POLITECNICO DI TORINO

Master's degree in Physics of Complex Systems

Master's thesis

A decision-theoretic framework for quantum error correction

Modeling surface code decoding as a partially observable Markov decision process

Supervisor
Prof. Davide Girolami

Candidate
Salvatore Ferraro
ID: 278214

External supervisor
Giacomo Vitali
Links Foundation

24 October 2025 — Academic Year 2024–2025

Contents

Introduction

Background

Quantum information is protected from noise by means of *error-correcting codes*. These codes allow for encoding of the logical qubits into a higher number of physical qubits, so that noise affecting the system corrupts just part of the information, while the rest can be used for recovery. Noise causes errors on the qubits that are detected by measuring properties called *syndromes*. Topological quantum error-correcting codes protect quantum information by encoding logical qubits into global, topological properties of a physical system. Their properties make them resistant to local noise models. This makes topological codes particularly attractive for building scalable, fault-tolerant quantum computers. Among topological codes, surface codes, introduced in [1], are the most extensively studied and experimentally viable. Surface codes require only local, nearest-neighbor interactions between qubits, aligning well with existing quantum hardware. A central limitation in the surface code architecture is that syndromes are not capable of fully determining the error affecting the qubits. This is why these codes need an operation called *decoding*, the means determining an appropriate correction given the information on errors coming from syndromes.

Due to the ambiguity inherent in syndrome measurements, decoding is a non-trivial computational task. Decoding methodologies must deal with many constraints: they must be faster than the system decoherence and cope with noise in syndrome measurements (syndrome measurements affected by noise are called *faulty syndromes*). Several approaches have been proposed, the first ones being algorithms performed at runtime. Modern approaches often use neural networks: they train them in advance so that corrections can be quickly evaluated at runtime.

A paradigmatic contribution in this direction is [2], where the decoding problem under noisy syndrome measurements is formulated as a reinforcement learning task and solved by the training of a *decoding agent*.

Problem statement

The scope of this thesis is to explore reinforcement learning solutions to the problem of decoding surface codes under faulty syndrome measurements and to address some of the limitations of [2].

Approach overview

We define a new framework to train decoding agents for quantum hardware with approximately known noise parameters. The framework is based on mapping the problem of optimal decoding of surface code to the problem of optimal control of a partially observable Markov decision process (POMDP). This mapping comes with computational challenges that we deal with. We explain how to find good policies for controlling our decoding POMDP by reinforcement learning. Our

framework allows for continuous update of the decoding strategy to track fluctuations in noise parameters. We sketch a procedure to perform this update.

Overview of this thesis

This thesis is structured into two parts. The first part is literature-based and prepares the reader for the second part, which presents our original research contributions.

Background material

Chapter 1: Basics of quantum information and error correction

This chapter starts by recapping some basic concepts of quantum information. It then presents the problem of quantum errors and the principles of quantum error correction through error-correcting codes. Readers already familiar with quantum error correction may skip this chapter.

Chapter 2: Stabilizer topological codes

This chapter dives into more advanced topics in quantum error correction. First, the stabilizer formalism is introduced as a framework for describing quantum codes. This formalism is then applied to topological codes, with a particular focus on the surface code. We end the chapter by describing the physical implementation of the surface code and the errors affecting it.

Chapter 3: Hidden Markov models and reinforcement learning

This chapter shifts to concepts from machine learning, information theory, and control theory. It begins with a review of hidden Markov models and POMDPs. Then we discuss reinforcement learning and policy gradient methods. We end with some information-theoretic algorithms that will be used in the research work.

Chapter 4: Decoding the surface code

This chapter surveys the current methodologies used to perform decoding of the surface code.

Original work

Chapter 5: A new approach to decoding

This chapter introduces our proposed decoding methodology. It starts with a critique of current approaches and justifies our approach as an effort to overcome their limitations. We then define the structure of our architecture.

Chapter 6: The surface code cycle as a hidden Markov model

In this chapter, we detail the mapping of the measurement cycle of the surface code to a hidden Markov model. We then prove that it is possible to describe the dynamics of the surface code as those of a hidden Markov model, so that decoding can be mapped to controlling a POMDP supported by this hidden Markov model.

Chapter 7: The decoding problem as a POMDP

We complete the mapping by defining the additional decoding POMDP elements.

Chapter 8: Solution of the decoding POMDP and definition of the workflows

This chapter presents a reinforcement learning approach to solving the decoding POMDP. The costituents workflows of our method are detailed.

Chapter 9: Conclusions

We summarized strength and weakness of our work and lay down a roadmap for future work.

Part I

Background material

1 Review of quantum error correction

This chapter reviews our read of the basic quantum error correction background for readers with a physics background but no prior exposure to this subject. The main purpose is to define notation, nomenclature and to introduce our reasoning on the topic (that will be referred to in the next chapters) rather than providing a complete exposition. The material is primarily based on [3], unless otherwise stated. For more detailed explanations, the reader is referred to the indicated sources.

1.1 Basics

This section introduces the notation used throughout the thesis and reviews fundamental concepts from quantum information theory.

1.1.1 States

A *qubit* is a two-level quantum system that serves as the fundamental unit of quantum information. A qubit may correspond to an actual physical system, in which case it is referred to as a *physical qubit*, or it may be defined as an abstract two-dimensional subspace embedded in a larger Hilbert space, in which case it is called a *logical qubit*.

A collection of *distinguishable* qubits used together in a quantum computation or protocol is called a *register*. Registers can likewise be either physical or logical.

The Hilbert space associated with a physical register of n qubits is

$$\mathcal{H}_P = \left(\mathbb{C}^2\right)^{\otimes n},$$

while a logical register of l qubits is described by the space

$$\mathcal{H}_L = \left(\mathbb{C}^2\right)^{\otimes l}.$$

The most general state of a register is represented by a density matrix ρ.

Types of states

A *pure state* is described by a single vector $|\psi\rangle$ in Hilbert space, or equivalently by the rank-1 density matrix $\rho = |\psi\rangle\langle\psi|$, which satisfies $\mathrm{Tr}(\rho^2) = 1$. Pure states contain maximal information about a system's preparation. A *coherent superposition* of basis states, such as

$$|\psi\rangle = \sum_i \alpha_i |i\rangle,$$

is still a pure state. Such superpositions do not represent epistemic uncertainty, but describe quantum coherence and interference effects over a known state, even if measurement outcomes are probabilistic.

Suppose we want to express *epistemic* uncertainty over a quantum state, that is expressing the state of the system as a hidden random variable. This can be written as:

$$\rho = \sum_i w_i \left|\psi_i\right\rangle \left\langle\psi_i\right|,$$

where $w_i \geq 0$, $\sum_i w_i = 1$, and each $|\psi_i\rangle$ is a pure state, is called a *mixture of pure states*. The w_i are called *weights*.

There is a fundamental difference between classical random variables and quantum random variables. In classical probability theory, one defines distributions over mutually exclusive and exhaustive categories, such as "cat","dog", or "plant". These categories are disjoint, and the system is in exactly one of them. This guarantees that the probabilities reflect ignorance over mutually exclusive options. However, if one instead defines categories like "animal" and "cat", which are not mutually exclusive, then assigning classical probabilities becomes ambiguous unless a refinement into exclusive subcategories is made. The same happens in quantum mechanics. The measurement of the state of a mixture is distributed with the same probabilities as the mixture only if it supported on *orthogonal* states, since orthogonality implies *mutual distinguishability*, which means there exists a measurement that can perfectly identify which state the system is in. If the $|\psi_i\rangle$ are non-orthogonal the probability of finding the system in one of the $|\psi_i\rangle$ is not equal to the weight, as we prove in the following. In this case the hidden state of the system is a random variable with a different distribution that the measure of its state.

More generally, a mixture can be expressed as a convex sum of density matrices:

$$\rho = \sum_i w_i \rho_i,$$

where each ρ_i is itself a density matrix. Again if the ρ_i are mutually distinguishable - meaning there exists a single measurement that can reliably tell them apart - the mixture corresponds to a classical probability distribution over different states.

A *mixed state* is any state ρ for which $\mathrm{Tr}(\rho^2) < 1$, and which cannot be described by a single vector in Hilbert space. Any mixed state can be written as a mixture of orthogonal pure states in some basis, due to the spectral theorem (since ρ is Hermitian). This means that any mixed state can be considered as classical random variable over some basis. The average value of any observable can be computed as average on the preparation probability distribution as :

$$\langle A \rangle = \sum_i w_i \left\langle\psi_i\right| A \left|\psi_i\right\rangle = \mathrm{Tr}(\rho A).$$

A density matrix may arise from a statistical ensemble or from a partial description of a larger joint system.

Joint spaces

The state of a composite system with Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$ is described by a density matrix ρ_{AB} acting on that space. A fundamental distinction among such states is between those that are *separable* and those that are *entangled*.

A state ρ_{AB} is said to be separable if it can be expressed as a convex combination of product states:

$$\rho_{AB} = \sum_k p_k \, \rho_k^A \otimes \rho_k^B,$$

where $p_i \geq 0$, $\sum_i p_i = 1$, and $\rho_i^A \in \mathcal{D}(\mathcal{H}_A)$, $\rho_i^B \in \mathcal{D}(\mathcal{H}_B)$.

If a state cannot be written in this form, it is called *entangled*. For example, the Bell state

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

is a pure entangled state on $\mathcal{H}_A \otimes \mathcal{H}_B$.

If we consider product states between bases of space $\mathcal{H}_A$, {
$keti$}, and $\mathcal{H}_B$, {
$ketj$}, that is if the state can be written in the form:

$$\rho_{AB} = \sum_{ij} p_{ij} \; |ij\rangle \otimes \langle ij| \tag{1.1}$$

then the resulting state can be interpreted as correlated probabilistic mixtures on the joint space.

Given a joint state the state of the A subspace is obtained taking the partial trace over subsystem B:

$$\rho_A = \mathrm{Tr}_B(\rho_{AB}).$$

Partial tracing is the quantum equivalent of classical marginalization. One can show that if the joint space is in the form of Eq. 1.1, then the result of partial tracing is equivalent to classical marginalization, i.e.

$$\rho_A = \sum_i p_i \; |i\rangle \langle i| \quad p_i = \sum_j p_{ij}$$

A two-qubit unitary U is *entangling* if there exists a separable pure state $|\alpha\rangle \otimes |\beta\rangle$ such that $U\big(\,|\alpha\rangle \otimes |\beta\rangle\,\big)$ is entangled.

For the CNOT gate with control on the first qubit and target on the second, let $|\alpha\rangle = a\,|0\rangle + b\,|1\rangle$ and $|\beta\rangle$ arbitrary. Then

$$\mathrm{CNOT}\left((a\,|0\rangle + b\,|1\rangle) \otimes |\beta\rangle\,\right) = a\;|0\rangle \otimes |\beta\rangle + b\;|1\rangle \otimes X\,|\beta\rangle\,,$$

Fidelity

Fidelity is a measure of the similarity between two quantum states. Given two density matrices ρ and σ, the fidelity is defined as

$$F(\rho,\sigma) = \left(\mathrm{Tr}\left[\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right]\right)^2. \tag{1.2}$$

It is symmetric and takes values between 0 and 1, with $F(\rho,\sigma) = 1$ if and only if $\rho = \sigma$, and $F(\rho,\sigma) = 0$ when ρ and σ have orthogonal supports.

When one of the two states is pure, say $\sigma = |\psi\rangle\langle\psi|$, the fidelity simplifies to

$$F(|\psi\rangle\,,\rho) = \langle\psi|\,\rho\,|\psi\rangle\,, \tag{1.3}$$

which is always non-negative because every density matrix is a positive semidefinite operator.

If ρ is also a pure state, say $\rho = |\phi\rangle\langle\phi|$, the fidelity further simplifies to

$$F(|\psi\rangle\,,|\phi\rangle) = |\,\langle\psi|\phi\rangle\,|^2.$$

Thus, in the case of pure states, fidelity reduces to the squared modulus of the inner product, and can be seen as a natural generalization of the scalar product to mixed states.

Classical simulability of quantum states

In general, classically representing a quantum state requires resources that scale exponentially in the number of qubits. A density matrix ρ for n qubits is a $2^n \times 2^n$ complex matrix, meaning that storing or manipulating ρ directly requires memory and computation time that grow exponentially with n. This makes exact classical simulation of arbitrary quantum systems intractable for large n and it is also one of the reason for the potential advantage in quantum information. Any classical attempt to simulate a quantum system must be aware of this.

1.1.2 Measurement

In quantum mechanics, the most general description of a measurement is provided by a *Positive Operator-Valued Measure* (POVM). POVMs extend the notion of measurement beyond projective measurements (as described in the Copenhagen interpretation), allowing a broader class of operations.

A POVM is defined by a set of positive semi-definite operators $\{E_m\}$ acting on the Hilbert space $\mathcal{H}$, satisfying the completeness relation:

$$\sum_m E_m = \mathbb{I}.$$

Each operator E_m corresponds to a possible measurement outcome m, and the probability of obtaining outcome m when the system is in a pure state $|\psi\rangle$ is given by the *Born rule*:

$$\mathbb{P}(m) = \langle\psi| E_m |\psi\rangle .$$

However, a POVM does not uniquely determine how the quantum state evolves after the measurement. It is just an abstract description of the statistics over the outcomes.

To describe the post-measurement state, one introduces a set of *measurement operators* (or Kraus operators) $\{M_m\}$, such that

$$E_m = M_m^\dagger M_m.$$

Upon obtaining outcome m, the pure state transforms as

$$|\psi\rangle \to \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| E_m |\psi\rangle}}.$$

Suppose now the system is initially prepared in one of several pure states $|\psi_i\rangle$ with classical probabilities p_i, so that the overall state is described by the density matrix

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| .$$

By law of total probability we obtain the probability of outcome m averaging over the Born probabilities of all the states in the mixture. Written in a basis independent way the probability becomes:

$$\mathbb{P}(m) = \mathrm{Tr}(E_m \rho),$$

Which is the generalization of Born rule to epistemic uncertainty.

The post-measurement state update reflects both the physical evolution of each component state due to the measurement and the observer's Bayesian update of their knowledge upon learning the outcome.

After observing outcome m, we update our classical belief about the hidden state of the quantum system by Bayes theorem:

$$\mathbb{P}(i \mid m) = \frac{\mathbb{P}(m \mid i)p_i}{\mathbb{P}(m)} = \frac{\langle\psi_i| E_m |\psi_i\rangle p_i}{\mathrm{Tr}(E_m\rho)}.$$

This, combined with the physical evolution of every pure state in the mixture, leads to the overall post-measurement state update:

$$\rho \to \frac{M_m\rho M_m^\dagger}{\mathrm{Tr}(E_m\rho)}.$$

When the measurement is followed by conditioning on the observed outcome, it is called a *selective measurement*. This remains true in every basis, so it can be extended to general density matrices, even not written as mixtures, because every density can be written as a probability distribution over a basis.

If instead the measurement outcome is not recorded or used, the resulting state is the weighted average over all possible outcomes,

$$\rho \to \sum_m M_m\rho M_m^\dagger,$$

which is called a *non-selective measurement*.

We can now show the difference between density matrices and classical probability distribution. The probability of finding the system in one of the pure state $|\psi_i\rangle$ constituent of the mixture is:

$$\mathbb{P}(\psi_i) = \mathrm{Tr}(|\psi_i\rangle\langle\psi_i|\,\rho).$$

When ρ contains non-orthogonal components this does not correspond to the weight p_i, as can be shown by some examples.

1.1.3 Pauli operators

Before moving on, we clarify the nomenclature related to Pauli operators.

A *generalized Pauli operator* on n qubits is an operator of the form

$$P = \alpha \bigotimes_{j=1}^{n} W_j, \quad \text{where each } W_j \in \{I, X, Y, Z\} \text{ acts on qubit } j, \quad \alpha \in \{\pm 1, \pm i\}.$$

The *Pauli group* $\mathcal{G}_n$ is the set of all such operators, equipped with operator multiplication:

$$\mathcal{G}_n := \left\{ \alpha \bigotimes_{j=1}^{n} W_j \;\middle|\; \alpha \in \{\pm 1, \pm i\},\ W_j \in \{I, X, Y, Z\} \right\}.$$

For $n = 1$, we have

$$\mathcal{G}_1 = \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}.$$

The *weight* of a Pauli operator is the number of qubits on which it acts nontrivially (i.e., with X, Y, or Z).

Some properties of $\mathcal{G}_n$:

- Each element squares to $\pm I_2$.
- Any two elements either commute or anticommute.

- Elements are either Hermitian or anti-Hermitian, depending on the phase.
- All elements are unitary.

To represent only the overall Pauli action without tracking global phases, we use *Pauli frames*. Each Pauli frame f is represented as a binary vector of length $2n$, where the first n bits indicate the presence of X components and the next n bits indicate the presence of Z components on each qubit. This encodes the tensor product structure via the mapping:

$$X^{a_j} Z^{b_j} \mapsto f = (a_j, b_j), \quad \text{where } a_j, b_j \in \{0,1\}.$$

For example, $Y = iXZ$ corresponds to $(a_j, b_j) = (1,1)$, and thus to the frame bit $f_j = (1,1)$.

We denote by $\mathcal{F}_n$ the set of Pauli frames on n qubits. Given a frame $f = (a_1, \ldots, a_n, b_1, \ldots, b_n) \in \mathcal{F}_n$, the corresponding generalized Pauli operator with unit phase is denoted $P(f)$, and defined as:

$$P(f) = \bigotimes_{j=1}^{n} W_j, \quad \text{where} \quad W_j = \begin{cases} I & \text{if } (a_j, b_j) = (0,0), \\ X & \text{if } (a_j, b_j) = (1,0), \\ Z & \text{if } (a_j, b_j) = (0,1), \\ Y & \text{if } (a_j, b_j) = (1,1). \end{cases}$$

Each W_j acts on qubit j.

The set of such unit-phase Pauli operators forms an orthonormal basis (under the Hilbert–Schmidt inner product) for the space of linear operators on n qubits. We refer to this as the *Pauli basis*, and denote it by $\mathcal{P}_n \subset \mathcal{G}_n$.

In the following, we use the symbol P to denote either an element of the Pauli group $\mathcal{G}_n$ or of the Pauli basis $\mathcal{P}_n$, and we make this distinction explicit whenever needed.

Any random string of single qubit Pauli operators can be reduced by commutation relation to a member of the Pauli group.

1.2 Quantum operations

POVM measurements can be better understood within the framework of *quantum operations*, a general formalism capable of describing a wide variety of quantum processes.

Formally, a *quantum operation* $\mathcal{E}$ is a linear superoperator acting on the set of bounded operators $\mathcal{B}(\mathcal{H})$ of a Hilbert space $\mathcal{H}$:

$$\mathcal{E} : \mathcal{B}(\mathcal{H}) \to \mathcal{B}(\mathcal{H}')$$

This formalism is used to describe the evolution of density matrices. To ensure that $\mathcal{E}$ maps valid quantum states to valid quantum states, it must satisfy two essential properties:

1. **Complete positivity**, to ensure that even when the system is entangled with an ancillary system, the extended operation $\mathcal{E} \otimes \mathcal{I}$ still maps positive operators to positive operators.
2. **Trace non-increasing**, to preserve the probabilistic interpretation of quantum states. Trace-preserving operations represent deterministic processes and are called *quantum channels*.

See [3] for a complete discussion.

Quantum operations describe a wide range of processes. Any unitary evolution is a quantum operation.

More generally, quantum operations extend the notion of evolution to *open systems*, i.e., systems interacting with an environment.

Consider a register made of an open quantum system, initially in state ρ, interacting with an environment in state ρ_{env}. Assuming the universe (system + environment) starts in the product state $\rho \otimes \rho_{\text{env}}$, its joint unitary evolution U leads to the following map on the system:

$$\mathcal{E}(\rho) = \text{Tr}_{\text{env}} \left(U \left(\rho \otimes \rho_{\text{env}} \right) U^\dagger \right)$$

From the system's point of view, the evolution is not unitary. The resulting quantum operation is referred to as a quantum *noise channel*. The effect of a noise channel on a quantum system is typically quantified by the fidelity between the pre-channel and post-channel states $F(\rho, \mathcal{E}(\rho))$.

Quantum operations can also describe measurements. Non-selective measurements are trace-preserving quantum operations, because they are deterministic processes. Selective-measurements are trace decreasing quantum operations. The trace gets restored to one by the subsequent bayesian renormalization, which is not included in the quantum operation being non-linear.

1.2.1 Chi matrix representation

Another widely used representation of a quantum channel $\mathcal{E}$ is the *chi matrix* (or χ-matrix) representation. This expresses the channel in terms of a fixed operator basis $\{F_i\}$, often chosen to be the Pauli basis for qubit systems.

Given a complete operator basis $\{F_i\}_{i=0}^{d^2-1}$ for operators acting on a d-dimensional Hilbert space (for example, the set of Pauli operators $\{I, X, Y, Z\}$ for a single qubit), any quantum channel can be written as:

$$\mathcal{E}(\rho) = \sum_{i,j} \chi_{ij} \, F_i \rho F_j^\dagger, \tag{1.4}$$

where χ is a $d^2 \times d^2$ complex matrix called the *chi matrix*.

The χ-matrix fully characterizes the channel $\mathcal{E}$ relative to the chosen basis $\{F_i\}$. It satisfies the condition $\chi \succeq 0$, meaning that the matrix is positive semidefinite. This ensures that the channel is completely positive.

Moreover, the trace-preserving property of $\mathcal{E}$ is equivalent to the constraint:

$$\sum_{i,j} \chi_{ij} F_j^\dagger F_i = I.$$

Intuitively, the χ-matrix can be viewed as the matrix of coefficients describing the channel's action as a superoperator, expressed in the basis of conjugations by operators F_i.

Stochastic channels

The on-diagonal elements of χ-matrix correspond to the stochastic part of the channel. If the only non-zero elements are on-diagonal, the channel maps its input ρ to a new mixture of states and channel is said to be *diagonal* or *stochastic*. In this case the $\chi_i i$ elements must be a proper probability distribution to maintain normalization. A stochastic channel maps mixtures to mixtures (of possibly different states), so it is equivalent to a transition kernel from classical probability theory.

Stochastic quantum channels are important in the context of classical simulation. Suppose one wishes to track the evolution of an observable on a quantum state undergoing noisy dynamics. In principle, this requires evolving the full density matrix via matrix multiplication at each timestep, which involves a $2^n \times 2^n$ object for an n-qubit system and thus becomes exponentially costly in both time and memory.

However, if the noise is modeled by a stochastic quantum channel then it becomes possible to approximate expectation values via Monte Carlo sampling. Stochastic channels make it possible to

track the state by simply recording the accumulated error string, avoiding explicit density matrix evolution. Observables can then be computed efficiently by evaluating their quantum expectation values on the randomly evolved pure states and averaging over many samples.

Coherent channels

The off-diagonal elements correspond to the *coherent* part of the channel, that is the genuinely quantum part of the channel. A coherent channel is a channel where some of this element are non-zero. If the input ρ is a mixture it gets mapped to a coherent superpositions of states, not just a new mixture. The effect of this channel cannot be simulated by Montecarlo sampling and requires a full calculation of the evolution of the density matrix.

1.2.2 Operator-Sum Representation

By diagonalizing the χ-matrix any quantum operation $\mathcal{E}$ can also be represented in the so-called *operator-sum representation*, also known as the *Kraus representation*. In this form, the action of $\mathcal{E}$ on a density matrix ρ is given by:

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger,$$

where $\{E_k\}$ are called the *operation elements* or *Kraus operators* associated with $\mathcal{E}$.

The set of operation elements must satisfy the following condition:

$$\sum_k E_k^\dagger E_k \leq I,$$

with equality if and only if the quantum operation is trace-preserving.

1.2.3 Stochastic Pauli channels

A fundamental class of quantum noise models is given by *stochastic Pauli channels*, defined as

$$\mathcal{E}(\rho) = \sum_{P \in \mathcal{P}_n} p(P)\, P \rho P^\dagger,$$

where $p(P)$ is a probability distribution over the n-qubit Pauli group $\mathcal{P}_n$. On a single qubit, common examples include:

- **Bit flip channel:** applies X with probability p, i.e. $\mathcal{E}(\rho) = (1-p)\rho + pX\rho X$.
- **Phase flip channel:** applies Z with probability p, i.e. $\mathcal{E}(\rho) = (1-p)\rho + pZ\rho Z$.
- **Bit-phase flip channel:** applies Y with probability p, i.e. $\mathcal{E}(\rho) = (1-p)\rho + pY\rho Y$.

Another notable case is the *depolarizing channel*, which replaces the state with a uniform Pauli error (excluding the identity). For a single qubit, we denote the channel by $\mathrm{Dep}^{(1)}$:

$$\mathrm{Dep}^{(1)}(\rho) = (1-p)\rho + \frac{p}{3}\left(X\rho X + Y\rho Y + Z\rho Z\right).$$

The depolarizing channel is the only example of an *isotropic* single-qubit channel. Isotropic channels are characterized by being covariant with respect to all rotations of the qubit space they work on:

$$\mathcal{E}\left(U\rho U^\dagger\right) = U\mathcal{E}(\rho)U^\dagger.$$

In the multi-qubit setting, any stochastic Pauli channel whose probability distribution $p(P)$ depends only on the weight of the Pauli operator P is isotropic.

Among isotropic channels, two natural generalizations of the single-qubit depolarizing channel are:

- **Local depolarizing channel:** each qubit independently experiences single-qubit depolarizing noise, i.e.

$$\mathrm{Dep}^{(n)}_{\mathrm{local}} = \bigotimes_{i=1}^{n} \mathrm{Dep}^{(1)}_i,$$

  where $\mathrm{Dep}^{(1)}_i$ acts on qubit i. This corresponds to a product distribution over Pauli errors:

$$p(P) = \prod_{i=1}^{n} p_i(P_i), \quad P_i \in \{I, X, Y, Z\}.$$

- **Global depolarizing channel:** with probability $1 - p$, the state is preserved, and with probability p, it is replaced by the maximally mixed state:

$$\mathrm{Dep}^{(n)}_{\mathrm{global}}(\rho) = (1-p)\rho + p\frac{I}{2^n}.$$

  Equivalently, this can be expressed as a uniform stochastic Pauli channel over all non-identity n-qubit Pauli operators:

$$\mathrm{Dep}^{(n)}_{\mathrm{global}}(\rho) = (1-p)\rho + \frac{p}{4^n - 1} \sum_{\substack{P \in \mathcal{P}_n \\ P \neq I}} P\rho P.$$

1.2.4 Pauli twirling

Pauli twirling is a technique used to approximate an arbitrary quantum channel by a stochastic Pauli channel. Formally, the twirled channel $\mathcal{E}_P$ is defined as:

$$\mathcal{E}_P(\rho) = \frac{1}{|\mathcal{G}_n|} \sum_{P \in \mathcal{G}_n} P^\dagger \, \mathcal{E}(P\rho P^\dagger) \, P,$$

where $\mathcal{G}_n = \{I, X, Y, Z\}^{\otimes n}$ is the n-qubit Pauli group.

The result is a Pauli channel of the form:

$$\mathcal{E}_P(\rho) = \sum_{P \in \mathcal{G}_n} p(P) \, P\rho P,$$

where $p(P)$ is a probability distribution that depends on the original channel $\mathcal{E}$, but does not directly correspond to the diagonal terms of its χ-matrix.

The advantage of twirling is that it preserves the average fidelity of the original channel while transforming it into a diagonal map in the Pauli basis, allowing for classical simulations and easier analytical treatment.

1.2.5 No cloning theorem

The quantum operation formalism allows to express conveniently one the most remarkable features of quantum information: the *no cloning theorem*. The no-cloning theorem states that it is

impossible to create an exact copy of an arbitrary unknown quantum state with a single channel. Formally, there does not exist a universal cloning quantum operation $\mathcal{E}$ such that:

$$\mathcal{E}(\rho \otimes |0\rangle\langle 0|) = \rho \otimes \rho$$

for all density operators ρ acting on a Hilbert space $\mathcal{H}$.

It will be shown in the next section that the no cloning theorem have critical consequences on quantum error correction.

1.2.6 Quantum tomography

Suppose we are given an unknown quantum state ρ. If we only have a single copy of ρ we cannot reconstruct the original probability distribution. Moreover, due to the no-cloning theorem, we cannot create multiple identical copies from a single instance of ρ. However, if we begin with many copies of ρ (each one produced by the same source, without one being copied from another) we can statistically estimate its parameters through repeated measurements, and the procedure takes the name of *quantum state tomography*. We can also reconstruct a whole unknown quantum operation by an technique called *quantum process tomography*, based on preparing quantum system in known states, applying the unknown quantum operation and applying state tomography on the output.

