, dynamic programming emerges as one of the widely embraced algorithms to tackle problems resembling Knapsack. However, when it comes to our problem, where I encounter a Knapsack-like situation, dynamic programming proves to be unviable. This is because the task of finding the perfect placement of holes cannot be simplified to the act of locating holes solely within subsections of the lattice. Simply concatenating these subsections in a naive manner within the lattice can lead to an undesirable increase in logical errors and significantly decrease the number of encoded logical qubits, as opposed to the effectiveness of locating holes across the entirety of the lattice.

Simulated Annealing

As an alternative to dynamic programming, we employ the simulated annealing (SA) approach [18] to find the optimal placement of holes in the lattice. SA is a probabilistic optimization technique inspired by the annealing process in metallurgy. It is used to find near-optimal solutions in complex search spaces by iteratively exploring the solution space. The temperature in SA plays a critical role in determining the algorithm's behavior. The

algorithm is more permissive at high temperatures, allowing it to explore a wide range of solutions, including those that may initially seem suboptimal. As the temperature decreases, the algorithm becomes more selective. The probability of accepting a worse solution is determined by the energy function and the current temperature, where solutions with smaller energy are optimal, and vice versa.

Its robustness and local search refinement make the SA a promising approach to find the positions of holes that yield the maximal number of logical qubits. In addition,its simplicity allows us to efficiently analyze even larger lattices. To adapt the SA to our requirements, we implemented several modifications. In my scenario, the solution space encompasses all potential arrangements of holes in a square lattice. The SA process commences with an open-boundary square lattice and a randomly selected hole. To generate neighboring solutions, I employ three distinct actions that are randomly chosen. The first action adds a new hole into the existing lattice in a random fashion. However, a hole must adhere to the selected code distance to be considered a viable candidate for generating neighboring solutions. Specifically, for a code distance of d, the holes in the lattice should be at least $d-1$ faces apart from themselves and the border of the lattice. The addition of holes can lead to poor error performance of the code.

Consequently, I have defined a second action: removing holes from the current solution. While the previous two actions alter the number of holes, the third action involves reassigning holes within the existing solution across the tiles to find a placement with a lower logical error rate. To evaluate potential solutions, I defined the following energy function:

$$E = e^{-N_Q} + E_{\text{error}} \tag{3.7}$$

where N_Q represents the number of encoded logical qubits, and:

$$E_{\text{error}} = \begin{cases} \alpha\,(\text{LER} - T_{\text{LER}}), & |\text{LER} - T_{\text{LER}}| < \varepsilon \\ \beta\,(\text{LER} - T_{\text{LER}}), & \text{otherwise} \end{cases} \tag{3.8}$$

with $\beta \gg \alpha$. Here, LER represents the obtained logical error rate for the current solution; T_{LER} represents the desired logical error rate for which I maximize the number of logical qubits; and α, β, and ε are real numbers. With the term e^{-N_Q} in Equation 3.7, the SA favors solutions that encode more logical qubits; however, with $\beta \gg \alpha$ in Equation 3.8, the SA penalizes solutions that deviate from the desired logical error.

Genetic Algorithms

In addition to SA, I employed genetic algorithms (GAs) [31]. GAs are heuristic optimization techniques inspired by natural selection. The GA iteratively evolves the population over generations. It starts with an initial population, evaluates each individual's fitness using the fitness function, and then uses selection mechanisms to choose individuals for reproduction, favoring those with higher fitness scores. The GA begins with lattices with one randomly selected hole and adds more through crossover and mutation. Crossover and mutation operators are applied to create a new population, which replaces the old one. This process continues until a termination condition, such as a maximum number of generations or a satisfactory fitness level, is met. During the crossover operation, I exchange two holes between the parent tiles. In the mutation operation, I introduce changes in a tile by adding or removing holes, resulting in a diversified solution space. As with the SA method, inserting new holes in the lattice must adhere to the code distance criterion. As I tackle the single-objective optimization problem, I employ classical steady-state criteria for selecting parents (SSGA) for its simplicity. The other widely used GA, NSGA-II, aims to solve multi-objective optimization problems, which is not the case in our paper. I use the following fitness func-

tion to quantify the quality of solutions in the iteration or generation of the single-objective optimization with GA:

$$E = N_Q + E_{\text{error}} \tag{3.9}$$

where:

$$E_{\text{error}} = \begin{cases} \alpha\,(\text{LER} - T_{\text{LER}}), & |\text{LER} - T_{\text{LER}}| < \varepsilon \\ \beta\,(\text{LER} - T_{\text{LER}}), & \text{otherwise} \end{cases} \tag{3.10}$$

and where LER represents the obtained logical error rate for the current solution; T_{LER} represents the desired logical error rate for which I am maximizing the number of logical qubits; and α, β, and ε are real numbers, where $\alpha \gg \beta$. The term N_Q in Equation 3.9 enables the GA to favor individuals (surface codes) that encode more logical qubits. Moreover, with $\alpha \gg \beta$ in Equation 3.10, the GA rewards those individuals whose logical error rates are close to the desired value.