1.3 Quantum error correction

This section adds [4] to its sources. To store a register in a reliable way, it is necessary to deal with noise. One of the most common approaches is *quantum error correction*. The scope of quantum error correction is recovering the original information after an error has occurred. To achieve such a task, we need some form of redundancy for the information to protect, so that after a noise process, the uncorrupted part can be used to restore the corrupted one by a *recovery* procedure. This redundancy is achieved by encoding the logical information into a larger Hilbert space, distributing it across multiple physical subsystems. An encoding modality of this kind is called a *quantum code*.

1.3.1 Quantum codes

In the literature (see [3] and [4]) a quantum code is defined as a subspace $\mathcal{C}$ of the Hilbert space $\mathcal{H}_P = (\mathcal{C})^{\otimes n}$ of n physical qubits. I prefer to give a slightly different definition:

A quantum code is a tuple

$$(\mathcal{C}, \mathcal{E})$$

where:

- $\mathcal{C} \subset \mathcal{H}_P$ is the *code space*, a subspace of the physical Hilbert space.
- $\mathcal{E} : \mathcal{H}_L \to \mathcal{C}$ is a linear, injective *encoding map* that embeds logical states into $\mathcal{C}$.

The vectors in $\mathcal{C}$ are called *codewords*. In my definition, the logical meaning of a codeword is uniquely defined given the code, while in the original definition different reading conventions may be applied to the same code. A logical Hilbert space $\mathcal{H}_L$ describing a k-qubit logical register will have 2^k basis states, each representing a logical state. Each logical state is mapped to a basis vector of the code space, so the code space has the same dimensionality as the underlying logical space.

A code is formally characterized by the triplet $[n, k, d]$:

- n is the number of physical qubits,

- k is the number of logical qubits encoded,
- d is the distance of the code.

The concept of *distance* will be defined in the following.

Encoding

The encoding map is physically realized by an *encoding circuit*. The quantum encoding suffers a serious limitation compared to the classical counterpart: redundancy cannot be achieved by cloning information because of the no-cloning theorem. Consider for example the 3-bit repetition code. In the classical case, the repetition code encodes a single bit by repeating it 3 times:

$$\mathcal{E}: \quad \begin{cases} 0 \mapsto 000 \\ 1 \mapsto 111 \end{cases}$$

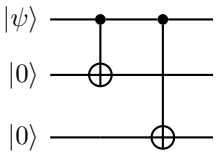
This can be done simply by copying the value of a bit. It is impossible to repeat the same mapping in the quantum case because of the no-cloning theorem. That is, it is impossible to achieve the map:

$$\mathcal{E}(|\psi\rangle) = |\psi\rangle\,|\psi\rangle\,|\psi\rangle$$

The equivalent code instead performs this mapping:

$$\mathcal{E}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) = \alpha\,|000\rangle + \beta\,|111\rangle$$

Such mapping is realized as follows. Starting from an arbitrary single-qubit state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ and two ancillary qubits initialized in $|0\rangle$, the encoding circuit applies CNOT gates with the original qubit as control and the two ancillary qubits as targets.



This pattern is ubiquitous in encoding circuits. Redundancy is achieved by entanglement of ancillary qubits to the original state by CNOTs. Since entangling ancillary qubits does not require measuring the system, entanglement allows redundancy to be achieved without collapsing the wave function.

Logical operators

A *logical error* is an operator of the physical space that acts nontrivially on the encoded information while preserving the code space.

A logical operator $\overline{L}$ satisfies

$$\overline{L} : \mathcal{C} \ \rightarrow \mathcal{C}$$

meaning it maps valid codewords to valid codewords but may change their logical value.

The basic types of logical errors are the one resulting in single qubit Pauli flips in the logical Hilbert space.

- **Logical $\overline{X}_i$ operators**: act as a logical bit flip on the i-th logical qubit.
- **Logical $\overline{Z}_i$ operators**: act as a logical phase flip on the i-th logical qubit.

- **Logical $\overline{Y}_i$ operators**: defined by $\overline{Y}_i = -i\overline{X}_i\overline{Z}_i$, act as a combined bit and phase flip.

Every logical operation can be implemented by tensoring and linearly combining these basic types with the identity.

A logical operation in the logical Hilbert space may be realized by one of different logical errors in physical space.

1.3.2 Error spaces

Consider the action of a noise channel $\mathcal{E}$ on a quantum state in the code space $\mathcal{C}$. Any noise channel can be expressed via its Kraus operators $\{E_i\}$:

$$\mathcal{N}(\rho) = \sum_i E_i \rho E_i^\dagger.$$

A Kraus element E_i of a noise channel acting on the physical Hilbert space is called an *error*. Applying an error E to a code state ρ maps the state into an *error space*, defined as

$$\mathcal{E}_E := E\mathcal{C} = \{E\,|\psi\rangle \mid |\psi\rangle \in \mathcal{C}\}.$$

Errors are classified according to their error space: if $\mathcal{E}_E$ lies outside the code space $\mathcal{C}$, the error is *detectable*.

If $\mathcal{E}_E$ coincides with $\mathcal{C}$, the error is *undetectable*. Undetectable errors include projectors (which act trivially on $\mathcal{C}$) and logical errors (which act nontrivially within $\mathcal{C}$).

In order to determine whether a detectable error has occurred, one can perform a *syndrome measurement*, which is a projective measurement that extracts information about the error by distinguishing subspaces of the Hilbert space.

In the most general case the syndrome measurement reduces to measuring the two-outcome projective measurement:

$$\{P_\mathcal{C},\, I - P_\mathcal{C}\},$$

where $P_\mathcal{C}$ is the projector onto the code space $\mathcal{C}$. The measurement outcome indicates whether the state remains inside the code space or has been mapped outside it.

By refining the syndrome measurement we can distinguish between *correctable* and *uncorrectable* sets of error.

1.3.3 Correctable sets of errors

Correctability is a property of a *set* of errors, rather than of a single error, because we must be able to identify which error occurred among a finite set in order to correct it. A *correctable set of errors* $\{E_i\}$ satisfies two crucial properties:

1. **Orthogonality of error spaces:** For distinct errors i and j in the set, the corresponding error spaces
$$\mathcal{E}_i = E_i\mathcal{C} \quad \text{and} \quad \mathcal{E}_j = E_j\mathcal{C}$$
must be mutually orthogonal subspaces of the physical Hilbert space.

2. **Preservation of inner products within each error space:** Within each error space $\mathcal{E}_i$, the inner product induced by the code space $\mathcal{C}$ is preserved up to a constant factor:
$$\langle\psi|\, E_i^\dagger E_i\, |\phi\rangle = \alpha_i \langle\psi|\phi\rangle \quad \text{for all } |\psi\rangle, |\phi\rangle \in \mathcal{C}.$$

These conditions are summarized in the Knill-Laflamme conditions: a set of errors $\{E_i\}$ is correctable for code $\mathcal{C}$ if and only if

$$\langle\psi| E_i^\dagger E_j |\phi\rangle = \alpha_{ij} \langle\psi|\phi\rangle \quad \text{for all } |\psi\rangle, |\phi\rangle \in \mathcal{C}. \tag{1.5}$$

The orthogonality of the error spaces $\mathcal{E}_i$ allows a user to distinguish which error occurred by a syndrome measurement. The measurement corresponds to the POVM of projectors onto the error spaces:

$$\{P_i\}, \quad P_i = \text{projector onto } \mathcal{E}_i.$$

Because of orthogonality, the outcome uniquely identifies the error space i, and thus the error that occurred.

Since we are measuring projectors over orthogonal spaces, the measurement consists of pure Bayes update of the density matrix of the system and the post-measurement state is equal to the hidden pre-measurement state. To return to the code space, we simply apply E_i^{-1}, which is always possible because orthogonality implies that E_i acts as an invertible map on $\mathcal{C}$.

This procedure allows exact correction of a channel whose Kraus operators form a correctable set $\{E_i\}$. It requires knowing the channel exactly, since the projectors P_i must be specified.

While this is rarely the case in practice, the procedure can often be extended.

Discretization of errors

Consider a quantum channel with Kraus operators $\{F_i\}$ acting on the code space. Each F_i can be expanded in a correctable operator basis $\{E_a\}$:

$$F_i = \sum_a \alpha_{ia} E_a.$$

We measure the projectors on the error spaces of $\{E_s\}$, measuring the syndrome, and the density matrix evolves according to the non-selective measurement

$$\rho \mapsto \sum_{i,s} \Pi_s F_i \rho F_i^\dagger \Pi_s = \sum_{i,s,a,b} \alpha_{ia}\alpha_{ib}^* \, \Pi_s E_a \rho E_b^\dagger \Pi_s = \sum_{i,s} |\alpha_{is}|^2 \, E_s \rho E_s^\dagger,$$

We see the Π_s projects onto the subspace corresponding to the basis. This shows that the measurement eliminates the off-diagonal terms, leaving a stochastic mixture over the discrete error basis. At this point observing the syndrome a identifies the E_a by Bayes update and we can correct it. Even if E_a was not the actual error that occurred, the measurement has collapsed the real error into it. This is the *discretization of errors*, which allows correction of arbitrary channels as long as the operator basis $\{E_a\}$ is correctable for the code space. This works even if the F are not orthogonal.

1.3.4 Uncorrectable sets of errors

All previous procedures implicitly assume that we can always measure the projectors we need for a custom error set. Experimentally, this is not the case: the physical setup determines which projectors are actually available. While for each set of orthogonal error spaces there exists a projective measurement capable of perfectly distinguishing which error occurred, in practice the available projectors are typically not ideal. Instead, the measured projectors project on unions of orthogonal error spaces. In terms of projectors, the Knill-Laflamme conditions can be written as:

$$P_s E_a \rho E_a^\dagger P_s = \lambda_{a,s} P_s,$$

for each projector label s. This may not be verified by the available projectors even if 1.5 is.

As a consequence, the probability of obtaining a syndrome s given an error E_a is

$$\Pr(s|E_a) = \mathrm{Tr}\big(P_s E_a \rho E_a^\dagger\big),$$

which is generally not a Kronecker delta. This implies that after measurement, the system is projected into a mixture of states corresponding to multiple errors, rather than a single pure post-error state. We are therefore unable to recover the exact state deterministically.

Identifying the actual error space from the measured syndrome is an inferential task known as *decoding*. Importantly, in quantum error correction decoding is not the inverse of encoding; it is a fundamentally different operation.

We decided to be clear about this issue because much of the literature emphasizes the non-orthogonality of error spaces. In reality, any set of Kraus operators can be decomposed into an orthogonal basis. The stabilizer codes we analyze do have orthogonal error spaces. The challenge arises from the degeneracy of the available projective measurements: each measured syndrome typically corresponds to multiple underlying errors. Decoding is therefore required to infer the most likely error given the observed syndrome, highlighting the probabilistic and inferential nature of error correction.

1.3.5 Recovery

Quantum error correction is the procedure of identifying a suitable *recovery channel* to mitigate the effects of noise.

Any error E may be corrected by an operator R such that

$$R \circ E\, P_{\mathcal{C}} = P_{\mathcal{C}},$$

meaning that the combination of E followed by R returns a codeword in the code space $\mathcal{C}$ to its original value. This is an example of a recovery operator for E. In general, any operator applied with the intention of mitigating errors can be considered a recovery operator, even if it does not immediately return the state exactly to the code space.

In practice, we do not know which error occurred, only the probabilities of different errors. A *recovery channel* $\mathcal{R}$ is therefore any channel that measures the syndrome s and applies a conditional recovery $R(s)$ based on the observed syndrome.

The performance of decoding is evaluated using the fidelity between the original state and the recovered state:

$$F\left(\rho,\ \mathcal{R} \circ \left(\frac{P_s\, \mathcal{E}(\rho)\, P_s}{\mathrm{Tr}\,(P_s\, \mathcal{E}(\rho))}\right)\right),$$

where P_s is the projector onto the subspace corresponding to syndrome s, and $\mathcal{E}$ is the noise channel. This fidelity quantifies the effectiveness of the decoding algorithm.

Thus, the lifecycle of encoded quantum information consists of iterating the cycle of noise, syndrome measurement, decoding, recovery, and repeating as long as necessary.

1.3.6 Distance of a code

We define the *weight* of an error on an n-qubit system as the number of non-identity single-qubit operators in its tensor-product decomposition.

The *distance* d of a quantum code for a given error set E is the minimum weight of a logical error in E.

The distance of a code is a critical parameter in determining its error-correcting capability. A code with distance d can detect all errors acting on up to $d-1$ qubits, while any error affecting d qubits may result in a logical error.

1.3.7 Faulty syndromes

When performing quantum error correction, the measurement of the syndrome may itself be faulty. This happens because, in practice, the measurement is performed by a small quantum circuit that is itself subject to quantum noise. In particular, the circuit sets the state of an ancilla qubit that gets measured in the syndrome extraction process. Errors can happen as quantum operations on the ancilla qubits or on other elements. As a result, the observed syndrome may not correspond to the actual error that occurred on the data qubits. Tackling this problem is the purpose of this thesis.

2 From stabilizer formalism to surface codes

This chapter covers quantum computing prerequisites not usually known to reader without specific preparation on error correction.

2.1 Stabilizer codes

This section draws from [4], but the material has been substantially reworked and reframed. The *stabilizer formalism* provides a compact and powerful way to describe certain quantum spaces, operations, and codes.

2.1.1 Group theory

Stabilizer formalism makes use of group theory, so we start the section by a recap of group theory results. The concepts of centralizers and normalizers are fundamental tools to understand the structure of groups and their subgroups.

Centralizer

The *centralizer* of an element g in a group G is the set of all elements in G that commute with g. Formally, it is defined as:

$$C_G(g) = \{x \in G \mid xg = gx\}.$$

The set $C_G(g)$ forms a subgroup of G, called the *centralizer subgroup* of g.

More generally, given a subgroup $S \subseteq G$, the centralizer of S in G is the set of elements of G that commute with every element of S:

$$C_G(S) = \{x \in G \mid xs = sx \quad \text{for all} \quad s \in S\}.$$

This again defines a subgroup of G.

Normalizer

Given a group G and a subgroup $S \subseteq G$, the *normalizer* of S in G is the set of elements in G that conjugate S into itself:

$$N_G(S) = \{g \in G \mid gSg^{-1} = S\}.$$

That is, an element $g \in G$ belongs to $N_G(S)$ if conjugation by g leaves every element of S inside S .

Notice that the normalizer is an extension of the centralizer. The centralizer of a subgroup is the set of elements that conjugate the the element into self, while the normalizer's elements just conjugate into the group.

So in general :

$$C_G(S) \subseteq N_G(S)$$

In quantum information, especially for stabilizer codes, the relevant group G is the Pauli group G_n, and S is an abelian subgroup of G_n (the stabilizer group) that does not contain $-I$.

Under this hypothesis :

$$N_{G_n}(S) = C_{G_n}(S).$$

Here the simple proof: since every element in G_n either commutes or anticommutes, for every $g \in N_{G_n}(S)$:

$$gsg^{-1} = \pm s.$$

However, $gsg^{-1} \in S$ must exactly match some element of S, and S does not contain $-s$. If $-s \in S$ then $-ss^\dagger = -I \in S$ that leads to contradiction. Therefore, $gsg^{-1} = s$ for all $s \in S$.

Thus, g commutes with every $s \in S$, meaning $g \in C_{G_n}(S)$.

Clifford group

The Clifford group $\mathcal{C}_n$ is defined as the normalizer of the Pauli group $\mathcal{P}_n$ in the unitary group $U(2^n)$

$$\mathcal{C}_n = \{U \in U(2^n) \mid U\mathcal{P}_n U^\dagger \subseteq \mathcal{P}_n\}.$$

It is thus the set of unitary gates on n qubits that map Pauli operations to Pauli operations. A n-qubit Clifford group is generated by three types of gates: two single qubit gates, the Hadamard gate H and the phase gate S, and the two qubit CNOT gate.

2.1.2 Stabilizer formalism for vector spaces

The *Pauli group* $\mathcal{G}_n$ is the multiplicative group of n-qubit Pauli operators. For a single qubit,

$$\mathcal{G}_1 = \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}.$$

Some properties:

- Each member squares to ± 1.
- Pauli operators either commute or anticommute with each other.
- Pauli operators are either Hermitian or anti-Hermitian, based on the scalar prefactor.
- All elements of $\mathcal{G}_n$ are unitary.

Consider a subgroup $\mathcal{S} \subseteq \mathcal{G}_n$. The group $\mathcal{S}$ is called a (pointwise) *stabilizer group* if there exists a subspace $V_\mathcal{S} \subseteq \mathcal{H}_n$ of the n qubits Hilbert space $\mathcal{H}_n$ such that every vector in $V_\mathcal{S}$ is invariant under all operators in $\mathcal{S}$. We say that $V_\mathcal{S}$ is stabilized by $\mathcal{S}$.

For $\mathcal{S}$ to stabilize a non-trivial subspace, it is necessary and sufficient that:

- $\mathcal{S}$ is commutative,
- $-I \notin \mathcal{S}$.

These conditions also imply that all stabilizers are Hermitian. We will assume these conditions for every stabilizer group in the following.

Dimensional Counting

Suppose $\mathcal{S}$ is generated by l independent commuting elements:

$$\mathcal{S} = \langle g_1, g_2, \ldots, g_l \rangle.$$

Then:

Proposition 2.1.1. *If the stabilizer group $\mathcal{S}$ has l independent generators, the stabilized subspace $V_{\mathcal{S}}$ has dimension 2^k, where $k = n - l$.*

A stabilized space can be used as a code space. This is what is done in an important class of quantum codes called *stabilizer codes*.

2.1.3 Definition and properties

Definition 1. *A* stabilizer code *is a subspace $\mathcal{C} \subseteq \mathcal{H}_n$ stabilized by a subgroup $\mathcal{S} \subseteq \mathcal{G}_n$ satisfying:*

- *$-I \notin \mathcal{S}$,*
- *$\mathcal{S}$ has $n - k$ independent, commuting generators.*

Such a code is an $[n, k]$ code. A proper codeword is thus a joint $+1$ eigenvector of all generators of $\mathcal{S}$.

Logical operators

The effect of Pauli operators on codewords can be understood by group theory. For a stabilizer group $\mathcal{S}$, under the hypotheses made up to now, the *normalizer* $\mathcal{N}(\mathcal{S})$ of $\mathcal{S}$ in $\mathcal{G}_n$ consists of all elements in $\mathcal{G}_n$ that commute with every element of $\mathcal{S}$:

$$\mathcal{N}(\mathcal{S}) = \{E \in \mathcal{G}_n \mid [E, M] = 0 \quad \forall M \in \mathcal{S}\}.$$

For any operator $E \in \mathcal{N}(\mathcal{S})$ and any codeword $|\psi\rangle \in \mathcal{C}$,

$$\forall S \in \mathcal{S}, \quad S(E|\psi\rangle) = E(S|\psi\rangle) = E|\psi\rangle.$$

Thus, $E|\psi\rangle$ is also stabilized by every $S \in \mathcal{S}$, and therefore $E|\psi\rangle \in \mathcal{C}$.

This means that the operators in the normalizer are either the identity (if they belong to the stabilizer) or non trivial logical operators (if they belong to $\mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$).

2.1.4 Coset structure of the errors

Two operators $E, E' \in \mathcal{G}_n$ are equivalent, that is have the same action on every codeword, if and only if they differ by an element of the stabilizer:

$$E \sim E' \quad \text{if and only if} \quad E = E'S, \quad S \in \mathcal{S}.$$

We can thus define the quotient:

$$\mathcal{G}_n/\mathcal{S},$$

which is the set of *error classes*, where each error class is a coset of the form

$$[E] = E\mathcal{S} = \{ES \mid S \in \mathcal{S}\},$$

for some representative $E \in \mathcal{G}_n$.

This structure also extends to the normalizer. We can organize $\mathcal{N}(\mathcal{S})$ into cosets of the form $M\mathcal{S}$, with $M \in \mathcal{N}(\mathcal{S})$, by considering the quotient

$$\mathcal{L} = \mathcal{N}(\mathcal{S})/\mathcal{S}.$$

$\mathcal{L}$ is the set of *logical operations* acting on the code space.

2.1.5 Error Spaces

The code space of a stabilizer code consists of the vectors in $\mathcal{H}_n$ that are simultaneous $+1$ eigenvectors of all the stabilizers. Let us now consider the other eigenvectors. Since the stabilizers are Hermitian and commute with each other, their simultaneous eigenvectors corresponding to different eigenvalues are orthogonal. This means that the eigenspaces of the stabilizers are orthogonal and can thus serve as *syndrome spaces*.

Each error E in the Pauli group can be assigned to a syndrome $s(E)$ by the commutation relations:

$$s(E) = ((-1)^{m_1}, \ldots, (-1)^{m_k}), \quad \text{where} \quad ES_i = (-1)^{m_i} S_i E.$$

This is because the subspace $E\mathcal{C}$ coincides with the syndrome subspace corresponding to the syndrome $s(E)$.

Thus, every Pauli error is either a stabilizer, a logical operation, or a detectable error. In particular, an error is detectable if $E \notin \mathcal{N}(\mathcal{S})$.

The normalizer $\mathcal{N}(\mathcal{S})$ consists precisely of the Pauli operators that commute with all stabilizers, i.e., those for which $s_i(E) = 0$ for all i.

Error detection is achieved by measuring the stabilizers.

Two errors have the same syndrome if and only if they differ by an element of the normalizer:

$$s(E_1) = s(E_2) \quad \Longleftrightarrow \quad E_1^\dagger E_2 \in \mathcal{N}(\mathcal{S}) \tag{2.1}$$

To prove this, we first notice that

$$s(E_1^\dagger E_2) = s(E_1) + s(E_2) \mod 2,$$

which follows from the facts that the stabilizers S_i are Hermitian, implying

$$s(E^\dagger) = s(E),$$

and from the calculation:

$$\begin{aligned}
(E_1E_2)S_i &= E_1(E_2S_i) \\
&= E_1\left((-1)^{s_i(E_2)} S_i E_2\right) \\
&= (-1)^{s_i(E_2)}(E_1 S_i E_2) \\
&= (-1)^{s_i(E_2)}\left((-1)^{s_i(E_1)} S_i E_1 E_2\right) \\
&= (-1)^{s_i(E_1)+s_i(E_2)} S_i(E_1E_2).
\end{aligned}$$

Thus,

$$s_i(E_1E_2) = s_i(E_1) + s_i(E_2) \mod 2,$$

and in vector notation:

$$s(E_1E_2) = s(E_1) + s(E_2) \mod 2.$$

Now for the main statement:

$\Rightarrow$ Suppose $s(E_1) = s(E_2)$. Then $s(E_1^\dagger E_2) = s(E_1) + s(E_2) = 0$, so $E_1^\dagger E_2 \in \mathcal{N}(\mathcal{S})$.

$\Leftarrow$ Conversely, if $E_1^\dagger E_2 \in \mathcal{N}(\mathcal{S})$, then $s(E_1^\dagger E_2) = 0$, and thus $s(E_1) = s(E_2)$.

Thus, we can take the quotient:

$$[s] = \{EN \mid N \in \mathcal{N}(\mathcal{S})\} = \{ELS \mid L \in \mathcal{L},\, S \in (\mathcal{S})\}$$

I will refer to these cosets as *syndrome classes*.

Thus, the set of errors corresponding to a given syndrome s is partitioned into logical sectors, each sector being a coset of the stabilizer group shifted by a logical operator.

2.1.6 Decoding and recovery

Decoding and recovering follow general patterns in stabilizer codes.

Consider the simplest case to fix ideas:

1. The system starts in a pure codeword state.
2. The noise channel is composed of just one Pauli error E, so that the state becomes
$$E\left|\psi\right\rangle\left\langle\psi\right|E^{\dagger}.$$
3. We consider a recovery channel composed of just one operation element R.

The goal of recovery is to choose R such that

$$RE \in \mathcal{S}.$$

In order to achieve this, it is necessary to identify the error class $[E]$, so that we can choose $R \in [E]$.

The only information available to the decoder is the syndrome class s.

Therefore, the decoder will choose a Pauli operator R_s that belongs to the syndrome class corresponding to the measured syndrome s.

As shown previously, this guarantees only that

$$R_s E \in \mathcal{N}(\mathcal{S}).$$

Thus, to recover correctly, we must choose

$$R_s = E'^{\dagger} S,$$

for some $E' \in [E]$ and $S \in \mathcal{S}$.

If instead we choose

$$R_s = L E'^{\dagger} S$$

where $L \in \mathcal{L}$ is a nontrivial logical operator, then the recovery will result in a logical error.

Choosing the right operator is an inference task. These considerations naturally extend to the case of a general noise channel, a general initial state, and a general recovery channel. In the general case, fidelity is used to evaluate the decoding.

2.1.7 Coherent errors

Up to now we just talked about Pauli errors. In general, not every error is a Pauli error. Since every operator can be decomposed in the Pauli basis, whenever an error is not Pauli it must correspond to a coherent superposition of Pauli error, so we talk of *coherent errors*. In principle, stabilizer formalism does not tackle coherent errors. We will see that anyway stabilizer codes can offer *some* protection against coherent errors, but not fully manage them.

2.2 Topological codes

This section is based on [5]. Topological codes are an important class of stabilizer codes because they provide an experimentally feasible and scalable way to do fault-tolerant quantum computing. The mathematical framework that allows the construction of every topological code is the so-called Z_2 *chain complex*.

2.2.1 Z_2 chain complex

Chains

Consider a lattice G over some manifold. Every lattice can be regarded as a graph on i-dimensional *cells* for every i between 0 and the manifold dimension of the lattice. For example:

- *0-dimensional cell* ($i = 0$): A 0-dimensional cell is a point e_k^0.
- *1-dimensional cell* ($i = 1$): A 1-dimensional cell is an edge e_k^1.
- *2-dimensional cell* ($i = 2$): A 2-dimensional cell is a face e_k^2.

Given a cell dimension, we can build a vector space C^i over the Z_2 field by considering the cells as a basis. A vector element is called an i-chain.

$$c^i = \sum_k z_k e_k^i$$

For example, a 0-chain is a set of vertices:

$$c^0 = \sum_k z_k e_k^0$$

where the weight z_k specifies which vertex belongs to the chain. The sum operation in this vector space is the binary sum of the scalars, meaning that two bases cancel each other out when summed.

Boundary operator

We can now introduce the homomorphism ∂_i, called the *boundary operator*. The boundary operator $\partial_i : C^i \to C^{i-1}$ is defined as the operator returning an $(i-1)$-chain that is the boundary of the c^i argument.

For example, $\partial_1 e_l^1$ is a 0-chain for which $z_k = 1$ if and only if e_k^0 is an endpoint of the edge e_l^1.

A chain c^i is called a *cycle* if it is in the kernel of the boundary operator: $\partial_i c^i = 0$.

For example, the boundary $\partial_2 e_l^2$ of the l-th face is a cycle, because $\partial_1 \circ \partial_2 e_l^2 = 0$.

A property of the boundary operator is that every boundary is a cycle, so

$$\partial_{i-1} \circ \partial_i = 0.$$

The reverse does not hold.

Definition of Z_2 chain complex

A vector space C^i is also an abelian group relative to the sum operation. A sequence of abelian groups C^i, connected by the boundary operators ∂_i with $\partial_{i-1} \circ \partial_i = 0$, is called a Z_2 *chain complex*.

Homology classes

A cycle that is the boundary of a chain one dimension higher is called a *trivial cycle*:

$$c^i = \partial c^{i+1}.$$

Trivial cycles define an equivalence relation among i-cycles: two i-cycles c_1^i and c_2^i are said to be *homologically equivalent* if there exists an $(i+1)$-chain c^{i+1} such that

$$c_1^i = c_2^i + \partial c^{i+1}.$$

The set of equivalence classes forms the *homology group* h_i, defined as the quotient

$$h_i = \ker(\partial_i) / \operatorname{Im}(\partial_{i+1}),$$

An element of h_i is called a *homology class*.

Dual lattice

From now on, we will restrict ourselves to lattices embedded on a surface. Let's switch notation so that any vertex element is denoted by v_k, any edge by e_k, and any face by f_k. Given a lattice $G = (V, E, F)$ on a 2D manifold, we construct the *dual lattice* $\bar{G} = (\bar{V}, \bar{E}, \bar{F})$ by the following procedure:

- For each face $f \in F$ of the primal lattice, we define a dual vertex $\bar{v} \in \bar{V}$.
- Two vertices are connected by an edge in the dual graph if the primal faces share a common primal edge. Since two faces can only share an edge and an edge can only be shared by two faces, the edges of the primal lattice are mapped to dual edges. This means that the dual edge $\bar{e}_m$ crosses the primal edge e_m, and the primal and dual graphs intersect transversally on the same surface.
- Automatically, each vertex $v \in V$ of the primal lattice corresponds to a dual face $\bar{f} \in \bar{F}$: the vertex v whose edge neighborhood is mapped to the boundary of $\bar{f}$ is the vertex mapped to it.

We can construct a Z_2 *chain complex* on $\bar{G}$ too, using the dual vector spaces $\bar{C}^i$. Of course, the only real lattice is the primal one. The elements of the chain complex of the dual lattice are can be mapped to elements of the primal one and the relations involving elements of the dual lattice should be interpreted as relations involving the corresponding elements of the primal lattice.

The duality between the primal and dual lattices induces the following mapping between the bases:

$$B(C^0) \longrightarrow \bar{B}(C^2), \quad B(C^1) \longrightarrow \bar{B}(C^1), \quad B(C^2) \longrightarrow \bar{B}(C^0),$$

In the following the notation $\bar{c}^1$ and n c^1 refer to independent chains of the primal and dual lattice, not a primal lattice and its own dual.

Algebraic relations

The inner product of two chains is defined by $c_1 \cdot c_2 \equiv \sum_l z_l z'_l$, where the addition is taken modulo 2.

Let $M(\partial_i)$ denote the matrix representation of the boundary operator ∂_i, expressed with respect to the bases $B(C^i)$ and $B(C^{i-1})$. This matrix representation must be interpreted relative to the binary matrix product.

Like for any scalar product, for any i-chain $c^i_k \in C^i$ and $(i-1)$-chain $c^{i-1}_l \in C^{i-1}$, we have the adjoint relation:

$$\left(M(\partial_i)c^i\right) \cdot c^{i-1} = c^i \cdot \left(M(\partial_i)^T c^{i-1}\right). \tag{2.2}$$

Considering the relations with the dual lattice:

$$M(\partial_1) = M(\bar{\partial}_2)^T, \quad M(\partial_2) = M(\bar{\partial}_1)^T. \tag{2.3}$$

This can be interpreted as an algebraic description of the duality relation. For example, consider the first equation: $M(\partial_1)_{ij}$ represents whether the edge j is bounded by node i. This is equivalent

to saying that the dual face i is bounded by dual edge j, and because edges are input in the left term and output in the right term, there is a transposition.

Since $\partial_1 \circ \partial_2 = 0$, it follows that:

$$M(\bar{\partial}_2)^T M(\partial_2) = 0. \tag{2.4}$$

We can now prove an identity showing that every primal lattice surface c^2 boundary has zero net overlap with any dual lattice surface $\bar{c}^2$ boundary:

$$\bar{\partial}_2 \bar{c}^2 \cdot \partial_2 c^2 = \bar{c}^2 M(\bar{\partial}_2)^T M(\partial_2) c^2 = 0. \tag{2.5}$$

Moreover, for any primal 1-chain c^1 and dual 2-chain $\bar{c}^2$, we have the relation:

$$c^1 \cdot \bar{\partial}_2 \bar{c}^2 = \partial_1 c^1 \cdot \bar{c}^2 \tag{2.6}$$

which is proved using (2.2) and (2.3).

2.2.2 Pauli error chains

We can embed a quantum system in a lattice and use the Z_2 chain complex to define operators. Consider a lattice G embedded in a surface manifold. We can associate a qubit to every edge e.

Generalized Pauli operators can be represented by the chain composed of the qubits they act on. A homogeneous Pauli operator is a Pauli operator in which the same single qubit Pauli operator acts on every qubit. The single qubit Pauli operator defines the *type* of the homogeneous Pauli operator.

For example a X-operator, given $c^1 = \sum_l z_l e_l$ (or dual 1-chain $\bar{c}^1$) is defined as:

$$X(c^1) = \prod_l X_l^{z_l}$$

Let W be a type. Since Pauli operators are idempotent, the following property holds for the associated homogeneous Pauli operator:

$$W(c_1^1) W(c_2^1) = W(c_1^1 + c_2^1). \tag{2.7}$$

Commutation relations

Two Pauli operators of the same type commute. The commutability between an operator $X(c_1^1)$ and an operator $Z(c_2^1)$ depends on the number of times the two chains act on the same qubit. If the chains overlap in an even number of spots, then the two products commute. Otherwise, they anticommute. This can be expressed by the following scalar products:

$$c_1^1 \cdot c_2^1 = 0, \text{ iff } X(c_1^1)Z(c_2^1) = Z(c_1^1)X(c_2^1) \quad \text{(commute)} \tag{2.8}$$

$$c_1^1 \cdot c_2^1 = 1, \text{ iff } X(c_1^1)Z(c_2^1) = -Z(c_1^1)X(c_2^1) \quad \text{(anticommute)} \tag{2.9}$$

Standard form of a Pauli operator

Since $Y = iXZ$, using these commutation relations we can write any Pauli operator as a product of a homogeneous $X-$chain supported on a dual chain $\bar{c}_i^1$ and a homogeneous $Z-$chain supported on a primal chain c_i^1.

From now on, c refers to 1-chains, f to 2-chains and v to 0-chains, so we can drop the superscripts. The subscript indicating dimensionality in ∂ is dropped too, because it will always be clear from context.

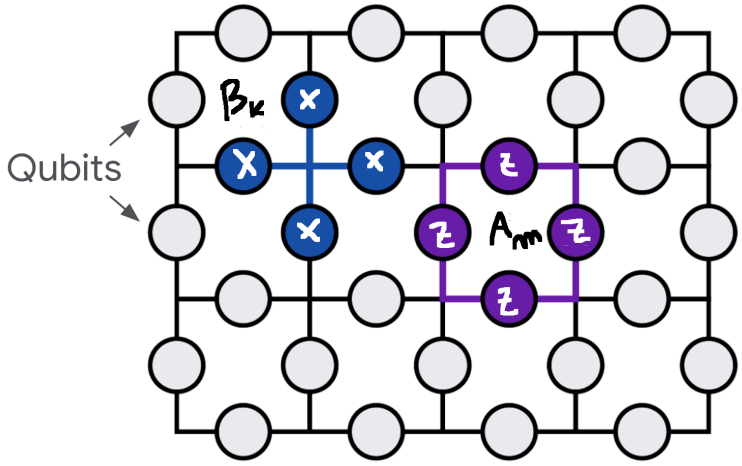


Figure 2.1. Section of the square lattice of a surface code. The Z and X stabilizer generators are defined by being associated with each face and vertex (dual face)

2.2.3 Square toric surface codes: definition

Consider a qubit distribution over the edges of a square lattice on a toroidal surface.

Stabilizers

We define stabilizers of the Z- and X-types for each face f_m and vertex v_k as follows:

$$A_m = Z(\partial f_m), \quad B_k = X(\delta v_k) = X(\partial \bar{f}_k). \tag{2.10}$$

Where δv_k indicate the set of links entering in v_k.

Because of (2.5), A_m and B_k commute.

The stabilizers A_m and B_k are called *plaquette* and *star* operators respectively, because of the shape of the edges configuration they act on. A schematic representation of the lattice and the stabilizer is provided in Fig. 2.1.

The code associated with such stabilizers is called a *square toric surface code*. The stabilizers A_m and B_k are not all independent. This dependence stems from the fact that in the toric code the lattice forms a closed surface, and each edge is shared by exactly two faces and two vertices.

Consider the product of all plaquette stabilizers:

$$\prod_m A_m = \prod_m Z(\partial f_m). \tag{2.11}$$

Each edge appears exactly twice in the combined boundaries of the faces (once for each adjacent face). Since $Z^2 = I$ for Pauli operators, the contribution from each edge cancels out, and thus

$$\prod_m A_m = I. \tag{2.12}$$

This means that any plaquette A_k can be obtained by the others as:

$$A_k = \prod_{m \neq k} A_m. \tag{2.13}$$

Same happens for star operators in the dual lattice. Thus one plaquette stabilizer and one star stabilizer must be omitted to form an independent set. We thus need to exclude one plaquette and one star to have the proper stabilizer generators. Now that we have the generators we can ask what is the general form of a member of the stabilizer group. Any element of the stabilizer group is a product of plaquette and star operators. Let R be the collection of faces spanned by the plaquette operators in the product. By the product rule (2.7), the product of Pauli operators corresponds to the Pauli operator associated with the sum of the underlying chains. Since the boundary operator ∂ is linear, the sum of the boundaries satisfies $\sum_{m\in R}\partial f_m = \partial\left(\sum_{m\in R} f_m\right)$, Thus, a product of plaquette operators over R can be rewritten as

$$\prod_{m\in R} A_m = Z\left(\sum_{m\in R}\partial f_m\right) = Z\left(\partial\left(\sum_{m\in R} f_m\right)\right).$$

That is, it is the Pauli Z operator associated with the boundary of the R region. Similarly, a product of star operators corresponds to a Pauli X operator acting on a trivial cycle in the dual lattice.

The set of stabilizer thus coincide with the products of $Z-$operators on a trivial cycle of the primal lattice and a X-operator on a trivial cycle of the dual lattice. Any two Pauli products supported on chains that differ by a boundary act identically on the code space:

$$Z(c+\partial f) = Z(c)S, \tag{2.14}$$

where $S\in\mathcal{S}$ is an element of the stabilizer group.

Normalizer group

A Pauli chain belongs to the normalizers if and only if its Z-part, supported on 1-chain c_z commutes with all the $X-$chains supported on trivial cycles of the dual lattice (i.e. any sum of star operators), and the X-part, supported on 1-chain $\bar{c}_x$, commutes with all the $Z-$chains supported on trivial cycles in the primal lattice (i.e. any sum of plaquette operators).

Using the commutation condition and 2.6, this means:

$$0 = c_z\cdot\bar{\partial}\bar{f} = \partial c_z\cdot\bar{f} \quad \text{for all dual 2-chains} \quad \bar{f}$$

$$0 = \bar{c}_x\cdot\partial f = \bar{\partial}\bar{c}_x\cdot f \quad \text{for all primal 2-chains} \quad f.$$

To prove the second equation, one must take the dual of 2.6.

This implies $\partial c_z = 0$ and $\bar{\partial}\bar{c}_x = 0$: the Z support must be a primal cycle or blank, the X support must be a dual cycle or blank.

We can thus decompose the normalizer into a direct product of two commuting subgroups:

$$\mathcal{N}(S) = \mathcal{N}_Z\times\mathcal{N}_X,$$

where $\mathcal{N}_Z$ consists of Z-type operators supported on cycles of the primal lattice, and $\mathcal{N}_X$ consists of X-type operators supported on cycles of the dual lattice. We can perform the same split for the stabilizer. Since two Pauli operators of the same type are logically equivalent if their support differ by a trivial cycle, if we take the cosets:

$$[\overline{Z}] = \mathcal{N}_Z/\mathcal{S}_Z \quad \text{and} \quad [\overline{X}] = \mathcal{N}_X/\mathcal{S}_X$$

any logical operation can be represented by a tensor product of a Z coset and a X coset. By identifying a Pauli operator with its support we see that both of these quotient groups are isomorphic to the homology classes of 1-chains on a torus, shown in Fig. 2.2:

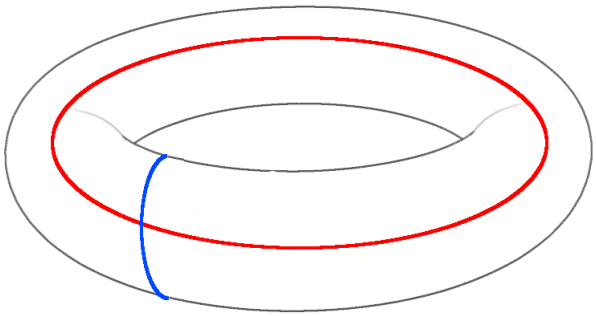


Figure 2.2. Representatives for the two non trivial homology classes of 1-chains on a torus. In blue a meridian cycle, in red a longitude.

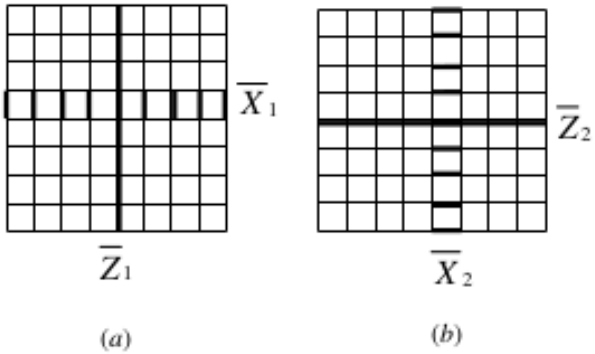


Figure 2.3. Representation of the logical Z and X operators. Boundary conditions are periodic, meaning the depicted chains wrap around the torus.

Let c_1 be primal lattice cycle of the meridian class, $\bar{c}_1$ a dual chain with axis aligned on longitude class and vice versa for c_2, $\bar{c}_2$.

We can thus define two independent pairs of logical Pauli operators:

$$\{\overline{Z}_1 = Z(c_1), \quad \overline{X}_1 = X(\bar{c}_1)\}, \quad \{\overline{Z}_2 = Z(c_2), \quad \overline{X}_2 = X(\bar{c}_2)\},$$

represented in Fig. 2.3

In order to be well-defined Pauli operators they must respect the canonical Pauli commutation relation.

Since every meridian crosses every longitude once and only once and c_1 and $\bar{c}_1$ are on different homology classes, then:

$$\overline{Z}_i\overline{X}_j = (-1)^{\delta_{ij}}\overline{X}_j\overline{Z}_i,$$

matching the structure of two logical Pauli operators. Keeping the correct commutation relations is the reason why the chains corresponding to different types of Pauli operator for the same logical qubit must be on different homology classes. Since we have two couples of logical X and Z, it means a toroidal lattice can host two logical qubits. A toroidal lattice with n qubits on

meridian and longitude circles has n^2 qubits. The corresponding code is thus a $[n^2,2,n]$ stabilizer code (see 1.3.1).

2.2.4 Square toric surface codes: Pauli errors

We now study the errors affecting the surface code. Since, as we said, stabilizer formalism in mainly meant to tackle Pauli errors, from now on, whenever we talk of error we imply Pauli error chain. We will see in next section how non Pauli errors are treated.

Homology classes for open error chains

Open error chains can be assigned a homology class by extending the notion of homology beyond cycles. As in the cyclic case, 1-chains c_a, c_b are homologous if:

$$c_a = c_b + \partial f, \tag{2.15}$$

for some 2-chain f.

As for the cyclic case, homologically equivalent error chains have the same effect on codewords.

Syndrome classes

Let E be a detectable error. At least one of its a X- and Z-type parts must be a detectable error. Let $\bar{c}_x$ and c_z be their support. A detectable error support must be an open chain, otherwise is in the stabilizer or normalizer. We thus ask which syndromes are triggered by an open chain:

- For an X-error chain $\bar{c}_x$, the stabilizer generator A_m associated with a primal face f_m satisfies

  $$A_m X(\bar{c}_x) = (-1)^{|\partial f_m \cap \bar{c}_x|} X(\bar{c}_x) A_m,$$

  where $|\partial f_m \cap \bar{c}_x|$ counts the number of dual edges of $\bar{c}_x$ crossing the boundary of the face f_m (in practice overlapping with on the sides of the plaquette). This means that the syndrome associated with a plaquette is triggered if and only if the X-chain overlaps with an odd number of sides of the plaquette.

- For a Z-error chain c_z, the stabilizer generator B_k associated with a dual face $\bar{f}_k$ satisfies

  $$B_k Z(c_z) = (-1)^{|\bar{\partial} \bar{f}_k \cap c_z|} Z(c_z) B_k,$$

  where $|\bar{\partial} \bar{f}_k \cap c_z|$ counts the number of primal edges of c_z crossing the boundary of the dual face $\bar{f}_k$ (in practice entering the corresponding star operator). This means that the syndrome associated with a star is triggered if and only if the Z-chain overlaps with an odd number of rays of the star.

If we consider a simple Z-error chain in the primal lattice, the triggered syndromes correspond to the X-type stabilizers B_k associated with the endpoints of the chain.

If we consider a simple X-error chain in the dual lattice, the triggered syndromes correspond to the Z-type stabilizers A_m associated with the endpoints of the chain.

The syndromes of more complex chains, such as those with bifurcations, can be determined by superposition of the syndromes of their simpler components. Thus, only the endpoints of the error chain determine the measured syndrome, and two error chains with the same boundary produce the same syndrome. Two homogeneous error chains of the same type belong to the same syndrome class if and only if the difference of their supports is a cycle, as shown is 2.1. This implies they have the same endpoints.

Two general error chains belong to the same syndrome class if both of their parts are in the same syndrome class. In other words, the syndrome class is a coset of the cycle group inside the 1-chain group. All error configurations compatible with a syndrome can be generated by taking an open chain as representative and adding stabilizers and logical operators.

Error correction

A necessary condition for successful recovery of an individual detectable Pauli error chain E is that the recovery operation cancels the observed syndrome violations. Specifically, any $Z(c_r)$ and $X(\bar{c}_r)$ with supports satisfying

$$\partial c_r = \partial c_z, \quad \partial \bar{c}_r = \partial \bar{c}_x \tag{2.16}$$

fulfill this condition. This means that the boundary of the recovery chains must match the boundary corresponding to the measured syndromes.

The error and recovery chains together define a cycle:

$$\gamma = c + c_r, \quad \bar{\gamma} = \bar{c} + \bar{c}_r.$$

Successful error correction requires these resulting 1-cycles to be trivial. Let l and $\bar{l}$ be non-trivial 1-cycles. Suppose that after the recovery procedure, the resulting 1-chains satisfy

$$\gamma = l + \partial f, \quad \bar{\gamma} = \bar{l} + \partial \bar{f}$$

Then, the recovery operation implements, by accident, a logical operation on the code state. Explicitly, applying the recovery Z-chain to the code state $|\psi\rangle$ disturbed by E results in:

$$\begin{aligned} Z(c_r)Z(c_z)\,|\psi\rangle &= Z(\gamma)\,|\psi\rangle \\ &= Z(l+\partial f)\,|\psi\rangle \\ &= Z(l)Z(\partial f)\,|\psi\rangle \\ &= Z(l)\,|\psi\rangle \\ &= \overline{Z}^{(l)}\,|\psi\rangle\,, \end{aligned}$$

where $\overline{Z}^{(l)}$ denotes the logical Z-operator associated with the homology class of the non-trivial cycle l.

Similarly, a logical X-error can be implemented by an improper choice of the $X(\bar{c}_r)$ recovery operator resulting in a non-trivial cycle $\bar{l}$.

As usual, the decoding operation chooses the best fit for the recovery chains. Because the only available information are the syndromes, that is, the boundaries, these best fits can just be inferred probabilistically, and there is no way to avoid entirely the risk of logical operation. Usual considerations apply to generalizing to noise channels supported on more Kraus elements,recovery channels supported on more Kraus elements, mixed initial states.

2.3 Physical surface code

We saw in the previous section the abstract mathematical structure of the surface code. In this section, we study its physical implementations.

2.3.1 Measurement cycle and quiescent states

In experimental settings, the surface code is implemented on lattices of physical qubits. The logical state of the code is encoded in the wavefunction of the lattice. The stabilizers are periodically measured, collapsing the general wavefunction of the lattice to a joint eigenvector of the stabilizers, called a *quiescent state* in the surface code terminology, corresponding to what we referred to as a syndrome subspace.

This cyclic measurement collapses coherence between components of the wavefunction belonging to different quiescent states, but it does not collapse coherent superpositions of Pauli errors within the same quiescent state.

For example, let $|\psi_L\rangle$ be a logical code state, Z_1 a Pauli-Z chain, and $\hat{Z}_1$ a chain with the same endpoints but opposite homology class, both belonging to the same syndrome class.

After the collapse of the wavefunction, the surface code can be in a state

$$|\psi_{\text{collapsed}}\rangle = \sqrt{\frac{3}{4}}\, Z_1 |\psi_L\rangle + \frac{1}{2}\, \hat{Z}_1 |\psi_L\rangle .$$

There is no recovery Pauli chain that can fully correct this state. One can reduce the difference with the original logical state by choosing the correct homology class of Z_1, but the system will remain in coherent superposition with an uncorrected component.

Thus, there is no way to prevent the accumulation of uncorrected coherent errors. This problem is well-known in the literature; for example, it was studied in [6], where it was found empirically that the surface code generally performs well enough against coherent errors.

From this point onward, we restrict ourselves to errors that have no coherence between different quiescent states, leaving the treatment of coherent errors for future work. In this case, one can always fully correct the state by "guessing" the right homology class.

2.3.2 Surface code circuit implementation

The standard approach follows the methodology outlined in [7].

Lattice structure

The lattice consists of alternating data qubits (open circles in Fig. 2.4) and ancilla qubits (filled circles). Each ancilla qubit is associated with either an X-type or a Z-type stabilizer.

Circuit for the measurement cycle

Each cycle begins with the initialization of every ancilla qubit (used for stabilizer measurement) in the ground state $|0\rangle$. The ancillas then interact with the data qubits through entangling gates, depending on whether they are associated with an X-type or Z-type stabilizer, as shown in Figures 2.6 and 2.5. These circuits implement controlled-X operations with the ancilla as the target qubit.

For an X-type stabilizer, the control is given by the product $X_a X_b X_c X_d$ acting on the four surrounding data qubits, while for a Z-type stabilizer, the control is given by the product $Z_a Z_b Z_c Z_d$. Applying these controlled operations and subsequently measuring the Z-operator on ancillas is logically equivalent to measuring the corresponding stabilizer operators and yelds the same eigenvalues.

Because the stabilizers commute, there is no conflict between measurement of different stabilizers. After a round is finished the lattice wavefunction is collapsed on a quiescent subspace. Because of errors and error correction, it may move to a different quiescent state in the following rounds. We now dive into errors typically affecting a surface code.

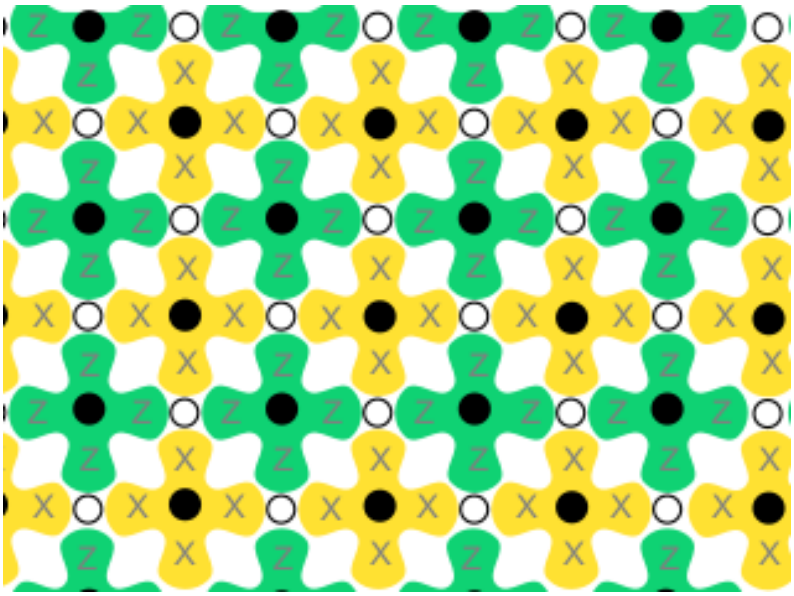


Figure 2.4. Structure of a surface code physical lattice implementation. Image adapted from [7]

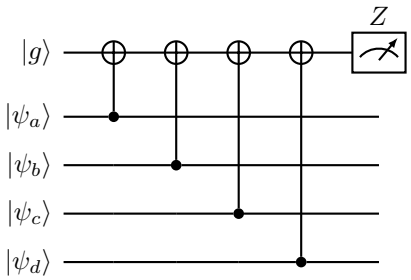


Figure 2.5. Z-type stabilizer measurement. Data qubits act as controls in CNOT gates.

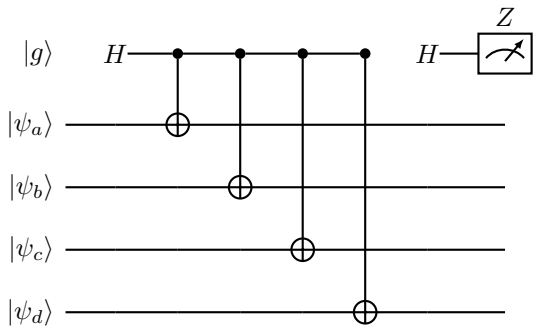


Figure 2.6. X-type stabilizer measurement. Hadamard gates rotate the ancilla to and from the X-basis.

2.3.3 Common errors affecting the surface code

The description of the errors and their propagation are based on [7] and [8] and [3] as a default source, except when explicitly stated.

The surface code is affected by a wide variety of noise processes. Although such noise acts continuously in time - evolving the system's density matrix ρ according to the von Neumann equation - we are typically interested only in the state at discrete time steps. Therefore, the effect of noise over a fixed time interval can be modeled using quantum channels. To simplify analytical treatment and enable classical simulability, these channels are often Pauli-twirled or otherwise approximated into stochastic Pauli channels.

Environmental noise

Interaction with background environmental noise is a primary source of errors in quantum systems. Typically, this noise is coherent, non-Clifford, and continuous in time. For instance, a single qubit interacting with its environment experiences both phase damping and amplitude damping, which depend on the state and on specific features of the system-environment coupling.

However, it has been shown in [9] that such general environmental noise can be *twirled* into an effective local depolarizing channel:

$$\mathcal{N}_{env}(\rho) = \bigotimes_i \mathrm{Dep}_i(\rho_i),$$

Leakage

Leakage refers to transitions outside the computational subspace, such as $|1\rangle \to |2\rangle$. These errors are not correctable by standard Pauli-based decoders and typically require special mitigation strategies. Leakage is outside the scope of this thesis.

Initialization errors

Ancilla qubits must be initialized in the $|0\rangle$ right after syndrome extraction. However, in practice, imperfect control during state preparation leads to a non-zero probability that the ancilla begins in the wrong state, corrupting the outcome of syndrome measurements.

Rather than being initialized in the pure state $|0\rangle\langle 0|$, ancillas are effectively prepared in a mixed state of the form:

$$\rho_{\text{init}} = (1-p)\,|0\rangle\langle 0| + p\,|1\rangle\langle 1|\,,$$

where p is the probability of initialization error. This noise can be interpreted as a stochastic bit-flip (Pauli-X) error occurring immediately after perfect initialization.

Such errors are particularly important because they directly affect the reliability of syndrome measurements and may lead to incorrect decoding decisions if not properly accounted for.

Ancilla readout errors

Measurement of ancilla qubits is also imperfect. A common model assumes a fixed probability of readout flip: a measurement result of 0 is reported as 1 (or vice versa) with some small probability.

Crosstalk

We consider crosstalk errors arising from residual Ising-type interactions between neighboring qubits, modeled by the following Hamiltonian suggested in [10]:

$$H_{\text{crosstalk}} = J \sum_{\langle i,j \rangle} Z_i Z_j,$$

where the sum runs over all neighboring qubit pairs $\langle i, j \rangle$, and J is the uniform coupling strength between qubits.

Over a fixed time step Δt the system evolves under the action of this hamiltonian by:

$$U = e^{-iH_{\text{crosstalk}}\Delta t/\hbar} = e^{-i\theta \sum_{\langle i,j \rangle} Z_i Z_j},$$

where

$$\theta = \frac{J\Delta t}{\hbar}$$

and so for a density matrix ρ,

$$\rho \mapsto U\rho U^\dagger.$$

Because this evolution is coherent simulating its effect exactly on a classical computer requires exponential resources, limiting efficient classical analysis.

To our knowledge, a twirling procedure that could simplify this coherent crosstalk noise into a classical stochastic error model has not been established in the literature, so we leave this for future work.