Squab

Squab (version 1.2) is software capable of constructing and displaying various lattice tilings and running fast Monte Carlo simulations to estimate error thresholds and logical error rates for different architectures. Several tools are available for simulating and decoding surface codes, including Stim and Qsurface. However, most of these tools lack native support for creating holes in the codes. Even when adding this functionality to a library is possible, the implementation can be complex. Squab is specifically designed for general 2D surface codes that simplify creating holes with open and closed boundaries in any shape. It was developed by Nicolas Delfosse et al. [17]. Squab utilizes the erasure channel, achieving greater computational savings compared to the depolarizing channel. It operates on the principle that the main difference between an erased and a depolarized qubit is the known location of

the erasure. Squab assumes the decoding algorithm is aware of the erased qubits' locations, which are typically measurable in experimental setups. Decoding stabilizer codes in this channel requires solving a linear system with generally cubic complexity. For generalized surface codes, Squab achieved a linear time benchmark by employing induced homology [17]. The software estimates errors by determining the correctability of an erasure for the surface code associated with surface G, its dual G^*, and an erasure error. Squab supports any 2D planar lattice in any shape, with or without holes, and with open or closed boundaries. Additionally, it generates a performance report detailing the surface code's features, the distribution of necessary measurements, and plots of the logical error rate after decoding X-error, Z-error, and both errors simultaneously.

As previously indicated, Squab provides feedback in the form of a logical error rate and the number of logical qubits encoded. It takes hole placements as input, generates a lattice with holes, and outputs code properties such as the logical error rate and the number of encoded logical qubits computed based on Equation 3.2. In the experiments that were conducted, we considered the logical error rate when the surface code was subjected to an erasure channel with an erasure probability of $p = 0.1$. The selected probability is lower than the error threshold in the surface code under error-free syndrome extraction. Furthermore, in the Squab, I performed 10,000 trials to assess the error of the designed quantum error correction code.

It is important to acknowledge that Squab's erasure channel model assumes immediate and perfect knowledge of erasure locations, thus representing an idealization compared to physical hardware implementations. In real quantum systems, detecting qubit erasures (such as leakage events) requires additional measurement circuits and resources. These detection mechanisms would introduce overhead in the form of time delays, additional qubits, and potential false positives/negatives in erasure detection. While my current benchmarking approach does not model these practical complexities, it provides valuable theoretical bounds

on the performance of different hole configurations. The comparison between different surface code layouts remains valid within this theoretical framework, as all configurations are evaluated under the same idealized conditions.

Chapter 4

Results

4.1 Maximizing Number of Qubits for a Given Logical Error Rate

In the first set of experiments, I assessed the maximal encoding density of square surface codes. Here, I fix the code distance of the surface code to three. Each experiment was repeated five times to ensure statistical accuracy, and the results were expressed with the mean and standard deviation. For the SA approach, I selected the initial temperature of 2400 K, which exponentially decreases to 0.5 K across 1000 iterations to cover the entire search space. At each temperature, the acceptance ratio is determined by Boltzmann probability, where I set the Boltzmann constant to one. In GA [31], I set the size of the population to 200 and the number of generations to 1000. To produce offspring in each generation, 50 parents are selected to produce offspring in each generation using the steady-state criterion. The genome length equals the maximal number of logical qubits (Equation 3.3), so I can represent all the potential holes of the lattice. The initial population represents one randomly selected hole, and subsequent holes are introduced through crossover and mutation. Each crossover

is followed by a mutation of the genome, which leads to diverse solutions. I observed the performance of my approach for a predefined set of logical errors (10^{-4}, 10^{-3}, 5×10^{-3}, 10^{-2}). In addition, I also assessed the behavior of the proposed solution for different sizes of the square lattice.

The results are presented in Figures 4.1 (a) and (b) for SA and GA, respectively. The y-axis represents the number of obtained logical qubits for the evaluated logical error rates on the x-axis per given tile size. As expected, I observe that I can encode more logical qubits with the increase in the logical error rate. Moreover, the tile size is crucial in obtaining more logical qubits. As discussed previously, smaller tiles yield fewer logical qubits due to constraints imposed by code distance. On the other hand, larger tiles enable the encoding of a higher number of logical qubits. Moreover, I can see that several tiles exhibit similar results for specific logical error rates. For example, all tiles exhibit a similar number of logical qubits for lower error rates, suggesting that I can employ smaller sizes and get the same number of logical qubits. By comparing results obtained from SA and GA, I can see that both methods deliver similar results, indicating the upper limits of the achieved number of logical qubits by inserting holes in the surface code lattice.