Gate imperfections

In realistic quantum hardware, quantum gates do not implement ideal unitaries exactly. Instead, each physical gate application corresponds to a noisy quantum channel ε, which deviates from the intended unitary evolution U. We write this decomposition as:

$$\varepsilon = \mathcal{N} \circ \mathcal{U}, \quad \text{where } \mathcal{N} := \varepsilon \circ \mathcal{U}^{-1}.$$

This representation expresses the noisy implementation as a perfect unitary gate followed by a noise-only channel $\mathcal{N}$ which captures all imperfections. This noise channel can in principle be coherent, but for simulability's sake we consider just the stochastic terms in the chi-matrix expansion. We assume the noise is isotropic, so the probability of a stochastic error must depend just on its weight. In practice in the literature the standard choice is the global depolarizing channel:

- CNOT: $\mathcal{N}(\rho) = (1-p)\rho + \frac{p}{16}\sum_{P\in\mathcal{P}_2} P\rho P^\dagger$
- Hadamard: $\mathcal{N}(\rho) = (1-p)\rho + \frac{p}{4}\sum_{P\in\mathcal{P}_1} P\rho P^\dagger$
- Pauli (used for recovery): assumed noiseless

2.3.4 Propagation of Errors

This subsection is based on [8].

The quantum gates used in the measurement circuits can propagate single-qubit errors into different single-qubit or multi-qubit correlated errors. Propagation of an error E through gates is studied by finding a solution E' for the the equation:

$\text{GATE} \cdot E = E' \cdot \text{GATE}$

that means finding a way describe the effect on the output of the error applied on the input.

CNOT error propagation: Let qubit c be the control and t be the target.

$$\begin{aligned}\text{CNOT} \cdot X_c &= X_c X_t \cdot \text{CNOT} \\ \text{CNOT} \cdot Z_c &= Z_c \cdot \text{CNOT} \\ \text{CNOT} \cdot Y_c &= Y_c X_t \cdot \text{CNOT}\end{aligned}$$

$$\begin{aligned}\text{CNOT} \cdot X_t &= X_t \cdot \text{CNOT} \\ \text{CNOT} \cdot Z_t &= Z_c Z_t \cdot \text{CNOT} \\ \text{CNOT} \cdot Y_t &= Z_c Y_t \cdot \text{CNOT}\end{aligned}$$

In words,this means that the X errors can spread from the control to the target, while the Z errors can spread from the target to the control.

Error propagation through Hadamard gates

Hadamard gates interchange X and Z errors and flip the sign of Y:

$$\begin{aligned}H \cdot X &= Z \cdot H \\ H \cdot Z &= X \cdot H \\ H \cdot Y &= -Y \cdot H\end{aligned}$$

In words Hadamard gates turn X errors in Z and vice versa.
The CNOTs in the stabilizer circuits propagate the errors as follows:

Error propagation in Z-type stabilizer circuits

Errors on data qubits:

- X on data $\rightarrow$ propagates as X on data and X on ancilla if the corresponding CNOT is yet to be performed.
- Y on data $\rightarrow$ propagates as Y on data and X on ancilla if the corresponding CNOT is yet to be performed.
- Z on data $\rightarrow$ remains localized on data only.

Errors on ancilla qubits:

- X on ancilla $\rightarrow$ remains localized on ancilla only.
- Y on ancilla $\rightarrow$ propagates as Y on ancilla and Z on data qubits connected by CNOT gates that are yet to be performed.
- Z on ancilla $\rightarrow$ propagates as Z on ancilla and Z on data qubits connected by CNOT gates that are yet to be performed.

Error propagation in X-type stabilizer circuits

Errors on data qubits:

- Z on data $\rightarrow$ propagates as Z on data and Z on ancilla if the corresponding CNOT is yet to be performed.

- Y on data $\rightarrow$ propagates as Y on data and Z on ancilla if the corresponding CNOT is yet to be performed.
- X on data $\rightarrow$ remains localized on data only.

Errors on ancilla qubits:

- Z on ancilla $\rightarrow$ remains localized on ancilla only.
- Y on ancilla $\rightarrow$ propagates as Y on ancilla and X on data qubits connected by CNOT gates that are yet to be performed.
- X on ancilla $\rightarrow$ propagates as X on ancilla and X on data qubits connected by CNOT gates that are yet to be performed.

CNOT Scheduling

The scheduling of CNOT gates during syndrome extraction critically determines the spreading of errors. Since each data qubit is entangled via CNOT gates to its four neighboring measurement qubits, it is not possible to perform all CNOT gates simultaneously without causing gate conflicts.

A common and effective scheduling strategy divides the syndrome extraction cycle into four sequential steps, each corresponding to one of the four cardinal directions: north, east, south, and west. In each step, all CNOT gates between ancilla qubits and their neighbors in the given direction are applied in parallel. This ensures that each qubit participates in at most one gate at a time. Different scheduling strategies can have distinct effects on error propagation and spreading and must be accounted for during decoding.

2.3.5 Time dependence

Each of the noise sources considered so far can exhibit some form of time evolution. This is particularly true for background noise, which may vary due to environmental fluctuations or hardware drift. Different noise patterns can operate at different time scales. Examples of noise evolution patterns are slow drifts over minutes or hours induced by temperature variation. Realistic quantum hardware is typically recalibrated periodically to mitigate the effect of such drifts and udpdate the known noise parameters.

3 Markov decision processes and reinforcement learning

The goal of this chapter is to introduce key concepts from control theory and machine learning that will be used in the subsequent sections. We recall only the elements necessary to define the notation and set the stage for the discussion. Unless otherwise stated, this chapter follows the presentation in [11], to which the reader is referred for a more in-depth treatment.

Decoding can be framed as a problem of sequential decision-making under uncertainty. We thus dive into the mathematical description of this subject.

3.1 Markov models

We assume the reader has knowledge of standard Markov models. A simple generalization is that of *hidden Markov models* (HMM), introduced in [12]. HMM are Markov models where the true states are not directly observable. The states emit *observations* that provide indirect information about the hidden states.

Formally, a HMM is defined by the tuple:

$$(S, O, T, K_O),$$

where:

- S is a set of hidden states,
- O is a set of observations,
- $T(s' \mid s)$,the transition kernel, is the probability of transitioning from state s to state s',
- $K_O(o \mid s)$, the emission kernel, is the probability of emitting observation o from state s

The system dynamics at each time step are: transition from state s_t to s_{t+1} according to T, then emission of an observation o_{t+1} according to K_O.

Since the states are hidden, the available information is encoded into a *belief*, a probability distribution over the states. The belief is updated after each transition and observation.

After a transition, the belief updates using the transition operator:

$$b'(s') = \sum_s K_T(s' \mid s) b(s).$$

After receiving an observation, the belief updates via Bayes' rule using the emission kernel:

$$b(s \mid o) = \frac{K_O(o \mid s)\, b(s)}{\sum_{s'} K_O(o \mid s')\, b(s')}, \tag{3.1}$$

Combining the two gives the whole transition.

3.2 Markov decision process

The main sequential decision-making framework we consider is the *Markov decision process* (MDP). It models the interaction between an *agent* and an *environment* as a discrete-time stochastic process. At each time step, the agent observes the current *state* of the environment, selects an *action*, receives a *reward*, and the environment transitions to a new state. These dynamics are governed by the Markov property: the next state and reward depend only on the current state and action.

We focus on infinite-horizon discounted MDPs. While several slightly different formalizations exist in the literature, we adopt a general one. An infinite-horizon discounted MDP is defined by the tuple

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, K_E, \gamma),$$

where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, and $\mathcal{R} \subseteq \mathbb{R}$ the reward space. The *environment kernel* $K_E(s', r \mid s, a)$ specifies the joint distribution over next state and reward given the current state-action pair.

The scalar $\gamma \in [0, 1]$ is the discount factor, which determines the weight of future rewards in the return. The process begins from an initial state $S_0 \in \mathcal{S}$. We allow for the existence of *terminal states* $s_{\text{term}} \in \mathcal{S}$, such that once entered, the system remains in s_{term} forever with zero reward.

At each time step t, the agent observes the current state S_t, selects an action $A_t \in \mathcal{A}$, and the environment samples a reward $R_{t+1} \in \mathcal{R}$ and a next state $S_{t+1} \in \mathcal{S}$ according to the environment kernel:

$$(S_{t+1}, R_{t+1}) \sim K_E(\cdot, \cdot \mid S_t, A_t).$$

A *policy* π defines the agent's behavior. Since the environment is Markovian, it suffices for the policy to depend only on the current state: $\pi(a \mid s)$ gives the probability of taking action a in state s. Policies may be stochastic, in which case $\pi(a \mid s)$ is a distribution, or they may be deterministic, in which case $\pi(s)$ is a function.

The combination of a policy with the environment defines a Markov chain over the process $\{(A_t, R_{t+1}, S_{t+1})\}_{t \geq 0}$ whose transition kernel can be written as

$$\Pr\big(A_t, R_{t+1}, S_{t+1} \mid A_{t-1}, R_t, S_t\big) = \pi(A_t \mid S_t)\; k_E(R_{t+1}, S_{t+1} \mid A_t, S_t)\,.$$

A sequence sampled from this chain is called a *rollout*. A rollout starting from an initial state and ending in a terminal state is called an *episode*.

An MDP is called *episodic* if it contains one or more terminal states that are guaranteed to be reached eventually under any policy.

The series on the reward encountered in rollout is called the *discounted return*.

$$\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_{t+1},$$

where $\gamma \in [0,1)$ ensures convergence of the infinite sum.

The objective is to choose a policy that maximizes the expected discounted return.

To compare policies, we first define the *value function*:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s, \pi\right],$$

which evaluates the expected return when starting from state s and then following policy π.

We also define the *action-value function*, or Q-function:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s, A_0 = a, \pi\right],$$

which evaluates the expected return of taking action a in state s and then following policy π.

The optimal policy π^* is the one that maximizes the expected discounted return. Since we are assuming finite action and state spaces, the policy space is compact (specifically, a product of simplices), and the return function is continuous in π. By the Weierstrass theorem, an optimal policy exists.

The environment kernel is also known as "world model".

3.3 Solving an MDP

Solving a MDP means finding an optimal policy. If the world model is known then in principle we can compute the optimal policy by solving the so called Bellman equations. If instead the world model is unknown, the agent must learn an approximately optimal policy through interaction. The agent collects samples of states, actions, and rewards, and uses them to improve its behavior over time. This is the *reinforcement learning* (RL) case.

Model-based reinforcement learning lies in between these two extremes. It combines data-driven learning with elements of planning. For this thesis we are interest in an approach called *RL on imagination*. In RL on imagination an agent first learns a model of the environment and then uses it to simulate synthetic rollouts. These imagined rollouts can be treated as if they were sampled by the environment and used to train a policy using standard reinforcement learning techniques. This is especially valuable when the model is good enough to simulate the dynamics, but exact planning is too computationally intensive. Two central classes of RL methods are *Q-learning* methods and *policy-based* methods. We need to introduce the first to analyze the decoding literature and the second to present our decoding framework.

3.4 Q-Learning

Q-learning methods are a classic family of algorithms that estimates the optimal action-value function $Q^*(s, a)$ by iteratively applying the *Bellman optimality update*:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right],$$

where α is the learning rate and (s, a, r, s') are sampled from rollouts.

Once the Q-function is learned, a so called *greedy* policy can be extracted as

$$\pi(s) = \arg\max_a Q(s,a).$$

The rollouts are generated by *exploration*. In principle they can be generated by an agent following any policy. This makes Q-learning methods *off-policy* methods. On-policy methods instead need to use the the policy under optimization, the *target* policy, to generate training rollouts. A policy used with sole purpose of exploring the data space is called a *behavioral* policy.

The typical exploration strategy in Q-learning is ϵ-greedy, where with probability ϵ we explore by a behavioral policy, otherwise we explore by the greedy policy associated to the current Q function.

The policy returned by Q-learning is deterministic. In practice, when the state space is too large to be thoroughly explored, Q-learning can be extended with function approximation, typically by parameterizing $Q(s, a)$ with a neural network. This allows generalization across states, but the action space must still be small enough to allow explicit maximization over a, as required by the update rule.

When the action space is exponentially large, the maximization step becomes intractable. In such cases, Q-learning is no longer practical, and we must adopt methods that directly parameterize the policy itself.

3.5 Policy gradient methods

Policy gradient methods are on-policy approaches that optimize a parametric policy by performing gradient ascent on the expected return. These methods rely on the *policy gradient theorem*, which provides an expression for the gradient of the objective function with respect to the policy parameters.

3.5.1 Policy gradient theorem

Consider a parametric policy $\pi_\theta(a \mid s)$.

The goal of policy optimization is to maximize the expected return under the given policy:

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]. \tag{3.2}$$

To achieve the maximization, we need to compute the gradient and then plug it into some form of gradient ascent.

To express the policy gradient compactly, we define the *discounted state visitation distribution* as:

$$d_\pi(s) \coloneqq \sum_{t=0}^{\infty} \gamma^t \sum_{s_0} p_0(s_0)\, p^{\pi_\theta}(s_0 \to s, t), \tag{3.3}$$

where we dropped the dependency of d on γ and θ in the notation for clarity. $p_0(s_0)$ represents an initial distribution over states and $p^{\pi_\theta}(s_0 \to s, t)$ represents the probability of transition in t steps.

This quantity measures how often, in expectation, the policy visits each state over time, with future visits discounted by γ^t (notice that is not a normalized distribution).

The *policy gradient theorem* shows that the gradient of the objective can be written as:

$$\nabla_\theta J(\pi_\theta) = \sum_s d_\pi(s) \sum_a \pi_\theta(a \mid s)\, Q^{\pi_\theta}(s, a)\, \nabla_\theta \log \pi_\theta(a \mid s). \tag{3.4}$$

This equation is the foundational result upon which policy gradient methods are built.

3.5.2 REINFORCE

The REINFORCE algorithm estimates the policy gradient using a Monte Carlo approximation. We assume the reader knows what a Monte Carlo approximation is. We limit to the case of episodic MDPs. Given an episode

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T),$$

of length T, sampled from the policy π_θ, the gradient of the expected return can be approximated as:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t \mid s_t), \tag{3.5}$$

where G_t is the return after time step t, defined as:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$$

The trajectory length T is the time in which the environment enters a terminal state.

To update the parameters, REINFORCE applies vanilla gradient ascent after every sampled trajectory:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \gamma^t G_t, \tag{3.6}$$

where α is the learning rate. Reducing the variance of the gradient estimate is crucial for improving the convergence speed of the algorithm.

The REINFORCE estimator suffers from high variance and thus the convergence can be extremely slow.

3.5.3 Baseline variance reduction

The variance of the policy gradient estimator can be reduced by subtracting a *baseline* from the return. Consider the generalized baseline-augmented policy gradient:

$$\nabla_\theta J(\pi_\theta) = \sum_s d_\pi(s) \sum_a \pi_\theta(a \mid s) \left(Q^\pi(s,a) - b(s,a)\right) \nabla_\theta \log \pi_\theta(a \mid s). \tag{3.7}$$

This expression is unbiased if the baseline satisfies:

$$\sum_a \pi_\theta(a \mid s) b(s,a) \nabla_\theta \log \pi_\theta(a \mid s) = 0 \qquad \text{for all } s. \tag{3.8}$$

A sufficient condition is to choose a baseline that is independent of the action, i.e., $b(s,a) = b(s)$. In that case:

$$\sum_a \pi_\theta(a \mid s) b(s) \nabla_\theta \log \pi_\theta(a \mid s) = b(s) \nabla_\theta \sum_a \pi_\theta(a \mid s) = b(s) \nabla_\theta 1 = 0,$$

which ensures unbiasedness.

With a state-dependent baseline, the Monte Carlo gradient estimate becomes:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(G_t - b(s_t)\right). \tag{3.9}$$

Although the baseline does not affect the expectation of the gradient, it can significantly reduce variance and improve convergence.

3.5.4 Starting policy

Training a policy from random initialization often results in poor performance, especially in environments with sparse rewards or long horizons. A standard strategy is to initialize the policy using *imitation learning* from expert demonstrations.

Let $\pi_\theta(a \mid s)$ be the trainable policy and $\pi_E(a \mid s)$ the expert policy. The imitation objective is to minimize expected cross-entropy between the expert and the trainable policy over the discounted state visitation distribution induced by the expert:

$$\mathcal{L}_{\text{imitation}}(\theta) = -\mathbb{E}_{s \sim d_E(s)} \sum_a \pi_E(a \mid s) \log \pi_\theta(a \mid s)$$

This can be rewritten as an expectation over the discounted state-action distribution $d_E(s,a) = d_E(s)\pi_E(a \mid s)$ of the following quantity:

$$\mathcal{L}_{\text{imitation}}(\theta) = -\sum_{s,a} d_E(s,a) \log \pi_\theta(a \mid s)$$

Empirical approximation

In practice, the expert policy π_E may not be known explicitly, but we can observe expert trajectories. Assume we are given a dataset of M trajectories from π_E, with trajectory i given by $\{(s_t^i, a_t^i)\}_{t=0}^{T_i}$.

We approximate the discounted state-action distribution with a weighted empirical distribution:

$$\hat{d}_E(s,a) = \frac{1}{Z} \sum_{i=1}^{M} \sum_{t=0}^{T_i} \gamma^t \delta(s = s_t^i, a = a_t^i) \quad \text{where} \quad Z = \sum_{i=1}^{M} \sum_{t=0}^{T_i} \gamma^t$$

This leads to a Monte Carlo approximation of the imitation loss:

$$\mathcal{L}_{\text{imitation}}(\theta) = -\sum_{s,a} \hat{d}_E(s,a) \log \pi_\theta(a \mid s) = -\frac{1}{Z} \sum_{i=1}^{M} \sum_{t=0}^{T_i} \gamma^t \log \pi_\theta(a_t^i \mid s_t^i)$$

By minimizing this loss we ensure that the policy starts training in a region of the parameter space that reflects competent behavior, allowing subsequent reinforcement learning to refine and improve performance more effectively. In this way we can also make sure that the RL trained policy surpasses in performance the expert policy.

We now can generalize to a more general version of the MDPs, supported on HMMs rather than standard Markov chains.

3.6 Partially observable Markov decision processes

In partially observable settings the agent cannot directly observe the state of the environment, nor the reward. We can formalize markovian decision process in such settings as *partially observable Markov decision processes* (POMDP). Before giving the definition to simplify the notation we first redefine the Markov decision process using an explicit transition kernel and a reward function, instead of a joint environment kernel. The joint formulation introduced earlier aligns with examples from the quantum decoding literature in the next chapter, where rewards and transitions are sampled together. However, for generalization to partially observable settings, it is more practical to decouple them.

3.6.1 Redefining MDPs

An infinite-horizon MDP is redefined by the tuple:

$$(\mathcal{S}, \mathcal{A}, R, K_T, \gamma),$$

where:

- $\mathcal{S}$ is a discrete set of states. In previous definition continuous state space were allowed, but here we limit to discrete.
- $\mathcal{A}$ is a set of actions.
- $R(s)$ is the immediate reward function assigning a scalar reward to each state $s \in \mathcal{S}$. Here the reward is a function of the arrival state after a transition.
- $K_T(s' \mid s, a)$ is the *controlled transition kernel* giving the probability of reaching state s' after taking action a in state s.
- $\gamma \in [0,1)$ is the discount factor.

At each time step t, the agent is in state s_t, selects an action a_t, and then:

$$s_{t+1} \sim K_T(s_{t+1} \mid s_t, a_t),$$
$$r_{t+1} = R(s_{t+1}).$$

3.6.2 Definition of POMDP

An infinite-horizon POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{O}, K_T, K_O, R, \gamma),$$

where the notation matches what we defined previously for MDPs and HMMs.

Since the agent cannot observe the environment state directly, it maintains a belief b over hidden states. Moreover, because the true reward $R(s)$ is also not directly accessible, we define the expected reward function over beliefs as:

$$r(b) = \sum_s b(s)\, R(s).$$

At each time step, the agent selects an action a_t based on its belief b_t. Here the policies are a function of the belief, not the state. After the action the agent updates its belief by the transition kernel $T(s' \mid s) = K_T(s' \mid s, a_t)$. It then receives an observation o_{t+1}, and updates it belief by Bayes rule like in 3.1. It thus gets the updated belied b_{t+1} and can use this belief to compute its expected reward $r_{t+1} = r(b_{t+1})$. Then the cycle repeats.

3.7 Solving a POMDP

In this formulation the world model consists of both the reward function and the controlled transition kernel. When the world model is known, it is in principle possible to solve a POMDP exactly. However, exact solutions are computationally intractable for all but the smallest problems, so here too we use RL as both a means of dealing with unknown world models or approximation.

3.7.1 Three markovian levels

Solving a POMDP usually requires defining a *belief MPD*, that is another MDP whose states are the belief and whose features are derived from the POMDP. Sometimes it's possible the belief MPD makes some approximation relative to the full POMPD. We have thus to manage three levels of markvian dynamics:

- the supporting HMM on which the POMPD is defined
- the POMDP defined on the HMM (actually, a family of HMM, each with transition kernel determined by the action)
- the belief MPD, supported on the POMPD, that is a continuous state space MPD witH certain transitions.

3.7.2 REINFORCE for POMDPs

Policy gradient methods can be extended to POMDPs by parameterizing a policy $\pi_\theta(a \mid b)$ that maps belief states to action distributions. Since these methods support continuous state spaces, it is natural to treat the belief b_t as the state of a belief MDP following the first definition. We

can thus solve POMDPs using technique for solving MDPs with continuous states. We are in particular interested in using REINFORCE. In this case, since training proceeds from the agent's point of view, and rollouts take the form:

$$(b_t, a_t, r_{t+1}),$$

where $r_{t+1} = \rho(b_{t+1})$ is the expected reward. Other than this the policy gradient framework remains unchanged.

3.7.3 POMDPs and Termination Conditions

Policy gradient methods require episodic tasks with clear termination conditions to properly define returns and update policies. In POMDPs, termination conditions are formally inherited from the underlying MDP: if the environment contains one or more terminal states $\mathcal{S}_{\text{term}} \subseteq \mathcal{S}$, then the belief state b is terminal if and only if the belief is fully concentrated on these terminal states, i.e.,

$$\sum_{s \in \mathcal{S}_{\text{term}}} b(s) = 1.$$

This means the system is known with certainty to be in a terminal state.

However, in many practical problems - especially those with continuous, very large, or combinatorial state spaces - this exact condition is rarely met. The belief may never collapse fully onto a terminal state. For this reason, it is common to define *terminal beliefs* directly for the belief MPD based on heuristic criteria that indicate sufficient certainty or completion of the task.

Formally, we define a subset $\mathcal{B}_{\text{term}}$ of terminal beliefs such that when the belief state $b_t \in \mathcal{B}_{\text{term}}$, the episode is artificially terminated from the perspective of training. This allows the use of policy gradient and other episodic reinforcement learning methods in POMDPs even when the underlying environment is continuing indefinitely.

4 Decoding the surface code

A vast literature has been developed on decoding techniques for the surface code. In this chapter, we present a brief overview of what is of interest for our work.

4.1 Decoding vs. Error Correction

The term "decoding" is somewhat ambiguous. Literally, decoding refers to inferring from the syndrome the error affecting the code. In practice, however, it is often used to refer to the error correction procedure itself.

These two notions coincide in one-shot decoding, where the recovery chain matches the most probable error or error class. When we introduce reinforcement learning decoders, the recovery chain may not be intended for immediate correction, so decoding and error correction can become distinct concepts.

To stay consistent with the literature, we continue to use "decoding" to mean "error correction" throughout the rest of this thesis.

4.2 Posterior decoding strategies

This section is based on [13] and [5]. Maximum posterior decoders are mathematically principled methods that choose corrections by estimating probabilities of different error configurations. They are considered the classical approach and serve as a benchmark for other decoding methods.

The posterior probability of an error E given an observed syndrome $\hat{s}$ is given by Bayes' theorem:

$$p(E \mid \hat{s}) = \frac{p(\hat{s} \mid E)p(E)}{p(\hat{s})}$$

The prior $p(E)$ is given by the noise model. The likelihood $p(\hat{s} \mid E)$ equals $\delta(\hat{s}, s(E))$ if syndrome measurements are perfect; if they are faulty, it is determined by a more general function derived from the measurement error model.

The *maximum posterior decoding* approach identifies the error configuration with the highest posterior probability given the observed syndrome $\hat{s}$:

$$\hat{E} = \arg\max_{E \in \mathcal{P}_n} p(E \mid \hat{s}).$$

and applies a recovery chain that corrects for it.

This decoding problem is NP-hard in the number of qubits under general error models. However, under the assumption of i.i.d. noise and perfect measurements, quantum maximum posterior decoding reduces to a *minimum-weight perfect matching* problem. In this formulation the most likely Pauli operator consistent with the measured syndrome corresponds to the lowest-weight error chain connecting syndrome defects.

In general, minimum-weight perfect matching is a classical combinatorial optimization problem defined on a graph. The goal is to pair up all nodes of the graph into disjoint edges (called a *perfect matching*) such that the sum of the weights of the chosen edges is minimized. This problem is nontrivial, as the number of possible matchings grows exponentially with the number of nodes.

In the context of decoding, we typically consider a fully connected graph where each node corresponds to a triggered stabilizer and the weight of an edge is given by the minimal path length between the two nodes on the abstract surface code lattice.

Despite this combinatorial complexity, the minimum-weight perfect matching problem can be solved in polynomial time thanks to the *Blossom algorithm*, introduced in [14]. In the presence of measurement errors, maximum posterior decoding must consider the distribution over possible true syndromes, typically by processing multiple rounds of measurement. The complexity increases but remains polynomial.

Although maximum posterior decoding can be implemented efficiently, it is not optimal in terms of logical accuracy. Since the effect of a correction is the same for all errors within a given homology class, the correction with the maximum probability of success corresponds to the homology class with the highest total probability, not necessarily the one containing the single most likely error. Given a measured syndrome $\hat{s}$, let $E(\hat{s})$ be a representative error satisfying

$$\sigma(E(\hat{s})) = \hat{s}.$$

Any other error E' consistent with the same syndrome can be expressed as

$$E' = E(\hat{s})\, S\, L,$$

where $S \in \mathcal{S}$ is a stabilizer and $L \in \mathcal{L}$ is a logical operator.

Each logical class corresponds to a coset $E(\hat{s})\,\mathcal{S}\,L$, and the posterior probability of the logical class L given the syndrome is

$$p(L \mid \hat{s}) = \sum_{S \in \mathcal{S}} p(E(\hat{s})\, S\, L \mid \hat{s}).$$

Degenerate maximum a posteriori decoding selects the logical class with the highest posterior probability:

$$\tilde{L} = \arg\max_{L \in \mathcal{L}} p(L \mid \hat{s}),$$

where $\tilde{L}$ denotes the MAP estimate of the logical operator class given the syndrome.

Computing $p(L \mid \hat{s})$ exactly is computationally intractable for general noise models, since it requires summing over an exponentially large number of stabilizers. In fact, the problem is #P-complete in the number of qubits.

An insightful approach to this challenge was presented in [5], where decoding is mapped to computing the partition function of a spin-glass model in the i.i.d. noise case. For the planar surface code, this can be solved in polynomial time using the Pfaffian algorithm. In the case of non-planar graphs, the problem becomes NP-hard.

4.3 Threshold theorem

Threshold theorems are foundational results in quantum information theory that guarantee the controllability of logical errors in fault-tolerant quantum computation.

Several versions of the threshold theorem exist, each applying to different codes, decoders, and noise models. For the surface code, a central result is due to [15], who proved that under the assumption of independent and identically distributed (i.i.d.) depolarizing noise with physical

error rate p, and using minimum-weight perfect matching for decoding, the logical error rate p_L obeys a bound of the form

$$p_L \leq C \left(\frac{p}{p_{\text{th}}} \right)^{\alpha d},$$

for sufficiently large code distance d, where $C > 0$ and $\alpha > 0$ are constants, and p_{th} is the *threshold* error rate.

This bound holds only when the physical error rate p is below the threshold p_{th}. In this regime, the logical error rate p_L can be exponentially suppressed by increasing the code distance d. Error correction is thus asymptotically effective with distance.

Conversely, when $p > p_{\text{th}}$, the bound no longer applies, and increasing the code distance does not necessarily reduce logical errors. Instead, p_L may plateau or even increase with d.

This exponential suppression of logical errors below threshold is not limited to i.i.d. noise and minimum-weight decoding: it also appears under a wide range of local noise models combined with sufficiently accurate decoding algorithms. Maximizing distance is an ubiquitous necessity in error-correction.

The performance of a decoder is governed by three key factors: accuracy, speed, and code distance. Speed is crucial because the decoder needs to keep with syndrome extraction rates. Since the number of error configurations grows exponentially with code distance, decoder speed generally decreases with increasing distance. Because we need to maximize both speed and distance a decoder's runtime must scale at most polynomially with distance to be considered viable. This is why minimum-weight perfect matching decoders are widely used despite their limited accuracy.

To complicate matters, to cope with faulty syndrome measurement decoders must integrate syndrome information across multiple rounds to reliably reconstruct the error history, further increasing the computational burden and requiring more sophisticated decoding strategies.

4.4 Quantum memory experiments

The performance of a decoding strategy for a given code and noise model is typically assessed through *quantum memory experiments*. These experiments evaluate the ability to store quantum information accurately over time. Logical qubits are first encoded using the code under investigation, and the system undergoes repeated rounds of syndrome measurement, decoding, and recovery.

At fixed time steps, the fidelity between the initially encoded logical state and the recovery state is recorded. The amount of time the fidelity remains above a certain threshold is called the *lifetime* of the logical qubit.

These experiments are most often conducted numerically, using either *simulated* or *emulated* data. Performing them on actual quantum hardware would require full quantum state tomography, which is too resource-intensive to apply at scale.

When running numerical memory experiments, it is important to distinguish between simulated and emulated data:

- *Simulated data* are generated using a classical stochastic model of the system and its noise. Simulation produces outcome statistics but not the full quantum state. Typically, Pauli errors are sampled from a noise model and propagated using the stabilizer formalism. This approach is efficient and scalable but limited to Clifford circuits and unable to capture non-Clifford noise or full quantum state properties.

- *Emulated data* are produced by numerically evolving the full quantum state under a noise channel. While this method is computationally more expensive, it allows for accurate modeling of realistic quantum dynamics, including non-Clifford effects.

Quantum memory experiments are often designed to measure two key properties: the *break-even point* and the *crossover point*.

4.4.1 Break-even experiment

A fundamental criterion for quantum error correction to be considered effective is that it improves the fidelity of the encoded logical qubit after noise and recovery, compared to the fidelity of an unencoded physical qubit subjected to noise alone. Formally, this condition reads

$$F\big(\rho, \mathcal{R} \circ \mathcal{E}(\rho)\big) > F\big(\rho_{\text{bare}}, \mathcal{E}(\rho_{\text{bare}})\big),$$

where ρ is the encoded logical state, ρ_{bare} is the corresponding unencoded physical qubit state. Notice that ρ is defined over the physical register Hilbert space, while ρ_{bare} is defined over another physical Hilbert space isomorphic to the logical Hilbert space. By encoded logical state we mean the physical realization of the code, while the physical bare qubit can be thought as the result of code mapping of logical to physical qubits one to one.

This fidelity criterion depends on time, as noise accumulates during storage and multiple rounds of error correction. To assess the long-term effectiveness of a error correcting scheme it is thus customary to compare the lifetimes of the bare and error-corrected qubits. This eliminates the time dependency. We can split the noise parameters space in region based on which qubit has longer lifetime. The break-even points are the transition point where the error corrected logical qubit's lifetime T_L surpasses that of the physical bare qubit T_P. Usually we have an i.i.d single qubit noise that is described just by one parameter so we have a single break-even point.

Achieving break-even demonstrates that quantum error correction yields a net benefit in preserving quantum information.

An example of a break-even study on hardware experimental data is [16].

4.4.2 Crossover experiment

To estimate the threshold error rate, decoding simulations or physical experiments are conducted across multiple code distances and physical error rates. Logical error rates are plotted as functions of the physical error rate for various code distances. We don't enter into the details of how these logical error rates are measured. These curves typically intersect at a single point, referred to as crossover point. Thus this point represents the physical error rate below which increasing the code distance starts to improve the logical error performance (technically, it is just the point where the curves differentiate, but if everything behaves as expected than below it the higher distance curves should start going lower in logical error rate faster). The crossover point thus provides an experimentally determined threshold. An example of such an experiment can be found in [17].

4.5 Separability

The surface code belongs to the family of Calderbank-Shor-Steane (CSS) codes (see [3]), which are a class of stabilizer codes that separately handle bit-flip and phase-flip errors.

One of the key features of CSS codes is their ability to simplify the decoding of Pauli errors. In principle, they allow separate decoding and recovery of X and Z errors.

Y errors can be decomposed as:

$$Y = iXZ,$$

so they can be detected and corrected by addressing the X and Z parts separately.

The optimality of this approach depends on the correlation between X and Z errors. If the two components are uncorrelated, separate decoding is indeed optimal. However, this is rarely the case in practice.

Depolarizing channels, for instance, introduce correlations. For example, conditioned on a Z error having occurred, the probability of an X error is $p(X \mid Z) = \frac{1}{2}$ while in general $p(X) = \frac{2p}{3}$. Not exploiting the information contained in such correlations can reduce decoding accuracy, so we will stick to joint decoding.

4.6 Neural network decoders

Traditional decoding strategies are not capable of operating in polynomial time under correlated error models. Such correlations commonly arise in realistic hardware due to crosstalk, faulty gates, and CNOT-induced error spreading. The computational complexity of posterior decoders has driven the development of new techniques aimed at reducing decoding time. Modern decoding approaches increasingly use neural networks. These decoders can be trained in various ways, often by supervised learning, where the network learns to classify syndromes by the homology classes they are most likely produced from. The main advantage of neural networks is runtime speed: once trained, decoding reduces to a simple feedforward pass, which is much faster than classical combinatorial search methods. Additionally, neural decoders can be trained directly on data without referring to a specific noise model. This property is known as *model agnosticism*. Neural networks can also cope with faulty syndromes by training on histories of syndrome data to learn how to recognize anomalous syndrome patterns.

4.7 Reinforcement learning decoders

Recently, reinforcement learning has emerged as a promising paradigm for training neural network-based quantum decoders. A milestone contribution was made in [2], where the problem of decoding faulty syndrome measurements in surface codes is reformulated as a reinforcement learning task.

4.7.1 Why reinforcement learning?

Traditional decoders adopt short-sighted strategies. They typically aim for immediate correction, maximizing fidelity after each correction step, assuming that any detected error must be fixed as soon as possible. However, this perspective overlooks the actual goal of quantum error correction: preserving logical information until it is ultimately needed. Immediate correction may sometimes be suboptimal when considering long-term fidelity, which is the only relevant metric for qubit lifeitimes. A more strategic decoder may not be the one that always tries to maximize fidelity after each shot but the one that, in the presence of uncertainty, performs exploratory actions rather than forcing a correction. This is why in this work, the decoding problem is framed as an episodic MDP with an unknown transition kernel, and solved using reinforcement learning on simulated data. The learned policy, called *decoding agent*, is then used as a decoder. The purpose is to achieve a decoder superior in lifetimes to traditional decoders.

4.7.2 Elements of the Markov decision process

Here we summarize of the MDP is built from the decoding problem, to reference in the following for further analysis. Consult the paper for more details.

Environment

The environment consists of a simulated quantum state subject to the stochastic noise generated by the *error model*, faulty syndrome measurements, and agent-applied corrections. Its underlying state, referred to as the *hidden state*, is defined by the total physical error affecting the data qubits.

This hidden state is not directly observable by the agent. Instead, the agent interacts with the environment through noisy syndrome measurements and responds by applying Pauli corrections over time.

Termination conditions

At each time step, the last syndrome is decoded in its perfect (noiseless) value, by a *referee decoder*, an auxiliary decoder used solely to assess whether the current correction sequence has successfully restored the logical state. This decoder has access to the full syndrome history and operates using a standard non RL decoder.

If the referee decoder fails to find a valid correction that returns the system to the code space, the quantum state is deemed unrecoverable, and the episode terminates.

States

The agent's state is a representation of the known information on the hidden state. It is a tuple of a fixed-length window of five past syndrome measurements, called *syndrome volume* and the *history* of Pauli corrections applied since the last syndrome round.

Actions

The action space comprises two types of choices. First, the agent can apply a local Pauli correction on a specific data qubit. These actions are represented as single-qubit operators X_i or Z_i, where the subscript i indicates the target qubit. The agent is restricted to selecting one such operator per time step.

Second, the agent may opt to perform an additional round of syndrome measurements instead of applying a correction. This "syndrome request" is encoded as a special action.

The total action space is thus the union of all single-qubit Pauli operators acting on the n data qubits, along with the syndrome measurement trigger, yielding a total of $2n + 1$ possible actions. At each step, the agent selects one action, which is executed immediately and either updates the system's error correction history or augments the available syndrome data.

Rewards

Rewards are assigned immediately after each action, based on the updated system state. At each time step, the agent receives a reward of 1 if the hidden state has returned to the original hidden state; otherwise, the reward is 0.

Transitions

There are two types of transitions:

- **Non-repeated Pauli flips**: the agent applies a Pauli operator that has not previously been applied at the same location in the current episode. In this case the history is updated while the syndrome volume is kept constant. The hidden state is kept constant.

- **Repeated Pauli flips or syndrome requests**: the agent either tries to reapply a previously used Pauli operator or asks for a new round of syndrome measurements. In either case the syndrome volume is updated with a new syndrome and the history is emptied, while the hidden state is perturbed by a new error from the error model.

4.7.3 Policy and training

The decoding agent selects actions using a convolutional neural network that represents the Q-function. The policy is defined by selecting actions greedily with respect to the Q-function,

This neural Q-function is trained using Q-learning. Training is performed entirely through interaction with a simulated environment. An ϵ-greedy strategy is employed during training, where with probability ϵ the agent takes a random Pauli flip near violated stabilizers or previously flipped qubits.

A key strength of this approach lies in its model-agnostic nature: the agent does not require explicit knowledge of the underlying error model.

4.7.4 Testing

The framework was tested by training the agent on a planar surface code subjected to i.i.d single-qubit depolarizing and bit-flip noise.

To evaluate the decoder's effectiveness, the authors performed a break-even test, although this test was referred to as a threshold evaluation in the original work. In this break-even test the logical qubit lifetime was measured as the average time before incorrect decoding by the referee decoder, rather that by a fidelity threshold. Physical qubit lifetime was simply the average time before the first error. Break-even was achieved for both bit-flip and depolarizing noise. At no point in the tested error rate range the RL decoder outperformed MWPM, not even increasing the rate of faulty syndromes.

4.7.5 Analysis by the authors

The work is intended as a paradigmatic proposal rather than a finished product. The authors identify some of its limitations. We cite just what is of interest for this thesis.

Inapplicability to the experimental setup

A fundamental limitation of the approach lies in its reliance on access to the underlying quantum state, both to compute rewards and to determine termination. This dependency is manageable in simulation, where the logical state is known by construction, but it is not available in experimental setups.

Inconsistency of model agnosticism

Since we are stuck with simulating the hidden state, we must postulate a specific error model, which defeats the goal of model agnosticism. The error model is also required by the referee decoder. Achieving model agnosticism would thus require to device a procedure that works both on experimental data and without a referee decoder.

Poor perfomance of Q-learning

The authors also suggest exploring more advanced reinforcement learning paradigms to improve decoder performance.

Part II

Original work

5 A New Approach to Decoding

We now analyze the reinforcement learning decoding strategy presented in the last chapter to identify its limitations, which will guide our search for a new framework.

5.1 Analysis of current reinforcement learning decoders

The decoder proposed in [2] is conceptually appealing, despite exhibiting limited performance. In this section, we extend the analysis provided by the authors, examining general strengths of reinforcement learning decoding that weren't considered and limitations of the specific case study in the paper.

5.1.1 Strengths

Reinforcement learning decoding offers several advantages that extend beyond those considered by the original authors of [2].

Modeling user access

The main idea behind the approach of [2] was to optimize long-term fidelity rather than immediate fidelity, with the goal of achieving longer qubit lifetimes. We can generalize this concept to a broader principle: *optimize fidelity when it matters*.

In many practical scenarios, maintaining perfect fidelity at all times is unnecessary, since an unmeasured qubit provides no utility to the user. Instead, the temporal pattern of qubit access is often determined by the specific use case. Consequently, optimal decoding should be designed *for the user's needs*, taking into account when and how the logical information is actually retrieved. This viewpoint encourages a user-centric formulation of decoding, where the optimization objective depends on the timing and nature of quantum data access.

Subjectivity of decoding performance

Fidelity alone cannot fully capture the quality of a decoding strategy. In realistic scenarios, the perceived performance of a decoder depends on subjective trade-offs. For instance, different applications may impose distinct thresholds for acceptable logical error rates, depending on their tolerance for the risk of logical failure.

Moreover, in some cases it may be preferable to declare data corrupted rather than attempt a risky recovery that could introduce undetected logical errors. We argue that these subjective components can be more effectively captured within a decision-theoretic framework. In particular, framing decoding as a Markov decision process allows the use of a subjective utility function to shape rewards and penalties according to the desired balance between data availability and logical reliability.

5.1.2 Limitations

We can raise a series of critiques and expand on the ones raised by the authors. Besides the poor performance, several challenges about scalability and applicability to the experimental scenarios remain open.

Non-markovianity of the model

The mapping of the decoding problem to a MDP presented by the authors is debatable. In a proper MDP, the transition and reward are sampled from a joint distribution that can depend solely on the current state and action, and the termination condition must be a function of the current state alone.

However, in the paper formulation, the rewards and the transitions are a function of the hidden quantum state. The hidden quantum state is correlated not only with the state of the MDP, that is the last five syndromes and corrections from last syndrome, but with all past syndromes and all past corrections. As a consequence, transitions, rewards, and termination are implicitly correlated with earlier states because of the correlation of those with the current hidden state. This violates the Markov property. As a result, the dynamics of the windows based state are fundamentally non-Markovian and the reinforcement learning machinery of MDPs cannot in principle be applied with success.

Inefficiency in information exploiting

The decoder operates on a fixed-size window of syndrome measurements. This design forces the agent to wait until the entire window is filled before taking any action, introducing unnecessary latency. Once the window slides forward, earlier syndrome data are discarded, resulting in the loss of precious information. Enlarging the window partially mitigates this issue but introduces new drawbacks: decisions are further delayed, and the dimensionality of the input space grows rapidly, which takes us to the third point.

Actions as single-qubit flips

Since the actions are single-qubit Pauli flips or the "request new syndrome" action, the Q-learning agent effectively learns to associate a syndrome volume with a specific error chain, given by the sequential single-qubit corrections starting from the state of the syndrome volume and empty history and continuing by the adding of flips to the state, rather than associating states to the homology class of the recovery chain.

This implies that, after a successful recovery operation has been found, precious exploration time is spent refining the shape of the error chain connecting syndrome defects, instead of moving on with the generation of new error configurations.

Inapplicability to the experimental case

We analyzed weather it would be possible to adapt the framework to the experimental case. Inferring the state in the experimental setup would require full quantum state tomography. This process is infeasible in real-time operation, as it demands many identical copies of the same state. We cannot have identical copies because even if we can independently prepare many copies of the same initial state,each copy would evolve according to a different trajectory. As a result, the reward signal used during training cannot be measured on actual hardware, which severely restricts the method's applicability beyond simulation.

Lack of inductive bias

In machine learning, "inductive bias" means a priori hypothesis on the learning target. Despite the fact that the noise model is known and used in training, the RL agent itself is trained tabula rasa, with no inductive bias on the noise. While this setup enables the decoder to generalize to arbitrary unknown noise, it also results in poor sample efficiency and slow convergence. Even in experimental settings some information about the noise model is usually available and could be incorporated into the policy or value architecture to improve learning. Ignoring this prior knowledge weakens the agent's performance and training efficiency. Since we are not achieving model agnosticism anyway, it is a waste not to incorporate the error model in the decoder.

Dependence on a referee decoder

The referee decoder imposes an artificial episode length determined by its own failure point. If the referee decoder prematurely returns an incorrect logical class, for example due to uncertainty in homology, it may end an episode even when the RL decoder could have recovered correctly with more time steps. This truncation of the reward signal introduces bias toward short-term decisions and restricts the RL agent from learning optimal long-horizon strategies. A better framework would allow the agent to use a independent stopping condition based on its own features.

Lack of full exploitation of surface code symmetries

The surface code exhibits several symmetries that can be exploited to reduce the number of independent parameters during training. Although the framework attempts to utilize translational symmetry via convolutional layers, this advantage is not fully exploited because of the last fully connected layers. A more systematic and consistent exploitation of these symmetries could significantly accelerate training.

5.2 The case for POMDP-based decoding

Reinforcement learning based decoders are indeed

The only way to describe decoding as a proper Markov decision process is to refer to hidden state of the quantum system as the decision process state. Since this state is not directly observable, decoding must be framed as a POMDP rather than a standard MDP. The surface code measurement cycle naturally resembles a hidden Markov model: syndrome measurements can be interpreted as emissions, and the error process as the hidden state transitions. This structural analogy suggests that mapping decoding to a POMDP is both feasible and justified.

This reframing requires a complete restructuring of the decoding framework from [2] to follow the POMDP formalism. In doing so, we also solve some of the other cited problems. In particular, the dependence on a referee decoder and on access to the hidden state become unnecessary. Within a POMDP, the agent is required to maintain a belief distribution over the hidden state space. This belief enables the agent to define internal reward and termination signals based solely on its current knowledge, making the framework applicable in real experimental settings.

5.3 Presentation of our framework

The decoding framework we present is not model-agnostic, as belief updates require knowledge of the underlying noise model. We find no viable way to perform truly agnostic decoding, and we see no strong reason to pursue it either.

In real hardware, the error model is measured during hardware calibration and is progressively updated during recalibration. Calibration and recalibration use approximated process tomography and pre-existing information to make an estimate of the error model. Notice that quantum process tomography in principle requires an exponential amount of operations in the system size, but there are polynomial approximation available.

Rather than assuming ignorance, we adopt a model-adaptive approach: the decoder make an internal stochastic approximation of the hardware error model that is updated with recalibrations as noise drifts over time. We use this internal noise model to train a reinforcement learning decoder decoder via simulated interactions and update the decoder to match recalibrations.

In this thesis we focus on toroidal qubits rather than planar ones due to the simpler topology of the toric code.

5.3.1 Structure

Our framework is organized following principles of object-oriented programming. It consists of global variables and modules. Each module encapsulates specific attributes and methods, which are detailed in later chapters. All components coordinate to implement four workflows.

The workflows are:

1. *Initialization*: initialize components from calibration data and train the decoder on simulated episodes via reinforcement learning.
2. *Decoding*: update the belief state based on observations and select recovery actions.
3. *Update*: perform quantum process tomography to refine the noise model and update the corresponding modules.
4. *Testing*: evaluate performance on benchmarks or simulated runs.

We cannot start the detailed explanation of the workflows without gradually developing the modules first.

The modules are developed through four steps:

- Identify a hidden Markov model that describes the dynamics of errors on encoded qubits.
- Design an efficient belief update mechanism from syndrome observations.
- Construct a valid POMDP over the HMM to represent decoding.
- Design a reinforcement learning pipeline to train an agent to solve the POMDP and evaluate its effectiveness in decoding.

The big challenge is to make sure every component of the framework remains polynomial in both representation size and runtime complexity.

We will show how to do this and then go back to the workflows in the last chapter.

6 The surface code cycle as a hidden Markov model

In order for POMDP-based decoding to be applicable, the quantum dynamics of the encoded qubits must be first be mapped to a classical hidden Markov model both in description and simulation. This reduction is in principle not possible under general conditions because of conceptual and computational problems, but under specific hypothesis and approximations can be achieved. This chapter identifies the regime under which such a mapping is possible and realizes it.

6.1 Quantum dynamics of the surface code cycle

We start by exact modeling the quantum dynamics of the surface code. As we have already said, the dynamics of the surface code obeys a cyclic pattern. We consider a system composed of data qubits and ancilla qubits evolving jointly in the Hilbert space of data and ancilla qubits $\mathcal{H}_D \otimes \mathcal{H}_A$. In order to describe states on it we need a coordinate system.

6.1.1 Lattice coordinates

Let Q denote the set of physical qubit indices. In a toroidal square surface code of distance d, we have $|Q| = 4d^2$.S Each qubit index $q \in Q$ corresponds to a coordinate pair (i, j) with $0 \leq i, j \leq 2d - 1$.

We partition Q into two disjoint subsets:

- $D \subset Q$: the set of *data qubits*
- $A \subset Q$: the set of *ancilla qubits*

Ancilla qubits are further subdivided according to the type of stabilizer they measure:

- $A_X \subset A$: the set of ancilla qubits associated with X-stabilizers
- $A_Z \subset A$: the set of ancilla qubits associated with Z-stabilizers

Let $L \subset Q \times Q$ denote the set of links between adjacent qubit pairs, where adjacency is defined by nearest-neighbor relations on the square lattice. The surface code is defined on a bipartite graph, where every link connects a data qubit to an ancilla qubit. For each pair $(q, q') \in L$, if q is an ancilla qubit then q' is one of its neighboring data qubits.

Thus, if $q = (i, j) \in A$, then

$$q' \in \{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\} \subset D,$$

and hence $L \subset A \times D$.

Due to the toroidal geometry of the code, each ancilla is connected to exactly four data qubits, even at the lattice boundaries. This is ensured by interpreting all coordinate arithmetic modulo $2d$, identifying $-1 \equiv 2d-1$ and $2d \equiv 0$.

The set of qubit to which a qubit q is connected is its neighborhood $\mathcal{N}(q)$.

We can define the directions $C = \{N, S, W, E\}$ corresponding to the four cardinal directions and the corresponding functions $\{N(a), ...\}$ returning the neighbor in the specified direction.

We also define the functions

$$\delta : L \to D, \quad \alpha : L \to A, \quad \lambda : Q \to L^4,$$

such that for any link $l = (a, d) \in L$, we have $\delta(l) = d$, $\alpha(l) = a$, and for any qubit $q \in Q$, $\lambda(q)$ is the set of links incident on q.

This notation avoids cumbersome double subscripts.

Using this notation we can partition also the link and the data qubit sets.

We can partition L into subsets based on direction c and stabilizer type s of the ancilla: $L(c, s)$ contains links oriented in direction c and associated with stabilizer type s.

We can divide the data qubits in D_x, D_z sets, based on whether the north neighbour $N(d)$ is a measure-X or measure-Z ancilla.

6.1.2 Structure of the cycle

Time is indexed by a pair (T, t), where: - T is the intercycle index (counting completed cycles), - t is the intracycle step. The time steps are separated by constant time intervals Δt.

The surface code cycle is modeled as follows:

- $t = 1$: Decoder finishes computing the recovery operation based on the syndrome, and recovery is applied to the data qubits.
- $t = 2$: Hadamard gates are applied to the ancilla qubits.
- $t = 3, 4, 5, 6$: At each time step, a set of non-overlapping CNOTs is applied to the lattice, following the scheduling north, east, south, west.
- $t = 7$: Final Hadamard gates are applied to the ancillas. This is the last gate operation of a cycle.
- $t = 8$: The cycle ends and a new cycle starts, with $T \mapsto T + 1$. Ancilla qubits are measured, producing the stabilizer syndrome, and are immediately reinitialized. The time-step index is then reset to $t = 0$ for the next cycle.

The state after step t is $\rho(T, t)$ except for $t = 8$ where is $\rho(T + 1{,}0)$

So the steps are indexed from 1 to 8, while the states from 0 to 7. Each event associated with these steps can be described by a quantum operation.

6.1.3 Transitions

We assume that all the quantum operations in the list occur instantaneously at their assigned time steps. Between these steps, background noise acts continuously and independently, modeled by a fixed quantum channel $\mathcal{N}$. This channel captures the effects of environmental noise and crosstalk.

Each time step corresponds to a *step transition*.

A step transition consists of two sequential components:

1. The application of the noise channel $\mathcal{N}$ to model background noise accumulated since the previous step.

2. The application of the step-operation channel performing the require quantum operation, possibly in a noisy way, generally noted by $\mathcal{C}_t$.

Measurement is also modeled as a quantum transition, unlike in classical hidden Markov models where observations are separate from transitions, because in the quantum formalism, every process is a quantum operation.

Therefore, the full step transition can be expressed by the step transition channel:

$$\mathcal{T}_t = \mathcal{C}_t \circ \mathcal{N},$$

6.1.4 Starting state

At the beginning of the procedure, the data qubits are prepared in the code state density operator λ, encoding the logical information we wish to protect. We thus identify the data density operator as

$$\rho_D(0,0) \;=\; \lambda.$$

Each ancilla undergoes the noisy initialization channel $\mathcal{S}$ independently. After this preparation, each single-qubit ancilla is in the state

$$\sigma_a \;=\; (1-p_i)\;|0\rangle\langle 0| \;+\; p_i\;|1\rangle\langle 1|\,,$$

where p_i is the ancilla's initialization error probability.

The joint state of the ancilla register is then

$$\rho_A(0,0) \;=\; \bigotimes_a \sigma_a.$$

Because the data and ancilla qubits evolve jointly, we consider the dynamics of the full system density operator $\rho(T,t)$ acting on $\mathcal{H}_D \otimes \mathcal{H}_A$. At the initial time, the joint system starts in the separable state

$$\rho(0,0) \;=\; \rho_D(0,0) \otimes \rho_A(0,0) \;=\; \lambda \otimes \bigotimes_{a\in\mathcal{A}} \sigma_a \;=\; \lambda \otimes \sigma^{\otimes m} \quad \text{(i.i.d. case)}.$$

After the initial preparation, the cycle begins.

6.1.5 Steps

Let's now describe the T-th cycle. We start from the state $\rho(T,0)$, coming from the last cycle, with syndrome $s(T) \in \{0,1\}^{|\mathcal{A}|}$, a binary vector indexed by the ancillas, measured in the last step.

In the case of the first cycle, $s(0)_a = 0$ for every a, independent of measurement faults.

At any step the state update is

$$\rho(T,t) = \mathcal{C}_t \circ \mathcal{N}\, \rho(T,t-1).$$

except for $t = 8$.

Step $t = 1$: Decoder correction

At the beginning of the cycle, the decoder computes the recovery operation based on the syndrome history. As we saw, the recovery operation consists of application of a Pauli chain, obtained experimentally as simultaneous application of Pauli gates.

The recovery channel $\mathcal{R}$ is applied to the data qubits, while the ancillas remain idle:

$$\mathcal{C}_\infty = \big(\mathcal{R}_D(s_T) \otimes \mathcal{I}_A\big) \circ \mathcal{N}$$

As said in section 2.3, we assume the Pauli gates are noiseless, so the recovery channel is too.

Step $t = 2$: Hadamard Gates on ancillas

Noisy Hadamard channels $\tilde{H}$ are applied to all ancilla qubits $a \in A_X$ corresponding to X-type stabilizer checks, while the identity channel acts on ancillas $a \in A_Z$ and data qubits.

$$\mathcal{C}_2 = \mathcal{I}_D \otimes \left(\bigotimes_{a \in A_X} \tilde{H}_a \otimes \bigotimes_{a \in A_Z} \mathcal{I}_a \right).$$

Steps $t = 3,4,5,6$: CNOT Entangling Operations

At each step, noisy CNOT gates act on ancilla-data pairs (a, d) in the specified direction. Let $d(t)$ be the direction associated to the time step.

$$\mathcal{C}_t = \bigotimes_{l \in L(d(t),X)} \widetilde{\mathrm{CNOT}}_{\alpha(l) \to \delta(l)} \otimes \bigotimes_{l \in L(d(t),Z)} \widetilde{\mathrm{CNOT}}_{\delta(l) \to \alpha(l)},$$

where directions and steps correspond as:

$$t = 3 \to \text{North}, \quad t = 4 \to \text{East}, \quad t = 5 \to \text{South}, \quad t = 6 \to \text{West}.$$

Step $t = 7$: Second set of Hadamard gates

The same update as in $t = 2$.

Step $t = 8$: Measurement and Reset of Ancillas

At $t = 8$, ancilla qubits $a \in A$ are measured with a uniform readout error rate p_r. The overall measurement channel acting on the full system is

$$\mathcal{I}_D \otimes \mathcal{M}_A, \tag{6.1}$$

Assuming there are no correlated the ancilla measurement map factorizes as a tensor product over each ancilla:

$$\mathcal{M}_A = \bigotimes_{a \in A} \mathcal{M}_a, \tag{6.2}$$

where $\mathcal{M}_a$ is the single-qubit faulty measurement channel. Assuming also uniform readout error p_r, we can define the POVM elements:

$$E_0 = (1 - p_r) \left|0\right\rangle_a \left\langle 0\right|_a + p_r \left|1\right\rangle_a \left\langle 1\right|_a, \tag{6.3}$$
$$E_1 = p_r \left|0\right\rangle_a \left\langle 0\right|_a + (1 - p_r) \left|1\right\rangle_a \left\langle 1\right|_a, \tag{6.4}$$

$$M_0 = \sqrt{1 - p_r} \left|0\right\rangle_a \left\langle 0\right|_a + \sqrt{p_r} \left|1\right\rangle_a \left\langle 1\right|_a, \tag{6.5}$$
$$M_1 = \sqrt{p_r} \left|0\right\rangle_a \left\langle 0\right|_a + \sqrt{1 - p_r} \left|1\right\rangle_a \left\langle 1\right|_a. \tag{6.6}$$

The overall POVM are the tensor product of the single ancilla POVM, and the overall measurement operators are the tensor product of the single ancilla measurement operators.