4.2 Evaluating Surface Codes with Partially Open Edges

In the previous section, experiments were conducted on holes with closed boundaries to determine the maximum number of qubits for a given logical error rate. This time, I investigate how holes with partially open edges affect the encoding density of surface code for a given logical error rate. Holes with partially open edges or partially open holes refer to holes with open, non-adjacent boundary edges. As demonstrated in Subsection 3.1.2, partially open holes can encode more logical qubits in a square lattice. Therefore, square lattices with partially open holes can encode more logical qubits compared to lattices containing closed holes of similar code distance values. I set the code distance to 3 and assess the maximum

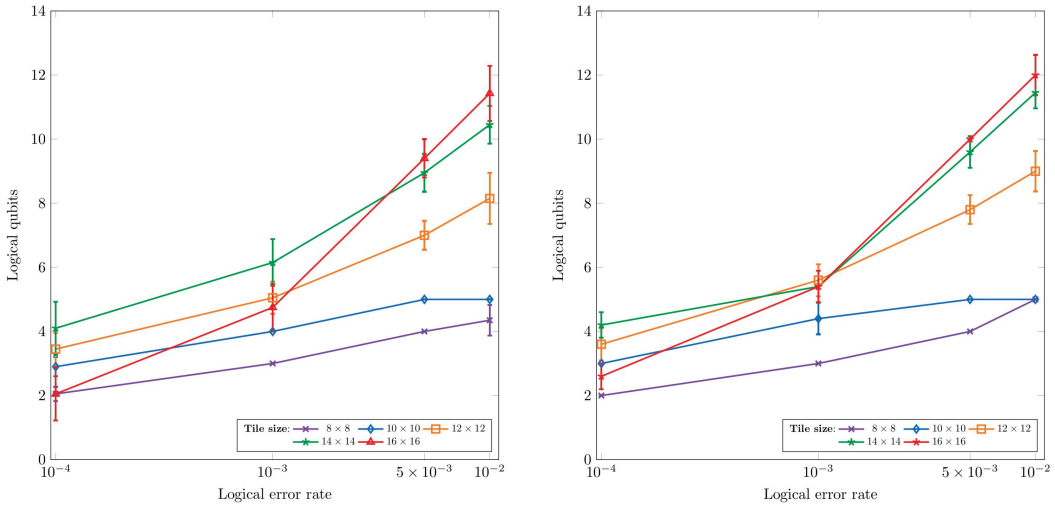

Figure 4.1: Results of simulating the number of logical qubits versus logical error rates using (a) SA and (b) GA.

number of qubits for a specific range of logical errors, just as in the previous section. To be more concise, in this part of the experiments, I only utilize the simulated annealing approach, with an initial temperature of 2400 K, and employ 1000 iterations. Similarly to the previous analysis, we also examine how the approach behaves across different sizes of square lattices.

Figure 4.2 shows the results of using partially open holes to encode logical qubits, showing that, on average, it leads to two to three times as many logical qubits as closed holes. These findings align with Equation 3.2 and demonstrate that surface codes with partially open holes produce more logical qubits under the same logical error rate. As in the previous case, the code distance acts as a limiting factor in increasing the number of encoded logical qubits. Additionally, I should consider the impact of partially open holes on errors by comparing the average number of placed closed and partially open holes.

Table 4.1 shows the average number of holes and logical qubits for different logical thresholds in a 14×14 square lattice obtained by SA. I can see that SA, on average, places more closed holes in a lattice than partial open holes under the given logical error. The results suggest that partially open holes increase the logical error rate, thus aligning with my expectations.