After measurement, a Bayesian renormalization $\mathcal{Z}$ restores the trace of the system state.

Each ancilla is then reset by the noisy initialization channel $\mathcal{I}_D \otimes \tilde{\mathcal{S}}_A$, which can be seen as a pure reinitialization of each ancilla to $\left|0\right\rangle_a \left\langle 0\right|_a$ (the $\mathcal{S}_A$ channel), followed by a single-qubit bit-flip channel on each ancilla, simulating ancilla initialization failure.

The full superoperator for this step is

$$\mathcal{C}_8 = \left(\mathcal{I}_D \otimes \tilde{\mathcal{S}}_A\right) \circ \mathcal{Z} \circ \left(\mathcal{I}_D \otimes \mathcal{M}_A\right), \tag{6.7}$$

and the system updates to the next cycle as

$$\rho(T+1,0) = \mathcal{C}_8 \circ \mathcal{N}\big(\rho(T,7)\big), \tag{6.8}$$

with syndrome $s(T+1)$.

After ancilla reset, one can obtain the state of data qubits by partial tracing away the ancillas.

6.2 Requirements of the mapping to classical model

To formulate decoding as a POMDP and obtain a decoder by training a policy, we must map the quantum dynamics of the surface code to a classical HMM that satisfies certain structural and computational requirements. The required properties are:

- **Finite state space:**

  To define the POMDPs, we restricted the underlying hidden Markov model to a finite state space. This restriction is computationally essential. When the set of hidden states is finite, the belief state a probability distribution over possible hidden states can be represented as a finite-dimensional vector. This representation enables numerical storage, update, and computation of expectations. If, instead, the hidden state space were countably infinite or continuous, the belief would become an infinite size vector or a function, and (at least) standard POMDP-solving techniques would not be applicable. Thus, from a computational standpoint, the assumption of a finite hidden state space is necessary to ensure that belief updates and decision-making remain tractable within the POMDP framework.

- **Sufficiency:** This state space must encode all the relevant information for decoding.

- **Polynomial scaling of the belief with code distance:** The memory required to store and manipulate the belief must scale polynomially with the code distance d. Since increasing d improves fault tolerance, a belief representation that scales exponentially would render decoding infeasible on classical hardware.

- **Markovian dynamics:** The hidden dynamics must satisfy the Markov property, so that the belief update depends only on the current belief and the most recent action and observation. This ensures that the process can be described within the MDP/POMDP formalism.

- **Independence from logical information:** The decoder must operate without knowing the information under error-correction, so both the state space and the transitions must be independent from it.

- **Polynomial-time belief updates:** All belief update operations must be computable in polynomial time in d. This guarantees that decoding can be performed in real time and that reinforcement learning methods remain computationally feasible during training.

- **Polynomial-time simulation of the hidden dynamics:** It must be possible to sample from the hidden dynamics in polynomial time in d, so that rollouts can be generated for both training of a reinforcement learning solution and testing of the learned policy.

- **Time homogeneity:** The HMM that supports the POMP must be time homogeneous, meaning that transitions and emissions are time independent, because the cycle repeats itself equally in time between decoding instances.

These conditions are sufficient to define, update, and simulate an HMM. While we do not impose that the dynamics follow the strict alternating structure of transitions and observations described in Chapter 3, it is desirable, especially for interpretability, that all quantum operations (channels, gates, measurements) be representable as compositions of stochastic transitions and Bayesian conditioning. However, the formalism can accommodate more general structures as long as the above computational requirements are met.

6.3 Stochastic approximations

In order to achieve a computationally treatable classical description is necessary that every channel is approximated by a stochastic channel. If coherent terms were considered, then the hidden state of the quantum system would be describe by a coherent superposition and the state space of the corresponding hidden Markov model could not be finite. In order to achieve this we need the dynamics of the hidden quantum space to be decomposed as a sequence of stochastic channels, Clifford channels and measurements (we will see why Clifford channels forward in this chapter). By exploiting the standard approximations employed in the literature all the channels in the quantum dynamics already fall under these prescriptions except for the cross-talk channel, which we now approximate.

6.3.1 Stochastic approximation of the cross-talk channel

The crosstalk energetic term evolves the system by the unitary:

$$U = \exp\left(-i\theta \sum_{l\in L} Z_{\alpha(l)} Z_{\delta(l)}\right),$$

where each $Z_{\alpha(l)}Z_{\delta(l)}$ acts nontrivially on qubits $\alpha(l), \delta(l)$ and as identity elsewhere. Since these operators commute,

$$[Z_{\alpha(l)}Z_{\delta(l)}, Z_{\alpha(l')}Z_{\delta(l')}] = 0, \quad \forall l, l' \in L,$$

the exponential factorizes as

$$U = \prod_{l\in L} \exp\big(-i\theta\, Z_{\alpha(l)} Z_{\delta(l)}\big).$$

Because each operator is idempotent $\big(Z_{\alpha(l)}Z_{\delta(l)}\big)^2 = I$, the Euler identity applies,

$$\exp\big(-i\theta\, Z_{\alpha(l)} Z_{\delta(l)}\big) = \cos\theta\, I - i\sin\theta\, Z_{\alpha(l)} Z_{\delta(l)},$$

and therefore

$$U = \prod_{l\in L} \big(\cos\theta\, I - i\sin\theta\, Z_{\alpha(l)} Z_{\delta(l)}\big).$$

The resulting unitary channel acting on a density matrix ρ is

$$\mathcal{N}_{\text{crosstalk}}(\rho) = \left(\prod_{l\in L} \big(\cos\theta\, I - i\sin\theta\, Z_{\alpha(l)} Z_{\delta(l)}\big)\right) \rho \left(\prod_{l\in L} \big(\cos\theta\, I + i\sin\theta\, Z_{\alpha(l)} Z_{\delta(l)}\big)\right).$$

We approximate this channel by keeping only the terms where the same product of chains acts on both sides of ρ, corresponding to sampling a chain S, written as a subset of links, by independently sampling each edge $l \in L$ with probability

$$p_c = \sin^2\theta,$$

resulting in the stochastic Pauli channel

$$\mathcal{N}_{\text{crosstalk}}(\rho) \approx \sum_{S \subseteq L} \left(\prod_{l \in S} p_c \prod_{l \notin S} (1 - p_c) \right) P_S \rho P_S,$$

where

$$P_S = \prod_{l \in S} Z_{\alpha(l)} Z_{\delta(l)}.$$

There are other diagonal terms corresponding to cases where different chains on the left and right of the conjugation share the same endpoints but differ in the middle. These terms are neglected here because taking them into account would require a disproportionate amount of work relative to their importance in this thesis. This approximation already represents an improvement over the current literature. A full Pauli twirling of the channel to treat these terms will be addressed in future work.

6.3.2 Stochastic approximation of the noise channel

Since stochastic Pauli channels commute with each other, and both environmental noise and cross-talk noise are approximated as stochastic Pauli channels, we can write the overall noise channel as the composition

$$\mathcal{N} = \mathcal{N}_{\text{crosstalk}} \circ \mathcal{N}_{\text{env}},$$

without needing to worry about the exact chronological order in which the noise events occur.

The composite noise channel $\mathcal{N}$ is itself a stochastic Pauli channel.

6.3.3 Error model

All the parameters of the stochastic channels by which we model the noise affecting the system are encoded into the *error model*. During the initialization workflow, the error model is initialized with the values coming from the calibration of the hardware. The hardware calibration return the parameters for a known fixed class of errors affecting the system. During the update workflow is updated with the information coming from quantum process tomography. The error model in detail is:

- **Background noise:**
  - *Environment-induced noise:* single-qubit depolarizing noise acting independently on each qubit. Parameter: p_d
  - *Crosstalk:* Correlated Z acting on neighbouring qubits. Parameter: p_c
- **Gate noise:**
  - *Hadamard gate noise:* single-qubit depolarizing noise following Hadamard operations. Parameter: p_h
  - *CNOT gate noise:* two-qubit depolarizing noise following CNOT operations. Parameter: p_n
- **Initialization errors:**
  - *Ancilla initialization failure:* preparation of $|1\rangle$ instead of $|0\rangle$. Parameter: p_i
- **Measurement errors:**
  - *Readout errors:* probability of bit-flipping a measurement outcome. Parameter: p_r.

6.4 Structure of the mapping

We found that we cannot perform a simple mapping to a HMM, but we need to pass between multiple interconnected HMMs before being able to fully define our POMDP. We call the hidden Markov model needed to support the decoding POMDP the *decoding-HMM*, and it will be one of the eight possible HMMs we can define. First of all, the decoding HMM does not need to track the full quantum dynamics at every instant. It is sufficient to have a belief on some informative state of the data qubits right after measurement only, at the so called *decoding times* $(T,0)$, i.e. we want to be able to track:

$$\rho_D(T,0) = \mathrm{Tr}_A\, \rho(T,0),$$

together with the corresponding syndrome s_T. Thus, the time index of the decoding-HMM naturally runs over the cycle index T.

However, to determine these states we still need to evaluate the full code dynamics at every step within the cycle, in the joint data-ancilla Hilbert space, by classical computation. This is because the evolution of the data density matrix between decoding times depend of the physical processes that happen at intracycle scale, that involve error-spreading processes between ancillas and data qubits. We can thus distinguish two features of the tracking HMMs: they can be *joint* or *data*, based on the qubits they track, and they could be *intercycle* or *time-complete* based on whether they track all the times or just T,0 times.

Tracking $\rho(T,t)$ is done by the joint time-complete HMM, from which we derive the data intercycle HMM by selecting just the after measurement time steps and tracing out the ancillas. The time-complete HMMs do not need to be time homogeneous as long as any time dependence is confined within the cycle. The only time dependence allowed is on the intracycle time t. The decoding-HMM needs to be completely time homogeneous instead.

The way we solve the mapping problem is to impose a physically principled ansatz on $\rho(T,t)$.

6.5 DPM ansatz

The hidden state is quantum: it is described by a density matrix ρ_D, which evolves under quantum channels and measurements. While the density matrix is already a belief over hidden quantum states, we cannot directly use it. Even if any density matrix is actually a classical probability distribution over some distinguishable state space, this set of states depends on the density matrix and changes during the evolution. Furthermore, computing this set is not feasible because it requires diagonalizing a matrix with a number of entries exponential in the code distance. So there is no straightforward way to define a proper finite state space for the POMDP. To overcome this, we use intuition.

Since we are twirling all the noise channels and the remaining channels are either Clifford, Pauli, or measurement, we can formulate the hypothesis that after repeated applications of such channels, at any time (T,t), the density matrix of the data can be expressed as a mixture of Pauli strings conjugated with the initial state λ. This is different than the general χ-matrix representation of the resulting state of the system because in this case the χ-matrix is constrained to be diagonal and with real diagonal terms forming a proper probability distribution.

In general, during the stochastic dynamics any string of single-qubit Pauli errors can be applied in any order, but we can reorder them in standard form and any phase factors would cancel out between the two sides of the conjugation.

This idea leads to the *DPM ansatz* (meaning Decoding by Pauli Marginals). The ansatz consists of the following five points:

6.5.1 Mixture of Pauli conjugation form of the joint density

For any cycle index T and intra-cycle time t, the joint density can be written as:

$$\rho(T,t) \;=\; \sum_{P\in\mathcal{P}_Q} B_{T,t}(P)\; P\,\rho(t,\lambda)\,P,$$

where $B_{T,t}$ is a probability distribution over the joint space Pauli basis $\mathcal{P}_Q$, called the *joint time-complete belief*, and $\rho(t,\lambda)$ depends only on the intra-cycle time t and the logical state λ, and is equal to the encoded logical qubits entangled with ancillas by the gates of the surface code.

We call a term of the form $P\rho(t,\lambda)P$ a Pauli conjugation of the cyclic state.

Notice: this is different than the normal

This means that the time-complete dynamics of the joint space can be described by a candidate time-dependent HMM defined on the finite state space:

$$S_{dynamics} = \mathcal{P}_Q$$

We can represent any of these states by their frames, organized in a binary tensor of shape $(2d, 2d, 2)$, where the first two dimensions correspond to the lattice coordinates of each qubit $q = (i, j) \in Q$, and the last dimension encodes the type of Pauli operator. Specifically,

$$f_{i,j,w} = \begin{cases} 1 & \text{if Pauli } w \in \{X, Z\} \text{ acts on qubit } q = (i, j), \\ 0 & \text{otherwise} \end{cases}$$

For any qubit $q \in Q$, the vector

$$f_q = (e_{q,X}, e_{q,Z})$$

denotes the 2-component error vector acting on that qubit.

Given a tuple of qubit indices, $\vec{t} = (q_1, q_2, \ldots, q_{|t|}) \subseteq Q$, we define the partial frame as

$$f_{\vec{t}} = \begin{bmatrix} f_{q_1} \\ f_{q_2} \\ \vdots \\ f_{q_{|t|}} \end{bmatrix}$$

and single type partial frame as

$$f_{\vec{t},s} = \begin{bmatrix} f_{q_1,s} \\ f_{q_2,s} \\ \vdots \\ f_{q_{|t|},s} \end{bmatrix}$$

This applies also to links (a link is tuple of qubits). Given any partial frame $f_{\vec{t}}$ the corresponding full frame extension $F(f_{\vec{t}})$ is the frame where we augment $f_{\vec{t}}$ with the other components set to 0.

A partial frame can be equivalently be extended to any domain U, by augmentation with zeros by the domain extension operator $F^U(f_{\vec{t}})$.

The belief is simply a normalized probability distribution over all possible Pauli frames f, represented as a vector:

$$B = \{B(f)\}_{f\in\mathcal{F}_{4d^2}},$$

where $\mathcal{F}_{4d^2}$ denotes the set of all the frames of the Pauli error configurations over the full lattice. The total number of possible Pauli error chains is

$$2^{2|Q|} = 2^{8d^2}$$

So the number of states grows exponentially with the distance. We call a belief over the joint space a *joint belief*.

The exponential size of the error configuration space renders a direct representation of the belief distribution intractable. A full belief vector would have $|E| - 1$ degrees of freedom, making storage, update, and inference computationally unfeasible. We will overcome this issue not directly working with this belief.

6.5.2 Separability before measurement

Immediately before ancilla measurement, the joint state is separable and can be written as

$$\rho_{\text{premeasurement}}(T,7) \;=\; \sum_{P_D\in\mathcal{P}_D}\sum_{P_A\in\mathcal{P}_A} B_{T,0}(P_D\otimes P_A)\left(P_D\,\lambda\,P_D^\dagger\right)\;\otimes\;\left(P_A\bigotimes_a|0\rangle\langle 0|_a\;P_A^\dagger\right).$$

where $\mathcal{P}_D$ and $\mathcal{P}_A$ are the Pauli bases on data and ancilla qubits.

This state is $\rho(T,7)$ after the application of the background noise. This separability property is a direct consequence of the surface-code circuit and noise design: ancillas are disentangled from the data in the last step before measurement. If they were not, the ancilla measurement would collapse and corrupt the information stored in the data qubits. By the no-cloning theorem, the ancilla cannot acquire any information about the data without becoming entangled; since disentanglement is enforced before measurement, the ancilla state is (up to Pauli conjugations) identical to its initial state. Since background noise is not entangling, the joint state stays separable upon measurement. Measuring the ancillas induces a Bayesian update on the joint belief $B_{T,0}$, without effects on the cyclic state. This implies that immediately after measurement, the ancillas are reset, disentangled and can be traced out, yielding the data intercycle state:

$$\rho_D(T,0) \;=\; \sum_{P_D\in\mathcal{P}_D} b_T(P_D)\;P_D\,\lambda\,P_D,$$

where the weights b_T are called *data intercycle belief*. Since $\rho_(T,0)$ is in the form of Eq. 1.1, this belief can be obtained as the result of marginalization over the ancillas:

$$b_T(P_D) \;=\; \sum_{P_A\in\mathcal{P}_A} B_{T,0}(P_D\otimes P_A).$$

We refer to each $P_D\,\lambda\,P_D$ as a Pauli conjugation of the initial state. If markovian, the dynamics of the data intercycle state can be described by the data intercycle hidden markov model, defined on the state space:

$$S_{\text{decoding}} \;=\; \mathcal{P}_D.$$

In principle, the emission and transition kernels of this reduced decoding HMM could be derived explicitly from those of the full HMM. In practice, this is unnecessary: we can simply update the time-complete joint belief in its HMM and then marginalize to obtain b_T. The only requirement for this reduction to be valid is that the underlying process is Markovian.

6.5.3 Markovianity of the Pauli conjugations

The dynamics of the quantum hidden state are markovian, as they proceed through a sequence of quantum operations. Quantum mechanics is physically memoryless. This implies that, if the ansatz is verified, the dynamics of the Pauli conjugations on the cyclic state can depend just on previous state and λ. If we prove that the distribution on the Pauli conjugations of the cyclic state evolves in time independently of λ then we can confirm the existence of a hidden Markov model for both time-complete joint and intercycle data dynamics.

These three parts of the ansatz will be proved by induction: first we show that an ansatz state is mapped markovianily to another ansatz state by the quantum dynamics, and then we show the initial state follows the ansatz.

6.5.4 Decoding by Pauli marginals

A recovery decision does not require access to the full distribution $b_T(P_D)$ over all Pauli chains of the data qubits P_D. The only relevant information for decoding is the *homology class* $\mathcal{H}$ of the Pauli conjugations.

Since the relation between Pauli chains and homology classes is many-to-one, we could in principle define a homology belief as the marginal distribution of the intercycle data belief over homology classes. However, there is no efficient way to compute or even approximate this marginal distribution, so there is no need to introduce additional notation.

The belief over the Pauli conjugations uniquely determines homology belief, so we can say it's a sufficient statistics for decoding. So the decoding HMM is the intercycle data only hidden markov model. The problem is the exponential state space, which makes the belief management impossible.

We need to resort to an approximation. In our framework, all noise processes are either single-qubit errors or pairwise-correlated errors between neighboring qubits, and the correlated spread of errors is confined to local neighborhoods.

This allows us to address the problem by assuming that only a subset of marginals of the joint belief constitutes a sufficient statistic for decoding. Because there are no long-range correlations in the considered noise models, we can expect it to be close to optimal, and thus we focus on these limited marginals. The parameters to describe the marginals are polynomial.

This means that the belief MPD we use the solve the decoding POMPD is defined on marginals rather the full belief. The set of marginals is denoted by $\vec{m}$.

List of marginals of interest

In principle we can consider any set of marginals, as long as we are capable of updating them without having to marginalize the full belief. See next section for how this constraints the marginal selection. In practice we are interested just in the marginals who track correlation that we have a physical explanation for, assuming the other marginals would be redundant.

Because depolarizing noise introduces correlations between X and Z error components, the marginals cannot be factorized into separate components. The CSS structure of the surface code cannot be exploited to decode X and Z errors independently, and each belief marginal must represent a joint (X, Z) distribution. The set of marginals considered is minimal in the sense that we do not include marginals obtainable by marginalizing a larger one.

The widest marginals are induced by the CNOT gates, which spread X errors from control to target and Z errors from target to control. We need to update the marginals during the time-complete dynamics in the joint space, which is where and when the error-spreading phenomena happen.

We denote marginals on the joint space on a domain R as

$$M_{(T,t)}[R](f_R),$$

and marginals on the data qubit space (single type) as

$$m_T[R, s](f_{R,s}),$$

where s is the remaining Pauli type. Here the domain R indicates the set of qubits for which the marginal is computed. Joint-space marginals are considered for time complete dynamics, so

they are associated to both the intercycle time T and the sub-step t, while data-space marginals are associated only to the intercycle time T.

- **Five-qubit marginals:** induced by the CNOT gates connecting ancillas with their neighborhoods:
  - Targeting the measure-Z ancillas (Z-type spreader):

    $$\forall a \in A_Z, \quad M_T[\{a\} \cup \mathcal{N}(a), Z](f_{\{a\}\cup\mathcal{N}(a),Z})$$

    Each marginal has $2^5 = 32$ components, and there are exactly d^2 such ancillas.
  - Targeting the measure-X ancillas (X-type spreader):

    $$\forall a \in A_X, \quad M_T[\{a\} \cup \mathcal{N}(a), X](f_{\{a\}\cup\mathcal{N}(a),X})$$

    Same count.
- **Three-qubit marginals (data-centred, ancilla–data–ancilla lines):**
  - For data qubits with a north X-type ancilla:

    $$\forall d \in D_X, \quad \begin{cases} m_T[\{N(d), d, S(d)\}, Z](f_{\{N(d),d,S(d)\},Z}) & \text{(vertical)}, \\ m_T[\{W(d), d, E(d)\}, X](f_{\{W(d),d,E(d)\},X}) & \text{(horizontal)}, \end{cases}$$

    Each marginal has $2^3 = 8$ components. Toroidal conditions give exactly d^2 vertical and d^2 horizontal marginals.
  - For data qubits with a north Z-type ancilla:

    $$\forall d \in D_Z, \quad \begin{cases} m_T[\{N(d), d, S(d)\}, X](f_{\{N(d),d,S(d)\},X}) & \text{(vertical)}, \\ m_T[\{W(d), d, E(d)\}, Z](f_{\{W(d),d,E(d)\},Z}) & \text{(horizontal)}, \end{cases}$$

    Each marginal has $2^3 = 8$ components. Exact counts: d^2 vertical and d^2 horizontal marginals.
- **Two-qubit marginals:** induced by depolarizing CNOT noise. They correlate all possible Pauli pairs (W_i, W_j) and cover also cross-talk and single-qubit environmental noise via marginalization:

  $$\forall l \in L, \quad m_T[l](f_l)$$

  Each 2-qubit marginal has $4^2 = 16$ components. Toroidal conditions give exactly $8d^2$ link marginals.

This set of marginals is minimal in the sense that no marginal can be derived from the others. The total number of parameters is $O(d^2)$.

Conditional marginals

In principle we can and should also consider *conditional marginals*, which we write as $M[R|Q](f_R)$ and eventually $M[R, s|Q](f_R)$ or $M[R|Q, s](f_R)$ if the conditioned or condition are single type.

Recovery chains do not introduce long range correlations

Recovery chains may connect arbitrarily far marginals because their actions is deterministic.

6.5.5 Markovianity of the marginals

Even when referred to marginals, for decoding we are still interested to the data only marginals at intecycle times, $\vec{m}_T$

The marginals are in principle not a sufficient statistics for the belief dynamics of a POMPD, so the dynamics of the marginal MPD should not be markovian. The same marginal can have different type of transition based on the state of marginalized variables. However, because the noise model we are considering can directly map marginals to marginals, in this case the marginals dynamics is markovian. This must be proven.

This means that even for marginals we must consider then joint space time complete marginals $\vec{m}_{(T,t)}$, compute explicitly their evolution, and then marginalize away ancillas at decoding times. The marginals of interest for decoding are the marginals of the list we gave. The three-qubit marginals are collapsed into one qubit on Pauli type marginals, while the two-qubit are collpsed into one qubit two variable marginals. The ex-three qubits marginals become redundant after marginalization of ancillas and just useful as a consistency check. Decoding is thus a function of 4-qubit single Pauli type neighborhood marginals and single qubit two Pauli type marginals.

6.5.6 Summary

We can summarize the whole landscape.

- decoding can be mapped to a POMDP supported on a intercycle data only hidden Markov model with state space given by Pauli conjugations of the data inital state.
- this hidden Markov model is actually markovian because it is derived from another hidden Markov model, the time-complete joint space hidden Markov model
- the markovianity of this last HMM can be proven by plugging a mixture of Pauli conjugations into the stochastic channels describing the quantum dynamics. It is derived from the markovianity of the quantum dynamics and the stochasticity of the channels.
- the decoding POMP is computationally impossible to solve in its naive form, even by RL.
- because of this, we define a marginal belief intercycle data MPD (marginal MPD in the following) which derives its markovianity from the decoding POMDP. Our policy is relative to this MDP and can be used as a decoder.

We call the overall ansatz *DPBM ansatz* (decoding by Pauli belief marginals ansatz). It is the set of approximations that we need to do to frame decoding as solvable POMDP that is actualy markovian and consistent.

6.6 Dynamics of the HMMs

We now describe the evolution of the HMM belief and marginals throughout a surface code cycle, showing that states in ansataz are mapped to states in the ansataz.

6.6.1 Starting state

At the beginning of the procedure, the data qubits are prepared in the code-state density operator λ, encoding the logical information we wish to protect:

$$\rho_D(0,0) = \lambda.$$

Each ancilla undergoes the initialization channel, resulting in a single-qubit state

$$\sigma_a = (1 - p_i)\ |0\rangle\langle 0| + p_i\, X_a\, |0\rangle\langle 0|\, X_a,$$

where p_i is the initialization-error probability of ancilla a, and X_a is the Pauli-X acting on that ancilla.

The joint system density operator at the initial time is therefore

$$\rho(0{,}0) = \sum_f B_{0,0}(f)\, P(f)\, \lambda\, P(f) \bigotimes_{a \in A} |0\rangle\langle 0|_a\,.$$

For simplicity, we write $|0\rangle\langle 0|^{\otimes A}$ to denote the joint state of all ancillas in $|0\rangle$.

The time-complete joint belief is

$$B_{0,0}(f) = \begin{cases} 0, & f_D \neq \vec{0} \text{ or } f_{A,z} \neq \vec{0}, \\ \displaystyle\prod_{a \in A} (1 - p_i)^{1 - f_{a,x}}\, (p_i)^{f_{a,x}}, & f_D = \vec{0},\ f_{A,z} = \vec{0}. \end{cases}$$

The starting data intercycle belief is its marginal:

$$b_0(f_D) = \begin{cases} 1, & f_D = 00\ldots0, \\ 0, & \text{otherwise.} \end{cases}$$

The time-complete joint marginals at the initial step are

$$M_{(0,0)}[R](f_R) = \begin{cases} \displaystyle\prod_{a \in R \cap A} (1 - p_i)^{1 - f_{a,x}} (p_i)^{f_{a,x}}, & f_{R \cap D} = \vec{0},\ f_{R \cap A, z} = \vec{0}, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall R \subseteq Q.$$

Finally, the data intercycle marginals at $T = 0$ are

$$m_0[R](f_R) = \begin{cases} 1, & f_R = 00\ldots0, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall R \subseteq D.$$

6.6.2 Transition elements

Computing the update of a belief requires performing stochastic transitions of the general form:

$$B'(f) = \sum_{f'} T(f, f')\, B(f'),$$

where $T(f, f')$ is the full transition kernel.

These full kernels are usually difficult to work with in simulations, computations, or even mathematical definitions. Luckily, the transitions we encounter can often be simplified as iterated applications of *transition elements*: specific, manageable cases that, when compounded, form the complete transition.

There are two kinds of transition elements we typically encounter: *noise transition elements* and *rule-based transition elements*. We define the rule to understand and handle both.

We will denote by U the domain of the transition element, i.e. the set of all qubits involved in the element.

Noise elements

Noise elements are additive operations that apply flips on a domain U according to a distribution independent of the current configuration of the qubits in U. Let $e_U \in \{0,1\}^{2|U|}$ denote a sampled flip configuration, and let $\tau(e_U)$ be the probability of sampling it. Since noise elements are stochastic extractions of Pauli flips, they map ansatz states to ansatz states.

Because the transition kernel is independent of the input frame, the updated belief can be written as a circular convolution:

$$B'(f) = \sum_{e_U \in \{0,1\}^{2|U|}} \tau(e_U)\, B(f \oplus F(e_U)).$$

This sum scales polynomially with the size of the domain U, so if the domain size does not grow with the lattice, the update remains efficient. However, the convolution must be performed independently for every configuration f, which becomes costly for the full joint belief over all qubits.