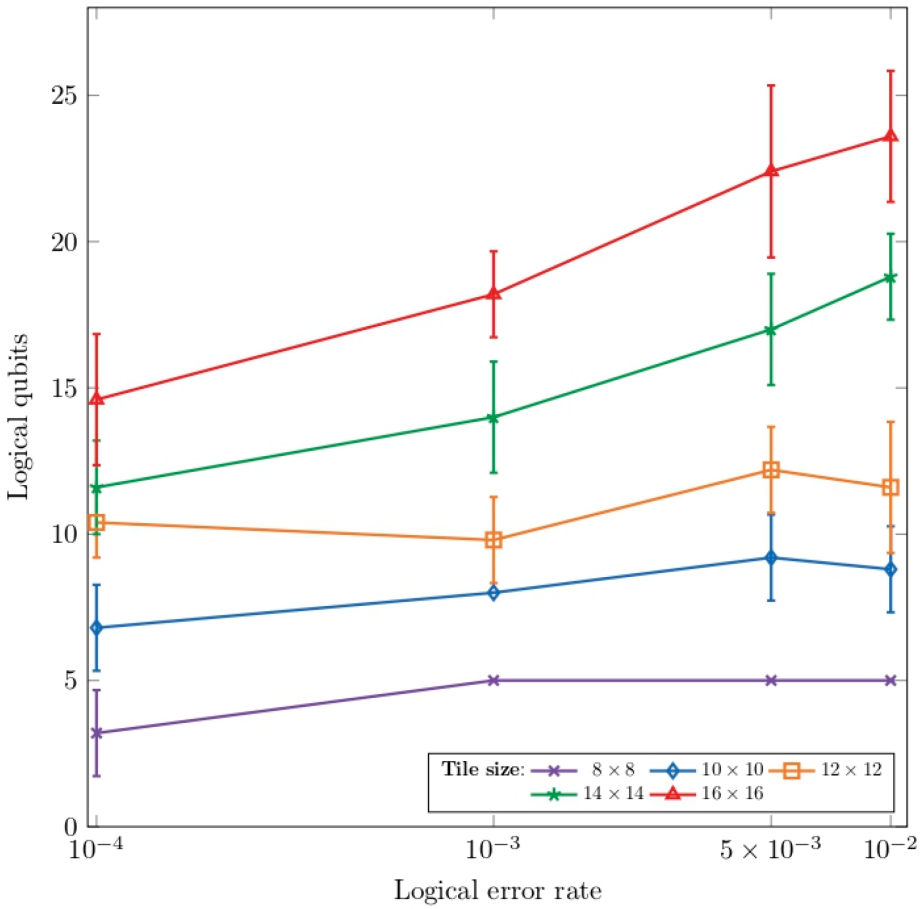

Figure 4.2: Results of simulating the number of logical qubits versus logical error rate for a partially open hole.

Removal of the stabilizer (represented by faces) and additional adjacent physical qubits (partially open edge) lead to worse error behavior than only removing the stabilizer (closed edge). However, partially open holes form more logical qubits in square lattices than closed ones.

4.3 Influence of Parameters of Optimization Algorithm on Numbers of Logical Qubits

In previous experiments, I employed the SA and GA to assess the number of logical qubits which can be encoded by inserting defects in 2D lattice. However, I only assess the perfor-

Table 4.1: Number of holes and logical qubits for different logical error thresholds in 14×14 square lattice.

Logical Error	Closed Holes		Partially Open Holes	
	Logical Qubits	Number of Holes	Logical Qubits	Number of Holes
10^{-4}	4.20	3.20	12.20	4.40
10^{-3}	5.40	4.40	12.20	4.40
5×10^{-3}	9.60	8.60	16.40	5.80
10^{-2}	11.40	10.40	18.40	6.60

mance of optimization for fixed value of hyperparameters. The performance of simulated annealing (SA) and genetic algorithms (GAs) is sensitive to hyperparameters that influence convergence, exploration–exploitation balance, and solution quality. In SA, factors like maximal temperature and cooling schedule affect the size of solution space; slower cooling enhances global search but increases computation time, while faster cooling risks premature convergence. In GAs, parameters such as population size, crossover probability, and mutation rate manage genetic diversity and the ability to escape local optima. Effective tuning of these parameters is essential for robust optimization, often requiring problem-specific experimentation and heuristics.

To evaluate the influence of hyperparameters on algorithm performance, I assessed the number of logical qubits produced by SA and GA under varying hyperparameter settings. My analysis focused specifically on a lattice 16×16, targeting a logical error rate of 5×10^{-3} to ensure an adequately large solution space. For SA, I investigated the impact of the maximum temperature (T_{MAX}) and the number of cooling steps. By selecting higher maximal temperatures and increasing the number of annealing steps, I aimed to investigate whether the SA algorithm could effectively explore the solution space to find lattices that encode a greater number of logical qubits. Regarding the GA, my analysis examined the effects of the number of generations and the number of parent solutions. The obtained results are summarized in Tables 4.2 and 4.3.

Table 4.2: Influence of SA hyperparameters on the number of obtained logical qubits in a 2D lattice.

T_{MAX} / **No. of Steps**	**2400 K**	**4800 K**	**7200 K**
500	8.80 ± 0.75	8.20 ± 0.75	8.20 ± 1.17
1000	9.40 ± 0.49	9.20 ± 0.40	9.00 ± 0.63

Table 4.3: Influence of GA hyperparameters on the number of obtained logical qubits in a 2D lattice.

No. of Solutions per Generation / **No. of Generations**	**50**	**200**	**400**
500	9.80 ± 0.75	9.60 ± 0.80	9.40 ± 0.49
1000	9.80 ± 0.40	10.00 ± 0.63	10.00 ± 0.63

The results presented in Table 4.2 indicate that the maximum temperature parameter of the SA algorithm has a negligible effect on the obtained number of logical qubits. In contrast, increasing the number of annealing steps results in marginal improvement, yielding less than one additional logical qubit on average. This limited gain can be attributed to the intrinsic constraints of surface codes: for a given threshold, the code reaches a point beyond which additional holes in the code significantly degrade the error behavior. These conclusions are further supported by the data in Table 4.3, as they exhibit similar trends. Specifically, the number of logical qubits produced across different configurations of the GA remains comparable, with variations falling within the range of standard deviation. Although GA yields a higher number of encoded logical qubits compared to the SA approach, the final counts remain relatively close.