If instead of the full belief we consider just the marginals defined on a qubit set R, denoted $M[R]$, we need to consider how R and U overlap. Let f_R denote a frame variable on R. There are three cases:

1. **No overlap:** $R \cap U = \emptyset$.
   The marginal remains unchanged:
   $$M'[R](f_R) = M[R](f_R).$$
2. **Full overlap:** $U \subseteq R$.
   Update the marginal by summing over all flips in U, extending each to R:
   $$M'[R](f_R) = \sum_{e_U \in \{0,1\}^{2|U|}} \tau(e_U)\, M[R]\big(f_R \oplus F^R(e_U)\big),$$
   where F^R denotes the extension of the flips e_U to the domain R.
3. **Partial overlap:** $R \cap U \neq \emptyset$ and $U \nsubseteq R$.
   Let $O = R \cap U$ denote the overlap. First, marginalize the noise kernel to the overlap:
   $$\tau_O(e_O) = \sum_{e_{U\setminus O}} \tau(e_O, e_{U\setminus O}),$$
   where $(e_O, e_{U\setminus O})$ denotes concatenation. Then update the marginal using the extension to R:
   $$M'[R](f_R) = \sum_{e_O \in \{0,1\}^{2|O|}} \tau_O(e_O)\, M[R]\big(f_R \oplus F^R(e_O)\big).$$

This explains what happens to full marginals. In case the marginals are single-type, we must marginalize away the opposite type from the noise element transition kernel to obtain a single-type kernel and then apply this picture.

There are three kinds of noise element we consider.

Single-qubit depolarizing noise element Let $U = \{q\}$ and p_d the depolarizing probability. The transition kernel is

$$\tau(e_q) = \begin{cases} 1 - p_d, & e_q = 00, \\ \frac{p_d}{3}, & e_q \in \{01,10,11\}. \end{cases}$$

Marginal update (only if $q \in R$):

$$M'[R](f_R) = \sum_{e_q} \tau(e_q)\; M[R](f_R \oplus F^R[e_q]).$$

Single-qubit bitflip noise element Let $U = \{q\}$ and p_i the depolarizing probability. The transition kernel is

$$\tau(e_q) = \begin{cases} 1 - p_i, & e_q = 00, \\ \frac{p_i}{3}, & e_q \in \{01,10,11\}. \end{cases}$$

Marginal update (only if $q \in R$):

$$M'[R](f_R) = \sum_{e_q} \tau(e_q)\; M[R](f_R \oplus F^R[e_q]).$$

there is no possible partial overlap.

Two-qubit depolarizing noise element Let $U = \{q_1, q_2\}$ and p_n the depolarizing probaiblity:

$$\tau(e_{q_1,q_2}) = \begin{cases} 1 - p_n, & e_{q_1,q_2} = 0000, \\ \frac{p_n}{15}, & e_{q_1,q_2} \neq 0000. \end{cases}$$

Marginal update (only if $U \cap R \neq \emptyset$):

- **Both qubits in** R:

$$M'[R](f_R) = \sum_{e_{q_1,q_2}} tau(e_{q_1,q_2})\; M[R](f_R \oplus F^R[e_{q_1,q_2}]).$$

- **Exactly one qubit in** R:

$$M'[R](f_R) = \sum_{e_q} \Big[\sum_{e_{q'} \notin R} \tau(e_{q,q'}) \Big]\; M[R](f_R \oplus F^R[e_q]),$$

  where $q \in R$ and $q' \notin R$.

Two-qubit ZZ noise element Let $U = \{q_1, q_2\}$ and p_c the crosstalk probability:

$$\tau(e_{q_1,q_2}) = \begin{cases} 1 - p_c, & e_{q_1,q_2} = 00, \\ p_c, & e_{q_1,q_2} = zz, \\ 0, & \text{otherwise.} \end{cases}$$

Marginal update (only if $U \cap R \neq \emptyset$): Same equations as two-qubit depolarizing. For single-type marginals, first marginalize away the opposite type in τ, then update using

$$M'[R](f_R) = \sum_{e_O \in \{0,1\}^{2|O|}} \tau_O(e_O)\, M[R]\big(f_R \oplus F^R(e_O)\big),$$

where $O = R \cap U$.

Rule-based elements

Rule-based elements are the opposite of noise elements: they update the frame by an input-dependent deterministic rule. Let

$$R : S_{\text{dynamics}} \to S_{\text{dynamics}}$$

be a function expressing the rule (a mapping from frames to frames). The belief then updates as

$$B'(f') = \sum_f B(f)\,\delta\big(f', R(f)\big).$$

If the rule is injective, this simplifies to

$$B'(f) = B\big(R^{-1}(f)\big)\,.$$

Usually, not all frame components are needed to determine the effect of the rule, and not all frame components are affected by it. Let U_1 denote the minimal subset of qubits determining the rule's outcome, and U_2 the minimal subset of qubits that are affected. Define the *rule domain* U:

$$U = U_1 \cup U_2.$$

The rule can then be represented locally by a transition element kernel

$$\tau : \{0,1\}^{|U|} \times \{0,1\}^{|U|} \longrightarrow \{0,1\},$$

with $\tau(e_U, e'_U) = 1$ if e'_U is the updated configuration of e_U, and 0 otherwise.

Writing a frame configuration as

$$f = (f_{\text{rest}}, e_U),$$

where f_{rest} denotes all components outside U, the full belief update is

$$B'(f_{\text{rest}}, e_U) \;=\; \sum_{e'_U} \tau(e_U, e'_U)\, B(f_{\text{rest}}, e'_U).$$

Where τ is a binary matrix checking the $\delta\big(f', R(f)\big)$ on U.

If τ is invertible, the sum collapses and the update simplifies to

$$B'(f_{\text{rest}}, e_U) \;=\; B\big(f_{\text{rest}}, \tau^{-1}(e_U)\big).$$

The update at a given frame is polynomial, but to update the whole belief it must be performed on exponentially many components. Hence, we cannot update the global belief directly, but instead restrict to marginals.

Let R denote the domain of a marginal of Pauli type. The effect of a rule element on a marginal depends on the overlap between R and the rule domain U.

1. No overlap ($R \cap U = \emptyset$): The rule does not affect the marginal:

$$m'[M] = m[M].$$

2. Full inclusion ($U \subseteq R$): The marginal update can be expressed explicitly using the same kernel restricted to U:

$$M'[R](f_{R\setminus U}, e_U) \;=\; \sum_{e'_U} \tau(e_U, e'_U)\, M[R](f_{R\setminus U}, e'_U).$$

3. Partial overlap with U_2, full inclusion of U_1: The marginal contains all components determining the rule, but not all components affected. Restrict the kernel to U_1 by dropping the update of qubits in $U_2 \setminus U_1$:

$$\tau \;\to\; \tau^{U_1},$$

then perform the update as in the full inclusion case:

$$M'[R](f_{R\setminus U_1}, e_{U_1}) \;=\; \sum_{e'_{U_1}} \tau^{U_1}(e_{U_1}, e'_{U_1})\, M[R](f_{R\setminus U_1}, e'_{U_1}).$$

4. Partial overlap with U_1, full with U_2: This case is more complicated. The marginal R contains all components affected by the rule, but not all components that determine it. Performing the update requires an auxiliary marginal for the part of U_1 outside R. This marginal cannot be free and independent, but must be conditioned on R, because a free marginal would return a probability obtained summing over configurations of R, while we use the marginal at fixed configuration, creating an inconsistency.

   Let $M_{\text{aux}}[U_1 \setminus R|R]$ provide a full distribution for the part of U_1 not in R. Here $\tau(e'_{U_1}, e_{U_2})$ is the (possibly non-square) transition kernel.

$$M'[R](e_{R\setminus U_2}, e_{U_2}) = \sum_{e'_{U_1}} \tau(e'_{U_1}, e_{U_2})\; M[R](e_{R\setminus U_1}, e_{U_1\cap R})\; M_{\text{aux}}[U_1 \setminus R \mid R](e_{U_1\setminus R}).$$

   The way this update is performed is by iteratively fixing a value of $e_{R\setminus U_2}$ *and* e_{U_2} and then summing over the values of e'_{U_1}.

   Practical approximation for the partial-overlap case. Tracking the exact conditioned marginals $M(e_{U_1\setminus R} \mid e_R)$ for every region R can lead to an explosion of distinct conditioned marginals (we call it marginal proliferation), which may reduce this scheme to full belief tracking. We cannot study this issue in too much detail to avoid exploding the thesis length. Instead we propose a Bethe-like approximation: take the wider subset $C \subset R$ for which conditioning is feasible (because it exploits other marginals we are already tracking) and approximate the unknown conditional by conditioning only on C:

$$M_{\text{aux}}[\,U_1 \setminus R \mid R\,](e_{U_1\setminus R} \mid e_R) \;\approx\; M_{\text{aux}}[\,U_1 \setminus R \mid C\,](e_{U_1\setminus R} \mid e_C).$$

   The approximation is reasonable when the coupling (or correlations) between $U_1 \setminus R$ and the remainder of R outside C is weak and exact when it is null (e.g. exponential decay of correlations, small interaction range, or when C is the Markov blanket).

5. Partial overlap with U_1 and partial overlap with U_2: In this case, we take the non-square transition kernel $\tau(e'_{U_1}, e_{U_2})$ and restrict it to $U_2 \cap R$ by dropping the rows associated with $U_2 \setminus R$. We then proceed as in the previous case (partial overlap with U_1, full with U_2):

In case we consider marginals of just a stabilizer s, it is sufficient to interpret sets U_1 and U_2 as sets of components rather than qubit, and apply this rule, in particular cases 2 and 4 in dropping the components relative to the opposite stabilzier

The rule-based transitions we consider are the following:

Hadamard rule (single-qubit) For a single-qubit Hadamard H_q, the Pauli frame components of qubit q swap:

$$(f_{q,X}, f_{q,Z}) \;\longrightarrow\; (f_{q,Z}, f_{q,X}).$$

The associated transition kernel $\tau^{\{q\}}(f', f)$ encodes this deterministic update:

$$\tau^{\{q\}}(f', f) = \delta\big(f'_q - (f_{q,Z}, f_{q,X})\big) \prod_{r\neq q} \delta(f'_r - f_r).$$

Its inverse coincides with itself:

$$\tau^{\{q\}\dagger} = \tau^{\{q\}}.$$

U_1 this transition element is single qubit, so for every marginal is either included or not, without partial overlap.

CNOT rule (two-qubit) For a CNOT with control q_1 and target q_2, the Pauli frame updates as:

$$f'_{q_2,X} = f_{q_2,X} \oplus f_{q_1,X}, \qquad f'_{q_1,Z} = f_{q_1,Z} \oplus f_{q_2,Z},$$

with other components unchanged.

The corresponding transition kernel $\tau^{\{q_1,q_2\}}$ is

$$\tau^{\{q_1,q_2\}}(f', f) = \delta(f'_{q_1} - (f_{q_1,X}, f_{q_1,Z} \oplus f_{q_2,Z}))\, \delta(f'_{q_2} - (f_{q_2,X} \oplus f_{q_1,X}, f_{q_2,Z})) \prod_{r \notin \{q_1,q_2\}} \delta(f'_r - f_r).$$

Its inverse is identical, since CNOT is self-inverse:

$$\tau^{\{q_1,q_2\}\dagger} = \tau^{\{q_1,q_2\}}.$$

In our setting each update rule acts on two qubits. If U_1 partially overlaps a tracked marginal R, then $U_1 \setminus R$ is a single qubit and $U_1 \cap R$ is the other. We can therefore obtain the required auxiliary conditional directly from the two-qubit marginal on U_1:

$$M_{\text{aux}}[\, U_1 \setminus R \mid C\,](e_{U_1 \setminus R} \mid e_C) \;=\; \frac{M[U_1](e_{U_1 \setminus R}, e_C)}{\sum_{e_{U_1 \setminus R}} M[U_1](e_{U_1 \setminus R}, e_C)}, \qquad C := U_1 \cap R.$$

In other words, instead of conditioning on the full R, it is sufficient to condition only on the overlapping site(s) between U_1 and R, which are already available. This choice is consistent with the Bethe-like factorization and avoids marginal proliferation while remaining exact in the two-qubit case.

6.6.3 Background noise transitions

As we already said for the quantum depiction, between steps there is always a background noise transition accounting for environmental single qubit noise and crosstalk. This transition is linear and stochastic in its twirled form, so operates like a stochastic transition kernel on the ansatz state.

We can model it by the following procedure, both in belief tracking and simulation:

- For every qubit, apply a single qubit depolarizing transition element to update the marginals. Order doesn't matter.
- For every pair or neighboring qubits, apply a ZZ noise element to update the marginals. Order doesn't matter.

Since there are polynomial marginals and each update is polynomial, the whole procedure is polynomial. We now describe what happens to the marginals at each step.

6.6.4 Steps

At every step, we must first check that ansatz state are mapped to ansatz state. This will prove by induction the ansatz. Then we must define how update the marginals so the corresponding change of the belief is currently tracked.

Step 1: Recovery chains

A recovery Pauli chain applies a deterministic flip to the frames, independent of the input state. It is the *result* of the application that is deterministic once a chain has been chosen; this has nothing to do with the stochasticity of the choice of the chain itself.

Applying a recovery chain clearly maps ansatz states to ansatz states. Let the function

$$C : Q \;\to\; \{0,1\}^2$$

describe the chain, where $C(q)$ is the flip applied to qubit q.

Perfoming the associated transition means iterating through the lattice applying the corresponding transition element for qubit q is

$$\tau(e_q) \;=\; \delta(e_q, C(q)),$$

to update all the marginals as we saw.

Step 2: First Hadamard layer

The noisy Hadamard consists of a pure Hadamard channel followed by a single-qubit depolarizing channel.

We assume that the ansatz is verified after the previous step:

$$\rho(T,1) \;=\; \sum_{P\in\mathcal{P}_Q} B_{T,1}(P)\; P\left(\lambda \otimes |0\rangle\langle 0|^{\otimes A}\right) P.$$

We commit a slight abuse of notation by not explicitly accounting for the noise between $t = 1$ and $t = 2$, so as to keep the exposition more concise. Strictly speaking, we should distinguish between the pre-gate state and the state after the previous gate. However, since the noise channels map ansatz states to ansatz states, this distinction is unnecessary. We will use this simplification throughout the chapter.

The Hadamard channel applies single-qubit Hadamards on all X-ancillas:

$$H \;=\; \bigotimes_{a\in A_X} H_a.$$

Thus, the updated state is

$$\rho(T,2) \;=\; \sum_{P\in\mathcal{P}_Q} B_{T,1}(P)\; H\,P\left(\lambda \otimes |0\rangle\langle 0|^{\otimes A}\right) P\,H^\dagger.$$

Equivalently,

$$\rho(T,2) \;=\; \sum_{P\in\mathcal{P}_Q} B_{T,1}(P)\; \left(HPH^\dagger\right)\left(H\left(\lambda \otimes |0\rangle\langle 0|^{\otimes A}\right)H^\dagger\right)\left(HPH^\dagger\right).$$

The conjugation by H factorizes as

$$HPH^\dagger \;=\; \bigotimes_{d\in D} P_d \bigotimes_{a\in A_Z} P_a \bigotimes_{a\in A_X} \left(H_a P_a H_a^\dagger\right).$$

Since Hadamards are Clifford gates, the conjugation $H_a P_a H_a^\dagger$ maps any Pauli operator to another Pauli operator. Therefore, the ansatz is preserved.

Finally, to update the coefficients we apply, for each X-ancilla: - the Hadamard transition element kernel - the depolarizing noise transition element with depolarizing probability p_H.

Steps 3-6: CNOT propagation

CNOTs are applied in four turns corresponding to the four cardinal directions relative the ancillas, and map ansatz states to ansatz states exactly like the Hadamard gates. The pure CNOT channel is followed by two-qubit depolarizing noise.

Let $c(t) \in C$ denote the direction of the t-th turn, and consider the set of links $L(c(t), s)$ for both stabilizer types $s \in \{X, Z\}$.

After the t-th turn, the density operator evolves as

$$\rho_{T,t}^{\text{pre-depolarizing noise}} = \sum_P \left(\bigotimes_{l \in L(c(t),X)} \text{CNOT}_{\alpha(l)\to\delta(l)} \; B_{T,t}(P) \; \rho_{\lambda,t-1} \; P \; \text{CNOT}^\dagger_{\alpha(l)\to\delta(l)} \right) \otimes \left(\bigotimes_{l \in L(c(t),Z)} \text{CNOT}_{\delta(l)\to\alpha(l)} \; B_{T,t}(P) \; \rho_{\lambda,t-1} \; P \; \text{CNOT}^\dagger_{\delta(l)\to\alpha(l)} \right).$$

Here, for X-stabilizer CNOTs the ancilla is control and data is target, while for Z-stabilizer CNOTs the data is control and ancilla is target in agreement with the standard surface code architecture.

The customary abuse of notation has been performed: we ignore the noise channel in between the CNOT step and the previous step.

Since CNOTs are Clifford and unitary gates, by executing the same manipulations as in the Hadamard case we see that the ansatz is preserved. The marginals are updated iteratively by each CNOT rule-based transition element, according to the overalapping of their support with the CNOTs targets and controls.

Then the marginals are updated by iteratively performing for the same links affected by the CNOTs the transition corresponding to the two-qubit depolarizing noise element.

Step 7: Second Hadamard layer and stabilizer formation

The second round of Hadamard gates updates the state to

$$\rho(T,7) = \sum_{P \in \mathcal{P}_Q} B_{T,6}(P) \; P \, H \, \text{CNOTs} \, H \, (\lambda \otimes |0\rangle \langle 0|^{\otimes A}) \, H \, \text{CNOTs} \, H \, P,$$

At this point, we can exploit the surface code structure to return to a Pauli conjugation of the initial state, maintaining the ansatz while forming the stabilizers. Consider an ancilla qubit a and its neighboring data qubits $d \in N(a)$.

As we saw in section 2.3, for X-type stabilizers, the circuit

$$H_a \prod_{d \in N(a)} \text{CNOT}_{a\to d} H_a$$

is equivalent to a controlled-X operator on the ancilla qubit a, controlled by the product of X_d over all $d \in \mathcal{N}(a)$, i.e controlled by the star stabilizer $X(\dashv)$.

Since $\lambda \otimes |0\rangle \langle 0|^{\otimes A}$ is a code state, the star stabilizer eingevalue is $+1$, so the controlled-X acts as the identity.

For Z-type stabilizer, the circuit:

$$\prod_{d \in N(a)} \text{CNOT}_{d\to a}$$

is equivalent to a controlled-X operator on the ancilla qubit a, controlled by the product of Z_d over all $d \in N(a)$. The same considerations apply, bringing the state back to

$$\rho(T,7) = \sum_{P \in \mathcal{P}_Q} B_{T,7}(P)\, P\left(\lambda \otimes (|0\rangle^{\otimes A}\langle 0|^{\otimes A})\right) P.$$

The effect of the circuit was thus to propagate the errors affecting the data qubit to the ancilla as X flips, and we implicitly realized its whole mechanism by modeling error propagation.

Step 8: Measurement and reinitialization

This step is more articulated.

Measurement Applying the measurement channel (Eq. 6.1) produces a binary readout $s_a(T+1) \in \{0,1\} \forall a$. The probability of obtaining $s_a(T+1)$ is given by the Born rule.

If the system is conjugated by operator P of frame f, the probability of observing s is

$$p(s \mid P) \;=\; \prod_a (1-p_r)^{\delta_{f_{a,x},\, s_a}}\; p_r^{\,1-\delta_{f_{a,x}\, s_a}}, \tag{6.9}$$

where $f_{a,x}(P)$ denotes the deterministic outcome predicted by the Pauli frame P for stabilizer a, and r is the readout error probability. Thus, the overall probability of observing $s(T)$ is

$$p\big(s(T+1)\big) \;=\; \sum_P B(P)\, p\big(s(T+1) \mid P\big). \tag{6.10}$$

After obtaining $s(T+1)$, the system evolves under the corresponding measurement operators. For each ancilla qubit a, the single-qubit measurement operator (for the observed outcome $s_a(T)$) acts on a state P with frame f as

$$M_{s_a(T+1)}\, P_a\, |0\rangle\langle 0|\, P_a\, M^{\dagger}_{s_a(T)} \;\propto\; \left[(1-p_r)^{\delta_{s_a(T+1),\, f_{a,x}}}\; p_r^{\,1-\delta_{s_a(T+1),\, f_{a,x}(P)}}\right] P_a\, |0\rangle\langle 0|\, P_a, \tag{6.11}$$

where p_r is the uniform readout error rate.

Collecting the factors coming from the measurement operators and renormalizing, the belief undergoes this update:

$$B'(P) \;=\; \frac{p\big(s(T+1) \mid P\big)\, B(P)}{\sum_{P'} p\big(s(T+1) \mid P'\big)\, B(P')}. \tag{6.12}$$

Which is completely indistinguishable from applying a classical emission kernel. The measurement acts classically because we are measuring a combination of projectors on a mixture of eigenvector, so there is no wavefunction collapse.

Marginal update. How are the marginals updated to reflect the belief update? We need an update of the marginals based on the marginals only, such that the new marginals are the marginals of the new belief. Luckily, it's possible prove that, because of the the emission kernel is factorized on nodes, marginals can be updated using Bayes rule in the same way as the full belief.

Proof. Let $B(f)$ be the current joint belief over the lattice, and let the emission factors be node-wise, so that

$$\prod_i p(s_i \mid f_i)$$

factorizes over the nodes. Consider the updated marginal over a region R:

$$M'[R](f_R) = \sum_{f_{-R}} \frac{B(f)\prod_i p(s_i \mid f_i)}{\sum_f B(f)\prod_i p(s_i \mid f_i)}.$$

We split the product over nodes in R and nodes outside R:

$$\sum_{f_{-R}} B(f)\prod_i p(s_i \mid f_i) = \sum_{f_{-R}} B(f)\left(\prod_{i\in R} p(s_i \mid f_i)\right)\left(\prod_{j\notin R} p(s_j \mid f_j)\right).$$

Multiplying and dividing by the marginal $M[R](f_R) = \sum_{f_{-R}} B(f)$ gives

$$\sum_{f_{-R}} B(f)\prod_i p(s_i \mid f_i) = \prod_{i\in R} p(s_i \mid f_i)\, M[R](f_R) \sum_{f_{-R}} \frac{B(f)}{M[R](f_R)} \prod_{j\notin R} p(s_j \mid f_j).$$

Similarly, the denominator can be written as

$$\sum_{f} B(f)\prod_i p(s_i \mid f_i) = \sum_{f_R} M[R](f_R) \prod_{i\in R} p(s_i \mid f_i) \sum_{f_{-R}} \frac{B(f)}{M[R](f_R)} \prod_{j\notin R} p(s_j \mid f_j).$$

Observe that the term

$$\sum_{f_{-R}} \frac{B(f)}{M[R](f_R)} \prod_{j\notin R} p(s_j \mid f_j)$$

appears in both numerator and denominator, and therefore cancels out. This yields the updated marginal

$$M'[R](f_R) = \frac{M[R](f_R)\prod_{i\in R} p(s_i \mid f_i)}{\sum_{f_R} M[R](f_R)\prod_{i\in R} p(s_i \mid f_i)}$$

Thus, the updated marginal depends only on the previous marginal $M[R]$ and the node-wise emission factors, without requiring access to the full belief $B(f)$. □

A similar proof can be done for conditional marginals.

Ancilla reset. The ancilla reset requires iterating over all ancillas and updating all the marginals by a rule-based transition element

$$\tau_{\text{reset}}(e_q, e'_q) = \begin{cases} 1 & e'_q = 00, \\ 0 & \text{otherwise,} \end{cases}$$

followed by a bit-flip noise transition element.

Partial tracing and intercycle state. At this point we can obtain the data intercycle belief b_{T+1} by marginalizing away the ancillas in $B_{T+1,0}$. As usual, this is computationally unfeasible because we can't work with the full belief. We don't need the data intercycle belief because it is sufficient to work with the marginals of b_{T+1}, that we denote by m_{T+1}. We can obtain them directly from the marginals $M_{T+1,0}$ of the full belief, again by marginalization over ancillas. This marginalization reduces the number of relevant marginals:

- The 5-qubit marginals become 4-qubit single type marginals
- The 3-qubit marginals become 1-qubit single type marginals

- The 2-qubit marginals become 1-qubit 2-type marginals, making the 1-qubit single type redundant.

We call the m_{T+1} *decoding marginals.* There are thus two types of decoding marginals:

- data decoding marginals: 2-type marginals for a single data qubit, expressed by a 4-component vector because the marginal is a probability distribution on an input space representing the possible values of a couple of binary variables, a variable for each type. They are denoted as:
- stabilizer decoding marginals: single type marginals for the quadruplet of data qubits in the neighborhood of a stabilizer ancilla. Each is expressed by a 16-components vector, because the marginal is a probability distribution on an input space representing the possible values of a quadruplet of binary variables.

At decoding times, the belief updater send decoding marginals the policy for computing a correction. We can now introduce the rest of the framework.

7 The Decoding problem as a POMDP

In this chapter, we complete the mapping of quantum decoding to a POMDP by specifying the construction of the rest of its elements. As usual for POMDPs, as we saw in chapter 3, the decoding POMDP can be solved as belief MDP, but because of the exponential explosion of the state space we need this belief MDP to be referred to the decoding marginals, not the full beliefs. We will thus show how construct the decoding POMDP elements in terms of the decoding marginals too, so that the agent can compute its rewards and terminal conditions during training with information that it owns, i.e. the decoding marginals.

7.1 Actions

Let's now define the components of a recovery action and their encoding.

In our framework, a decoder's action is a triple:

$$(C, \mathcal{R}_X, \mathcal{R}_Z)$$

where:

$C \in \{0,1\}$ is the deliverability toggle,

$\mathcal{R}_X$ is the X-type homology class,

$\mathcal{R}_Z$ is the Z-type homolgy class.

And it represents a recovery chain and a signal for the user to wait or access the register.

We now clarify each part.

7.1.1 Deliverability Toggle

As discussed in Chapter 5, our goal is to optimize the fidelity at the time of user access, rather than instantaneous fidelity. This means that the decoder may perform explorative actions to gain information, during which the qubit may temporarily go even further from its original state.

Now, assume that the user does not require immediate access to the qubit whenever it queries for it and can tolerate delays if this improves the quality of information. If a user accesses the encoded qubit without control and the decoder is performing an explorative action, the user will receive lower-quality information than with a one-shot decoding.

To address this, we introduce a *deliverability toggle* $C \in \{0,1\}$, designed to distinguish between ready-to-use states and temporary states. Specifically, $C = 1$ signals that the user may access the register, while $C = 0$ indicates that the user must wait.

This way we can gain some control on user access time to improve fidelity at access.

7.1.2 Homology classes

Since every recovery chain in the same homology class has the same effect on the code, we can describe the action space in a non-redundant way by reducing it to the space of homology classes and picking always the same representative chain for each class. This way we achieve superior sampling efficiency, because the information about the effect of an action is not uselessly spread on updating equivalent but different chains.

To achieve this, it is necessary to find a method to encode the homology classes with a polynomial size representation.

7.1.3 Encoding the homology classes

We start from the simpler task of identifying homology classes of open simple chains, and then build from it the encoding of homology classes for general recovery chains.

Lattice unfolding

To represent the lattice, we consider the abstract definition of the surface code, with data qubits on edges and stabilizers on vertices. One physical ancilla qubit is chosen as the *root*, representing the origin in the unfolding. Two cuts, located at a distance $d/2$ from the root along orthogonal directions, define the right and upper edges of the unfolded lattice. These edges are assigned full boundaries in planar square representation, corresponding to stabilizers, connected by qubits (vertical edges on right side, horizontal in the upper side) while the corresponding lower and left edges are assigned empty boundaries, corresponding to qubits with no stabilizer on the outer end. The edges are representing the the interior of the lattice to empty boundaries represent qubit of the opposite side of the stabilizer.

Homology class of a simple chain

We first consider the case of a homogeneous cycle. We define an algorithm to compute the homology class of an homogeneous cycle, that will also be the basis for encoding homology classes. Given a cycle E, it either implements a stabilizer or a logical operator. To determine which, we exploit the commutation relations: logical operators satisfy among themselves the same commutation rules as the logical Pauli operators they represent. We fix a pair of logical chains corresponding to the cycle defining the borders, of the opposite type of E, for example:

$$\overline{X}_1, \overline{X}_2, \overline{Z}_1, \overline{Z}_2,$$

and compute the commutation relation of E with each reference chain using the scalar product formula of Eq. 2.9. The cycle implements the logical operation identified by anticommutation with the opposite logical operator on the same qubit. For example is $\overline{X}_1$ if it anticommutes wiht $\overline{Z}_1$ only. The scalar product determine that the anticommutation is defined by the parity of the border crossing and the type of the chain. An homogeneous chain with odd crossings with a border is a logical operator relative to qubit associated with the border. Thus the homology class of a homogeneous cycle can be represented by the pair of binary values

$$(l_1, l_2)$$

giving the parity of the number of crossings.