In addition to the previously conducted experiments, I further analyzed the influence of SA hyperparameters on the number of logical qubits generated by inserting holes with partially open edges into the lattice. Introducing partially open edges significantly enlarges the solution search space, compared to scenarios involving fully closed-border edges, due to increased

degrees of freedom for hole placement and shape optimization. As in previous experiments, the impact of maximum temperature and cooling steps was investigated.

The results presented in Table 4.4 demonstrate a greater sensitivity of the simulated annealing (SA) algorithm to its parameter settings, which can be attributed to the larger search space in this case. However, the observed variations in the number of encoded logical qubits remain within the bounds of the mean and standard deviation. These findings further highlight the limitations of surface codes, suggesting that their capacity to encode logical qubits is fundamentally restricted.

Table 4.4: Influence of SA hyperparameters on the number of obtained logical qubits with partially open holes.

T_{MAX} / **No. of Steps**	**2400 K**	**4800 K**	**7200 K**
500	20.00 ± 1.90	18.80 ± 3.06	20.60 ± 1.20
1000	23.60 ± 1.20	21.80 ± 1.47	21.20 ± 2.40

4.4 Influence of Code Distance on Error Characteristics

In the previous section, I analyzed the maximal number of qubits for a specific logical error rate. However, this analysis was specifically focused on a code distance of three. To understand how the code distance affects the error behavior of the surface code, I utilized optimization algorithms in a different scenario. Instead of aiming to find the maximum number of logical qubits for a given logical error rate, my focus shifted to holding the number of logical qubits at one and attempting to identify a surface code with a minimal logical error rate. In other words, my goal is to minimize the logical error rate while keeping the number of logical qubits fixed at one in the surface code. I employ only closed holes because I do

not consider the number of logical qubits.

In my optimization algorithm, the code distance serves as a parameter. Like in the first experiment, the SA process is initiated with an open boundary square lattice. When adding a new hole, the hole needed to have d bordering edges and be at least $d-1$ faces away from the lattice border for a code distance of d. A larger code distance could result in a reduction in the number of holes placed on the lattice. Therefore, I only considered codes with two logical qubits or one hole in the lattice with an open border for each code distance. To generate potential solutions for SA, I added a hole whose size was determined by the code distance and moved it across the lattice.

To evaluate potential solutions, I employ hyperbolic tangents for energy function:

$$E = \tanh(\text{LER}) \tag{4.1}$$

where LER represents the logical error rate obtained for the current solution in this equation. The energy function saturates large values of error rate, thus pushing the SA algorithm to avoid a hole position with a large logical error rate. Figure 4.3 illustrates the error characteristics of the obtained 18×18 surface codes with varying code distances on a lattice encoding two logical qubits. From the image I can differ three groups of that share similar error performance. The tiles with code distances of 3, 4, and 5 exhibit the poorest error performance. In contrast, tiles characterized by code distances 7 and 8 demonstrate the best performance, achieving the lowest logical error rates across the evaluated erasure probabilities. Between these two lies the error distribution corresponding to tiles with a code distance of 6. These results align well with the expected behavior of surface codes, where increased code distance results in improved error resilience. However, it is important to emphasize that, in a defect-based encoding approach, increasing the code distance involves strategically removing physical qubits (creating defects) rather than adding qubits to the lattice.

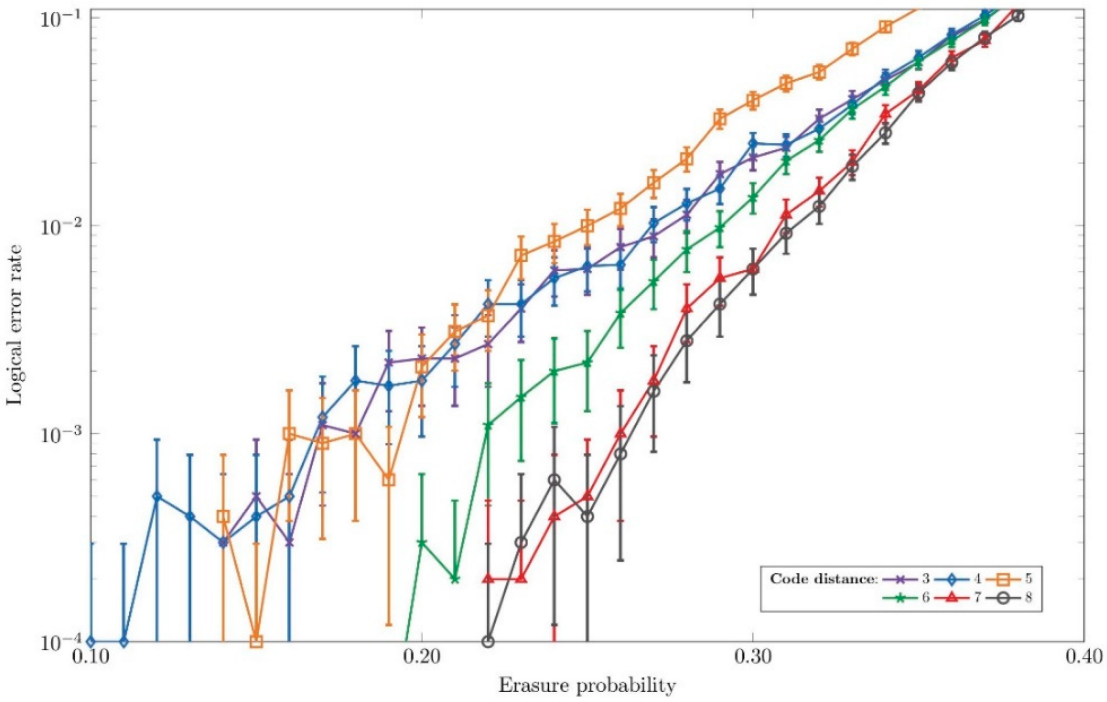

Figure 4.3: Logical error rates for given erasure probabilities for surface codes with a size of 18×18 at various code distances of 3, 4, 5, 6, 7, and 8 are illustrated.

4.5 Discussion

The previous sections have demonstrated the potential of a defect-based approach for encoding multiple logical qubits. These results are based on the Squab simulator, which evaluates the error rate of examined lattices. While practical implementation aspects of defect-based encoding are beyond the scope of this paper, this section reflects on my findings to highlight specific advantages and limitations inherent to this approach and analysis.

I first consider the encoding density achieved by my method. Referring to the results obtained for different logical error thresholds on a 14×14 square lattice (Table 4.1), the lattice configuration consists of 420 physical qubits, yielding a maximal encoding density of up to 30 physical qubits per logical qubit. To evaluate the effectiveness of my approach, I compare my results with the theoretical limitations for logical qubit packing in planar surface code. According to Delfosse et al. [17], the maximum number of logical qubits encoded via closed

holes is approximately as follows:

$$n \sim 3kd^2 \tag{4.2}$$

where n denotes the number of physical qubits, k is the number of logical qubits, and d is the code distance. For a 14×14 lattice with a code distance of 3, the theoretical maximum equals 15 logical qubits. As indicated in Table 1, my SA algorithm achieves results very close to this theoretical limit. However, it is important to emphasize that Delfosse et al. [17] do not consider logical error rates when deriving their theoretical bounds for logical qubit encoding on square lattices. Moreover, my results for lattices incorporating partially open holes are aligned with the conclusion presented in [17], demonstrating that partially open defects offer distinct advantages over standard surface codes for storing quantum information within two-dimensional lattices.

In experiments, the simulator assumes that the decoding algorithm knows the locations of the erased qubits, which can usually be measured in experimental setups. Decoding stabilizer codes within this context entails solving a linear system, which generally has cubic complexity. However, real-life decoding is more complex than demonstrated in experimental settings. One significant downside of the defect-based approach is the complexity involved in decoding stabilizer measurements. The author in [32] employs a tensor-network decoding algorithm for near-optimal error correction. The approach revolves around modeling logical channels. According to the proposed approach, the complexity of encoding is related to a logical channel, with its size increasing exponentially as the number of encoded logical qubits rises. Nevertheless, they do not dismiss the possibility of an optimal decoder within this framework. Several machine learning-based decoding algorithms have recently been introduced to enhance performance and reduce latency in QEC codes [33]. Given their rapid and constant inference time and State-of-the-Art decoding performance, I anticipate that neural network-based decoding will provide an effective solution for correcting errors in

surface codes containing holes.

While my simulation results demonstrate the potential advantages of the defect-based approach, I must acknowledge limitations in my benchmarking methodology. The erasure channel model employed by Squab assumes perfect knowledge of erasure locations, representing an idealization of real quantum hardware. In physical implementations, detecting erasures would require additional measurement circuitry and syndrome extraction rounds. These measurements could introduce errors by themselves, and pausing the main syndrome extraction to check for erasures could allow other errors to accumulate. Additionally, real detection mechanisms might suffer from false positives (incorrectly identifying intact qubits as erased) or false negatives (failing to detect actual erasures), thus affecting the decoder's performance. Though these practical considerations are not captured in my current simulations, they represent important engineering challenges for implementing defect-based surface codes in hardware. The theoretical performance bounds established in this work provide targets for hardware implementations to aspire to, while acknowledging that practical implementations will likely operate with some performance gap relative to these idealized results due to the overhead of erasure detection.

In physical implementations of surface codes, erasure detection requires specific measurement protocols. In superconducting qubit platforms, erasures often manifest as leakage events where the qubit state leaves the computational subspace. These leakage events are detectable through high-fidelity readout schemes that can discriminate between computational states and leakage levels. The detection typically involves applying controlled operations between the potentially leaked qubit and an ancilla qubit, followed by measurement of the ancilla [34, 35, 36]. For trapped ion systems, erasures might correspond to ion loss or transitions to unwanted electronic states, which can be detected through fluorescence measurements [37, 38]. The physical noise sources that generate erasures include microwave pulse miscalibration in superconducting circuits, laser intensity fluctuations in trapped ions, and

environmental coupling effects. Experimental implementations often use engineered noise channels to benchmark code performance, where controlled perturbations are deliberately introduced through miscalibrated gates or engineered coupling to simulate realistic error processes [36, 38].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Expanding the number of logical qubits is crucial for advancing quantum computing, especially in the NISQ era. While the patch-based approach for encoding multiple qubits is prevalent, it is equally important to explore the potential of the defect-based approach. The latter method, which encodes logical qubits by placing the holes in a lattice, poses a trade-off between the number of logical qubits and logical error rates. To tackle the trade-off above, I employed optimization algorithms, including SA and GA, to assess the maximum number of logical qubits for a given error rate. As expected, larger lattices yield a larger degree of freedom when moving the holes in code, and, thus, I can accommodate more logical qubits for a range of error rates. The results also revealed properties of partially open holes, which help encode more logical qubits than closed holes, showcasing the potential for improving the code density behind defect-free approaches. I also conducted experiments regarding code distance and holes in surface code. The findings show that higher code distances result in better performance for error correction. While existing research, such as the development of patch-based quantum processors like Google's Willow [5], favors a patch-based approach

for encoding multiple qubits, combining patch-based and defect-based approaches could lead to more efficient encoding of logical qubits. For instance, defect-based methods may offer higher qubit density or improved scalability when integrated with patch-based architectures.

5.2 Future Work

Exploring the synergy between patch-based and defect-based approaches is an area I plan to investigate further in my future work. In addition, while defect-based surface codes support braiding-based logical operations without ancillary patches, preserving the effective code distance during such dynamic operations remains a critical challenge. Improper spacing or routing of defects during braiding may reduce the minimum distance, increasing the risk of logical errors. Future work will extend my current framework to explore strategies and automated techniques for preserving code distance during braiding, potentially by integrating braid scheduling heuristics or adaptive layout constraints into the optimization process. Finally, I acknowledge the limitations of my current benchmarking approach. My simulations rely on the erasure channel with perfect knowledge of erasure locations; it provides computational efficiency but idealizes certain aspects of physical implementations. In practice, detecting erasures would introduce additional complexity, potential errors, and overhead not captured in my current model. Future work should expand my analysis to include more realistic noise models that account for imperfect erasure detection, false positives/negatives in detection mechanisms, and the additional error sources associated with erasure checking in physical implementations. Combining more comprehensive noise models with my optimization approach would provide even more robust guidance for practical implementations of defect-based surface codes in future quantum computing hardware.

References

[1] John Preskill. "Quantum Computing in the NISQ Era and Beyond". In: *Quantum* 2 (2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[2] Christian Kraglund Andersen et al. "Repeated Quantum Error Detection in a Surface Code". In: *Nature Physics* 16 (2020), pp. 875–880. DOI: 10.1038/s41567-020-0920-y.

[3] Rajeev Acharya et al. "Suppressing Quantum Errors by Scaling a Surface Code Logical Qubit". In: *Nature* 614 (2023), pp. 676–681. DOI: 10.1038/s41586-022-05434-1.

[4] Daryus Chandra et al. "Quantum Topological Error Correction Codes: The Classical-to-Quantum Isomorphism Perspective". In: *IEEE Access* 6 (2018), pp. 13729–13757. DOI: 10.1109/ACCESS.2017.2784417.

[5] Rajeev Acharya et al. "Quantum Error Correction below the Surface Code Threshold". In: *Nature* 638 (2025). Published online 2024-12-09, pp. 920–926. DOI: 10.1038/s41586-024-08449-y.

[6] A. Y. Kitaev. "Fault-Tolerant Quantum Computation by Anyons". In: *Annals of Physics* 303 (2003), pp. 2–30. DOI: 10.1016/S0003-4916(02)00018-0.

[7] Austin G. Fowler et al. "Surface Codes: Towards Practical Large-Scale Quantum Computation". In: *Physical Review A* 86 (2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324.

[8] Clare Horsman et al. "Surface Code Quantum Computing by Lattice Surgery". In: *New Journal of Physics* 14 (2012), p. 123011. DOI: 10.1088/1367-2630/14/12/123011.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th Anniversary. Cambridge, UK: Cambridge University Press, 2010.

[10] David McMahon. *Quantum Computing Explained*. Hoboken, NJ, USA: Wiley-Interscience, 2007.

[11] Emanuel Knill and Raymond Laflamme. "Theory of Quantum Error-Correcting Codes". In: *Physical Review A* 55 (1997), pp. 900–911. DOI: 10.1103/PhysRevA.55.900.

[12] Josu Etxezarreta Martinez et al. "Approximating Decoherence Processes for the Design and Simulation of Quantum Error Correction Codes on Classical Computers". In: *IEEE Access* 8 (2020), pp. 172623–172643. DOI: 10.1109/ACCESS.2020.3025619.

[13] Chris Wilmott. "On the Operator-Sum Formalism". In: *arXiv* (2011). arXiv: 1104.1313 [quant-ph].

[14] Daniel Gottesman. "Stabilizer Codes and Quantum Error Correction". PhD thesis. Pasadena, CA, USA: California Institute of Technology, 1997.

[15] James Roffe. "Quantum error correction: an introductory guide". In: *Contemporary Physics* 60.3 (2019), pp. 226–245. DOI: 10.1080/00107514.2019.1667078.

[16] David S. Wang et al. "Threshold Error Rates for the Toric and Planar Codes". In: *Quantum Information and Computation* 10 (2010), pp. 456–469. DOI: 10.26421/QIC10.5-6-6.

[17] Nicolas Delfosse, Pradeep Iyer, and David Poulin. "A Linear-Time Benchmarking Tool for Generalized Surface Codes". In: *arXiv* (2016). arXiv: 1611.04256.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[19] Savvas Varsamopoulos, Koen Bertels, and Carmen G. Almudever. "Comparing Neural-Network-Based Decoders for the Surface Code". In: *IEEE Transactions on Computers* 69.2 (2020), pp. 300–311. DOI: 10.1109/TC.2019.2948615.

[20] Y. Chen, T. Václavík, and S. Pirandola. "Quantum-Enhanced Error Correction in a Lossy Bosonic Channel". In: *Physical Review Research* 6.1 (2024), p. 013154. DOI: 10.1103/PhysRevResearch.6.013154.

[21] Oscar Higgott. "PyMatching: A Python Package for Decoding Quantum Codes with Minimum-Weight Perfect Matching". In: *ACM Transactions on Quantum Computing* 3.3 (2022), pp. 1–16.

[22] Guillaume Duclos-Cianci and David Poulin. "A Renormalization Group Decoding Algorithm for Topological Quantum Codes". In: *2010 IEEE Information Theory Workshop (ITW)*. IEEE, 2010, pp. 1–5.

[23] N. Sundaresan et al. "Demonstrating Multi-Round Subsystem Quantum Error Correction Using Matching and Maximum Likelihood Decoders". In: *Nature Communications* 14.1 (2023), p. 2852.

[24] Bettina Heim, Krysta M. Svore, and Matthew B. Hastings. *Optimal Circuit-Level Decoding for Surface Codes*. 2016. arXiv: 1609.06373 [quant-ph].

[25] Johannes Bausch et al. "Learning High-Accuracy Error Decoding for Quantum Processors". In: *Nature* 635 (2024), pp. 834–840. DOI: 10.1038/s41586-024-08148-8.

[26] Nicolas Delfosse, Pradeep Iyer, and David Poulin. "Generalized Surface Codes and Packing of Logical Qubits". In: *arXiv* (2016). arXiv: 1606.07116.

[27] Benjamin J. Brown et al. "Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code". In: *Physical Review X* 7 (2017), p. 021029. DOI: 10.1103/PhysRevX.7.021029.

[28] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Berlin, Heidelberg: Springer, 2004. DOI: 10.1007/978-3-540-24777-7.

[29] Daniel Litinski. "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery". In: *Quantum* 3 (2019), p. 128. DOI: 10.22331/q-2019-03-05-128.

[30] Kento Hamada, Yasunari Suzuki, and Yuki Tokunaga. "Efficient and High-Performance Routing of Lattice-Surgery Paths on Three-Dimensional Lattice". In: *arXiv* (2024). arXiv: 2401.15829.

[31] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989. ISBN: 978-0201157673.

[32] A. S. Darmawan. *Optimal Adaptation of Surface-Code Decoders to Local Noise*. 2024. arXiv: 2403.08706 [quant-ph].

[33] R. W. Overwater, M. Babaie, and F. Sebastiano. "Neural-Network Decoders for Quantum Error Correction Using Surface Codes: A Space Exploration of the Hardware Cost-Performance Tradeoffs". In: *IEEE Transactions on Quantum Engineering* 3 (2022), pp. 1–19.

[34] M. McEwen et al. "Removing Leakage-Induced Correlated Errors in Superconducting Quantum Error Correction". In: *Nature Communications* 12 (2021), p. 1761.

[35] K. Chang et al. *Surface Code with Imperfect Erasure Checks*. 2024. arXiv: 2408.00842 [quant-ph].

[36] H. Levine et al. "Demonstrating a Long-Coherence Dual-Rail Erasure Qubit Using Tunable Transmons". In: *Physical Review X* 14.1 (2024), p. 011051. DOI: 10.1103/PhysRevX.14.011051.

[37] A. Quinn et al. *High-Fidelity Entanglement of Metastable Trapped-Ion Qubits with Integrated Erasure Conversion*. 2024. arXiv: 2411.12727 [quant-ph].

[38] P. Schindler et al. "A Quantum Information Processor with Trapped Ions". In: *New Journal of Physics* 15 (2013), p. 123012. DOI: 10.1088/1367-2630/15/12/123012.