Now consider the case of a simple open chain. Let a_1, b_1 be the two stabilizers at its endpoints. We define a *canonical connection* between a_1 and b_1 as any simple chain joining them without crossing any reference border. All canonical connections between the same pair of stabilizers are in the same homology class.

Adding a stabilizer to the chain leaves the canonical connection in the same homology class, since any cycle crosses every border an even number of times and therefore does not change the border-crossing parity. In contrast, adding a logical operator changes the homology class by flipping the corresponding parity bit.

Thus, the homology class of a simple open chain is uniquely represented by the tuple

$$\big(a_1, b_1\big) \times (l_1, l_2),$$

where (l_1, l_2) are the same border-crossing parities defined above for cycles.

Any chain can be recast into a canonical connection between (s_1, s_2) and logic error corresponding to (l_1, l_2).

Homology class of general homogeneous chains

A general homogeneous recovery chain, modulo stabilizers, is a union of simple open chains and non-trivial cycles. We could represent the homology class using the homology class of all the simple chains, but this would not be minimal, because the logical errors corresponding to different chain can cancel with each other. Thus the homology class of a general homogeneous chain made of n simple open chains is defined by the endpoints of each simple chain and the total net logical errors, specified by

$$\big\{(a_1, b_1), \ldots, (a_n, b_n)\big\} \times (l_1, l_2)$$

were no chain extremities are overlapping, otherwise we would lose a simple chain.

Under these conditions, the set $\big\{(a_1, b_1), \ldots, (a_n, b_n)\big\}$ represents what in combinatorics is called a *matching* of the nodes in the lattice.

This means that the set of homology classes and thus actions is represented by the set of matchings times $(l_1, l_2) \in \{0,1\}^2$. The matchings grow exponentially, which means the action space also grows exponentially.

Managing this is possible and will be addressed in the next chapter.

7.1.4 Chain Selector

In the following we will need to translate an homology class into a recovey chain to execute an action. The chain selector algorithm translates the encoding of an homology class into an actual recovery chain. Every pair in the matching is connected by a canonical connection chosen by some fixed algorithmic policy. One example could be using Dijkstra's algorithm on the unfolded lattice with open boundary conditions, so border crossing are avoided. It then adds the logical errors indicated by the parity of the crossings.

7.2 Rewards

We now need to define how to design a proper reward system capable of capturing a user's subjective preferences. Here, we make some simplifying assumptions and leave to future work the development of a general procedure for translating user-defined parameters into a reward system.

7.2.1 Query model

Now that we have decided to explicitly model access to the qubit with the deliverability toggle, we have to define an *query model*. Deliverability matters only when the user requests the qubit. We model the requests from the user as independent and assume that, at each decoding time,

the qubit is queried with query probability p_a. From the perspective of decoding time T_1, the probability that the next access occurs exactly at $T_2 > T_1$ is

$$p_a(1-p_a)^{T_2-T_1-1}.$$

The decoder delivers the qubit or not based on the C.

The query model should in principle specify what happens in case of unavailability. We make a simplification: in this case the user just forgets about it and keep asking with the same rate. This avoids conditioning the query probability on past toggle states.

7.2.2 Performance metrics for decoding

In the most general case, the user specifies a series of parameters or rules, such as an acceptable logical overall error rate or how the usefulness of information depends on the waiting time, that define their own utility system by a *utility model*.

The overall subjective performance of decoding is evaluated as a function of the utility model and the query model, and its quantified by a *performance metric* . The rewards of the POMDP must be shaped so that a decoding agent, aiming at the maximal expected reward, automatically targets the best performance metric for the user.

Since this work is primarily aimed at introducing the concepts, we proceed with a simplified utility model:

1. We assign zero utility to qubits that are not requested or not delivered.
2. We assign a positive or negative score based on the closeness of the delivered state to the original state.
3. Delivered qubits with errors are penalized more than undelivered qubits.

We postpone to future work a full treatment on how to build more complex, eventually time-correlated, utility models.

For our simplified utility model, a naive score system would be the fidelity between the error-corrected state $\rho_D(T)$ at decoding time T and the original encoded pure state $|\lambda\rangle$:

$$F\big(|\lambda\rangle\langle\lambda|, \rho_D(T)\big) = \langle\lambda|\rho_D(T)|\lambda\rangle .$$

Since the error-corrected state can be written as

$$\rho_D(T) = \sum_P b_T(P)\, P\, |\lambda\rangle\langle\lambda|\, P,$$

we are led to evaluate terms of the form

$$\langle\lambda| P |\lambda\rangle .$$

These terms have the following properties:

- If P is not a cycle, the scalar product is 0.
- If P is a stabilizer, it acts as the identity on $|\lambda\rangle$, so the term equals 1.
- If P is a logical operator, two cases arise:
  1. If $|\lambda\rangle$ is an eigenvector of P, the term equals 1.
  2. Otherwise, it equals 0.

This analysis shows that raw fidelity is not an ideal score: it pushes the decoder to complete an error chain so that it becomes either a stabilizer or a logical error for which $|\lambda\rangle$ is an eigenvector, which is undesirable. This introduces complications for training, since the reward would not unambiguously encourage correct decoding.

A better choice is to consider the expected fidelity relative to a uniform distribution over logical states. In this case, the expected fidelity reduces to the probability that the current error chain belongs to the stabilizer group:

$$\mathbb{E}_\lambda\big[F(b_T)\big] = \sum_P b_T(P)\, \mathbb{I}[P \in \mathcal{S}]\,, \tag{7.1}$$

where $\mathbb{I}[\cdot]$ is the indicator function.

This corresponds to the probability that the error acting on the system is in the stabilizer group, rather than the probability of returning to the specific original logical state.

From the expected fidelity we may construct the utility function. Considering for the fidelity F it holds $0 \leq F \leq 1$, we define:

$$U(b_T) = 2C\left(\mathbb{E}_\lambda\big[F(b_T)\big] - p\right)$$

where p is a penalty term such that $p > \frac{1}{2}$, so that utility gets negative if a qubit with the same fidelity to the original as a random one is delivered, realizing the third requirement of the utility model.

The performance metric M that the decoder should maximize is the *expected utility at the time the qubit is accessed*, that is, the utility function averaged over request times:

$$M = \sum_{T=0}^{\infty} p_a(1-p_a)^T\, U(b_T). \tag{7.2}$$

This is exactly the form of expected discounted return in a POMDP, with $U(b_T)$ as expected reward and reward function:

$$R(P) = C\, \mathbb{I}[P \in \mathcal{S}] \quad \text{and} \quad \gamma = 1 - p_a.$$

The optimal policy of a POMDP with this reward system is thus the optimal decoder.

7.2.3 Monte Carlo approximation of the expected reward

As usual, the computation of the expected reward in Eq. 7.1 is infeasible due to the exponential size of the Pauli basis. Moreover, we need to rewrite this reward as a function of decoding marginals for the decoding marginals belief MDP. We therefore make two approximations.

First, we employ a Monte Carlo approximation: we draw a fixed number N of Pauli errors $P^{(1)}, \ldots, P^{(N)}$ from the belief distribution and average the value of the indicator over these samples. Formally, the Monte Carlo estimate of the expected reward is

$$\hat{r} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\Big[P^{(i)} \in \mathcal{S}\Big]\,.$$

To implement this, we require a polynomial-time procedure to sample from b, which we only know by the marginals m. We employ a generalization of the Bethe factorization that we developed for this purpose.

Generalized Bethe factorization. Our sampling procedure works as follows:

- Declare a data qubit q as the root.
- Sample the X and Z configuration for q from its single-qubit marginal $m[q](\cdot)$.
- For each ancilla a in the four-neighbourhood $\mathcal{N}(q)$, condition its marginal $m[\mathcal{N}(a)](\cdot)$ on the already extracted variables of q, and sample the remaining variables in $\mathcal{N}(a) \setminus \{q\}$.
- For any qubit where only one of the two variables (X or Z) has been extracted, sample the other variable from its single-qubit marginal $m[q](\cdot)$ conditioned on the already extracted variable.
- Treat these qubits as new roots and repeat the process for their surrounding, not-yet-covered qubits.

The procedure is iterated until the entire lattice has been covered.

Checking if a chain is a stabilizer

Determine if a chain is a stabilizer can be done easily. It is if both the the homogeneous components chain are. To check this:

- iterating on the stabilizer to check the parity of the number of Pauli flips in the neighborhood. If all even is a cycle and go to next step, else return false.
- Checking the parity of the intersection with each logical error chain, if they are all even return true, else false.

7.3 Terminal conditions

The underlying hidden Markov model of the decoding POMDP does not admit any terminal states, but we need terminal beliefs on the belief MDP to apply policy gradient algorithms. Therefore, we heuristically define termination conditions.

Intuitively, these termination conditions should relate to our ability to understand the system's state, so they should be based on entropy. A first naive approach is to use the entropy of the belief b, which directly quantifies our ignorance about the system's state. However, this does not work because the only information relevant for decoding is the homology class of the error chain, not the exact chain itself. Ignorance within the same homology class does not affect action selection.

Instead, we consider the entropy over the distribution of homology classes induced by the belief. Let H be a random variable denoting homology classes, and let $b^H(H)$ be the marginal distribution of the belief over homology classes. The entropy is defined as

$$S(b^H) = -\sum_H b^H(H) \log b^H(H).$$

As usual, this sum requires an exponential number of terms, computing the marginalization b^H itself is intractable and anyway we track and solve the belief MDP by marginals, not full beliefs. We therefore resort to a Monte Carlo approximation. By recycling the samples of error chains P drawn from the marginal beliefs in the reward computation, we can compute an empirical marginal distribution over homology classes. Then, we estimate the entropy of this empirical distribution. Since the number of homology classes with nonzero empirical probability is at most the number of

samples, and terms corresponding to zero occurrences do not contribute, this approximation is efficient.

We can then set an entropy threshold and terminate episodes when the entropy exceeds this threshold. As a fallback, we also use a safety timeout to stop episodes that run for too long.

8 Solution of the decoding POMDP and definition of the workflows

Now that we have defined all the elements of the POMDP with polynomial size and runtime, we can focus on solving the error-correction problem by finding an optimal policy and test the proposed solution.

8.1 Using reinforcement learning as an approximation

In principle, since we have access to the kernels of the POMDP, a direct solution via dynamic programming is theoretically possible, and decoding could be framed as a control problem. However, directly solving a POMDP is computationally feasible only for very small problems (on the order of tens of states and actions). For realistic surface-code decoding problems, the state and action spaces are far too large for exact methods.

The only viable way to exploit the known kernel is to adopt a *reinforcement learning on imagination* approach: training occurs entirely within a simulated environment generated by the internal error model of the decoder, without requiring real rollouts from the physical device.

Unlike in [2], we define the action space as the set of homology classes of recovery actions, rather than individual qubit flips. This choice prevents us from applying Q-learning directly, since we cannot efficiently compute the arg max over the exponentially large matching space. Instead, we rely on a policy gradient method to train the policy, which is parameterized by a neural network.

8.1.1 Decoding workflow

The decoding workflow specifies how the decoder operates in a production setting. It is important to emphasize that the policy solves the belief marginals MDP, but it is not enough to constitute the decoder itself. We define the *decoder* as the tuple:

$$\text{Decoder} = (\text{belief marginals updater},\ \text{policy},\ \text{chain selector}),$$

where:

- **Belief marginals updater:** maps syndrome measurements to belief marginals states. At decoding times, computes the decoding marginals and send them to the policy:

$$\text{syndrome} \longrightarrow \text{decoding marginals}.$$

- **Policy:** maps the decoding marginals to actions,

$$\text{belief marginals} \longrightarrow \text{action}.$$

- **Chain selector:** maps the homology classes in the actions to specific recovery chains,

$$\text{action} \longrightarrow \text{recovery chain.}$$

These three element are sequentially called during the *decoding loop*.

- Iteratively update the full belief marginals using the latest syndrome measurements.
- At decoding times, marginalize them to get the decoding marginals.
- Send the decoding marginals to the policy module and compute an action.
- Pass the resulting action to the chain selector, extract a recovery chain.
- Apply the corresponding recovery chains.

A variable of the decoding workflow is the *decay time*, determining after how much time a user assume the qubit is lost after his request.

Now we can specify the structure of the workflow

- **Initialization:** Initialize the full belief marginals state.
- **Protection of the logical state**: While no qubit delivery is requested:
- Decoding loop
- **Qubit delivery request:** When a qubit delivery is requested:
  - At the first intercycle time, check the *deliverability toggle*.
  - If the toggle is positive, deliver the qubit and end the process.
  - If the toggle is negative:
    - Start a *decay counter*.
    - While the toggle remains off and decay counter $<$ decay time:
      - Decoding loop.
    - If the toggle becomes positive, deliver the qubit.
    - If decay time is reached without toggle activation, return a *lost qubit*.

8.2 Policy

The actual implementation of a policy requires two submodules:

- the **score generator**, which is a neural network used to compute the probabilities that each stabilizer is matched with any other stabilizer
- the **action extractor**, which samples matchings based on the scores.

This design was necessary to map marginals to matchings, because there is no way to represent a probability distribution over matchings by a neural network output layer.

The architecture of the score generator neural network is not trivial. The decoding marginals are not a classical tensor shaped input, because stabilizer and data marginals have different number of components. The classical neural network architecture require a tensor shaped input. Another problem to consider is that in designing a neural network decoder is useful to take into account the symmetries of the problem. Symmetries help impose equivariance costraints which reduce the number of independent parameters. We digress into these two topics before describing the architecture of the score generator.

8.2.1 Symmetries of the toroidal surface code and equivariance

In its flattened 2D representation with periodic boundary conditions, the toric code exhibits several discrete symmetries. In particular, the code space is invariant under:

- Horizontal and vertical translations (due to the toroidal topology),
- Horizontal and vertical reflections,
- Rotations by 90°, under which logical operators are permuted (e.g., $\bar{X}_1 \leftrightarrow \bar{X}_2$, and similarly for $\bar{Z}_1$ and $\bar{Z}_2$).

These symmetries are not limited to the code itself, but also extend to our stochastic error model.

To fully exploit these symmetries in the design of the decoder, it is important to consider the equivariance properties of the entire Markov Decision Process (MDP) involved in the decoding task. This includes the reward function, transition dynamics, belief updates, and terminal conditions. A complete symmetry-aware analysis of the MDP is beyond the scope of this work and is left for future investigation. At this level we just assume it can be done.

8.2.2 Embedding the marginals in a factor graph

We could in principle just flatten the decoding marginals into a single vector. This would result in a tensor-shaped input, fit to be fed to a classical neural network. The problem with this approach is that we would lose the inductive bias about the geometrical structure of the lattice and the decoder would have to learn it from scratch. To exploit this information it is thus necessary to fit the decoding marginals into a data structure that accounts for their relations, i.e. that tracks the adjacency relations between the data and stabilizers marginals. A proper structure for this task is a *factor graph*. Factor graphs are well-studied structures, usually used in statistical inference. A factor graph is composed of a bipartite graph and a set of *factors*. The bipartite graph is denoted by:

$$\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E}),$$

where

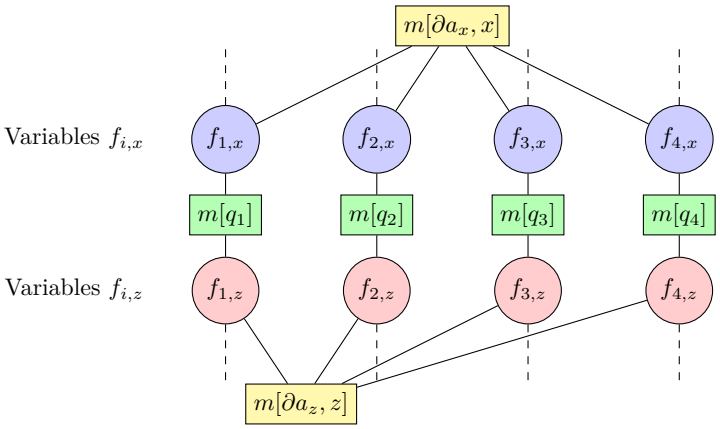
- $\mathcal{N}$ is the set of *variable nodes*,
- $\mathcal{W}$ is the set of *factor nodes*,
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{N}$ is the set of edges connecting variable nodes to factor nodes.

Each variable node $n \in \mathcal{N}$ is associated with a variable v_n. Each factor $w \in \mathcal{W}$ is associated with a function m_w, which takes as input the variables corresponding to the variable nodes linked to w.

In our case, we define two variable nodes for each qubit, one for each type, denoting them by the tuple qubit-type (q, s) and associate with them a binary variable f describing the corresponding component of the Pauli frame. Then, we have two types of factor nodes:

- One linking variables related to the same qubit and different types, representing single-qubit 2-type marginals,
- Another linking variables corresponding to the four qubits of a stabilizer, representing 4-qubit stabilizer marginals.

The factor graph can be represented as:



We can now store the marginals as factors. Notice that we are not defining a probability distribution as the product of the factors, as is usually done with factor graphs. We are simply using them as a data structure.

Now, there is a class of neural networks called *factor graph neural networks*, introduced in [18]. These are neural networks that take as input a factor graph and can be used to learn functions of factor graphs. This constitutes a perfect match for our score generator, allowing to fully leverage on the topology of the data.

8.2.3 Score generator

The score generator neural network in factor graph neural network taking as input the current decoding marginals and producing a set of outputs in corresponding *output heads*:

Output heads:

- **Weight graph:** A fully connected undirected weighted graph on the stabilizers augmented with a dummy "no match" node.
- **Logical vector head:** A 4-dimensional vector representing the probability of each logical error class.
- **Deliverability toggle**: A scalar representing the probability of deliverability.

The weight graph encodes the likelihood of pairing any two stabilzer nodes during chain selection, while the logical vector provides a soft prediction over logical error to apply to complete the encoding of the homology class.

To go from the input to the output head we need multiple layers. These layers must be designed in such a way to exploit the equivariance constraints to reduce the number of parameters to learn. Doing this is a research topic on its own that cannot be fit in the scope of this thesis. We thus postpone the complete design of neural network architecture to future work, since this work is about the definition of an abstract framework.

To convert an output in an action we must be able to extract an action from this data.

8.2.4 Extracting an action

The first thing that we need to do is extracting a matching from the weight graph. We can do it by the *matching extractor* algorithm:

$$G = (V, E, W), \quad V = S \cup \{\text{dummy}\}, \quad E = V \times V,$$

be the weighted graph, where S is the set of stabilizer nodes, the dummy vertex allows a stabilizer to remain unmatched, and $W : E \to \mathbb{R}_{\geq 0}$ assigns non-negative weights to edges in a symmetric way.

Available nodes: Define the set of available nodes

$$A \subseteq S$$

which initially contains all stabilizer nodes: $A \leftarrow S$.

Extraction procedure: Repeat the following steps until A is empty:

1. **Select a node.** Choose the node $s \in A$ with the maximum total outgoing weight (outdegree):

$$s^* = \arg\max_{s \in A} \sum_{v \in V} W(s, v).$$

   If there is a tie, the sample is discarded.

2. **Select a match.** Choose a node $v \in A \cup \{\text{dummy}\}$ with probability equal to the edge weight normalized by s^*'s outdegree:

$$p(v \mid s^*) = \frac{W(s^*, v)}{\sum_{u \in A \cup \{\text{dummy}\}} W(s^*, u)}.$$

3. **Update available nodes.**

   - If $v = \text{dummy}$, remove only s^* from A.
   - If $v \in A$, remove both s^* and v from A.

The result is a partial matching where each stabilizer is matched at most once, or left unmatched via the dummy node.

After this we need to extract the logical chains to complete the homology class and a toggle. We can do it by simply extracting a binary value with the probability specified by their nodes.

8.2.5 Likelihood of an action

This procedure has been designed so that the likelihood of a given action, given the belief marginals, can be computed in polynomial time. We need the gradient of this likelihood for training. Given the marginals, we can obtain the output the the score generator by plugging them into the neural network, so we have the weights as a function of the parameters of the neural network. For computing the likelihood of a matching we use the following *likelihood computer* algorithm:

1. Initialize the set of available nodes $A \leftarrow S$ and set the likelihood $L \leftarrow 1$.

2. Repeat until A is empty:

   (a) Select the stabilizer with maximum outdegree:

$$s^* = \arg\max_{s \in A} \sum_{v \in V} W(s, v).$$

   If there is a tie, the sample is discarded.

   (b) Let v^* be the node matched to s^* in the given action.

(c) Compute the conditional probability:

$$p(v^* \mid s^*) = \frac{W(s^*, v^*)}{\sum_{u \in A \cup \{\text{dummy}\}} W(s^*, u)}.$$

(d) Update the likelihood: $L \leftarrow L \cdot p(v^* \mid s^*)$.

(e) Update available nodes:

- If $v^* = \text{dummy}$, remove s^* from A.
- If $v^* \in A$, remove both s^* and v^* from A.

The resulting L is the likelihood of the observed matching under the current weights W. This gets multiplied by the likelihood of the extracted toggle and logical chains to complete the likelihood.

8.2.6 Gradient of the likelihood

Assume that the parameters W are far from points where the arg max selection changes. Then the stabilizer extraction is locally constant, and the gradient of the likelihood can be computed as follows:

1. Initialize the set of available nodes $A \leftarrow S$ and the gradient accumulator $\nabla L \leftarrow 0$.
2. Repeat until A is empty:

(a) Select the stabilizer with maximum outdegree:

$$s^* = \arg\max_{s \in A} \sum_{v \in V} W(s, v).$$

(Tie points are excluded.)

(b) Let v^* be the node matched to s^* in the given action.

(c) Compute the conditional probability:

$$p(v^* \mid s^*) = \frac{W(s^*, v^*)}{\sum_{u \in A \cup \{\text{dummy}\}} W(s^*, u)}.$$

(d) Compute the gradient of the log-likelihood for this step:

$$\nabla \log p(v^* \mid s^*) = \frac{\partial \log W(s^*, v^*)}{\partial W} - \sum_{u \in A \cup \{\text{dummy}\}} \frac{W(s^*, u)}{\sum_{u'} W(s^*, u')} \frac{\partial W(s^*, u)}{\partial W}.$$

This simplifies to:

$$\frac{\mathbf{1}_{(s^*, v^*)}}{W(s^*, v^*)} - \frac{\mathbf{1}_{(s^*, \cdot)}}{\sum_{u \in A \cup \{\text{dummy}\}} W(s^*, u)},$$

where $\mathbf{1}_{(s^*, v)}$ is an indicator for the corresponding weight.

(e) Accumulate the gradient:

$$\nabla L \leftarrow \nabla L + \nabla \log p(v^* \mid s^*).$$

(f) Update available nodes:

- If $v^* = \text{dummy}$, remove s^* from A.
- If $v^* \in A$, remove both s^* and v^* from A.

The final ∇L is the gradient of the log-likelihood of the observed matching with respect to the weights W. This can then be combined with the gradients of the toggle and logical chains to get the full policy gradient.

Comment of the singularities

In principle the argmax operation involved in likelihood extraction is not derivable. In devising an extraction mechanism, we faced a design crossroad: if we made the choice of the stabilizer stochastic, then the likelihood is derivable but computing the likelihood take exponential time, because it involves summing on all possible order of extraction of the stabilizers. Making the order of the extraction deterministic solves the issue but at the cost of perfect derivability. In practice the set of point where maximum is a tie and the operation is not derivable has zero measure, so we can still use this approach and treat the argmax as a identity function relative to the parameters, so we can slide the derivative inside it. Anyway if because of some numerical setting we hit the frontier, we can just raise an exception and discard the datum.

8.3 Training

We designed the decoder so that the training procedure does not require access to environment-generated rollouts. Instead, training occurs entirely *in imagination*, using the decoder's internal error model.

This approach allows training to proceed even in experimental settings, once the error model has been obtained (e.g., from external sources or from tomography). Since the training relies only on the internal representation of the environment maintained by the decoder - which is fully stochastic - it can be performed entirely through classical stochastic simulations, and it must be done so even when direct emulation of the physical system is available, because the belief update mechanism is based on a stochastic approximation. Simulation is performed by the *training environment simulator* module of the decoder.

For simulating the environment we suggest the Stim package, presented in [19]. Training is carried out via two methods of the decoder's policy module, invoked during both initialization workflows and update workflows.

8.3.1 Initialization workflow

At startup, quantum hardware usually perform a *calibration* routine, during which the parameters of the error model are set. They are usually inferred by some form of approximated tomography and some pre-existing information. The initialization workflow is launched after calibration. It consists of all the operations need to train a policy from scratch.

- The decoder initializes its stochastic approximation of the error model.
- A pre-training phase follows:
  - The policy is trained to imitate a minimum-weight perfect matching decoder via imitation learning, by the *imitate* method.
  - The baseline network is trained to approximate the value function of the MWPM decoder by the *baseline computer* method.
- Finally, the *improve method* is called, in which the decoder improves the policy neural network using the REINFORCE algorithm.

8.3.2 Update workflow

Because the error parameter drift during the hardware's uptime, periodic *recalibration* is performed to update the noise parameter. Whenever this is done the update workflow is launched to adapt the decoder to new error model.

- The decoder updates its internal stochastic approximation of the error model based on the new information

- The improve method is called to update the neural network part of the decoder to the eventual shifts in parameters.

In principle one could point out that the baseline needs to be updated before improvement to the value function of the current policy, but we don't expect dramatic difference with MWPM we keep the previous baseline.

Let's see these methods in detail.

8.3.3 Imitation method

The imitation method initializes the policy network by mimicking the behavior of a classical MWPM decoder. This warm start provides a high-quality prior policy, enabling faster convergence during reinforcement learning. Without pretraining, learning a good policy from random weights would be infeasible.

MWPM serves as a natural benchmark for our approach. We train the score generator to produce weight matrices that induce MWPM-like behavior. This is easy because our policy outputs homology classes as actions. The imitation learning procedure works in a standard pattern for every syndrome based decoder.

Imitation learning procedure:

1. Generate syndrome-recovery chain samples using the classical decoder.

2. Convert each syndrome into a belief state using the belief updater.

3. Translate each recovery chain into an action by computing its homology class and setting the corresponding toggle to 1.

4. Compute the likelihood π of each action-belief pair according to the POMDP based policy

5. Optimize the entropy loss described in Chapter 3 to train the score generator.

8.3.4 Baseline computer

The baseline function $b(s)$ is crucial to reduce variance in the REINFORCE policy gradient estimator without introducing bias, provided it depends only on the state s.

We model the baseline as the value function of MWPM. It is implemented by a graph neural network taking the marginal belief as input, with the same input layers seen and outputting a scalar. The training is performed by least square regression on (marginal belief,value) pairs. The value to associate to each marginal belief is computed by Monte Carlo averaging of the total return encountered on a rollout by using the pretrain network as policy in the decoder.

8.3.5 Improve method

Apply the REINFORCE method, like seen in chapter 3, using the likelihood computer algorithm and extracting the gradient as previously shown.

8.4 Testing workflow

The final part of the framework is the testing methodology, applied after training. Because the physical structure of the toric code does not allow encoding a single logical qubit into a single physical qubit, each experiment must be performed on a *pair* of qubits. Testing can be performed either on simulated or emulated qubits by resorting to the *environment backend*.

8.4.1 Environment back-end

The environment back-end is used to study the effect of the decoder on a real system. It is used to compute the evolution of a register under the recovery operation, its syndromes and actual hidden state. It can be simulated, emulated or experimental. We postpone the study of a quantum tomography based testing pipeline on experimental backends to future works. In case of simulation or emulation we need to define a *backend error model* which regulates the actual hidden dynamics and can be different that the error model of the decoder, which is an internal representation.

Testing on simulated backend requires to perform the same stochastic approximation as in training and can be done again by the Stim package. Testing on emulated qubits is not scalable but allows to avoid stochastic approximations. In particular, it allows for keeping the full unitary form of the crosstalk. In any case, the decoder will update its own representation of the state of qubits, which is stochastic, and apply recovery chains on the simulated/emulated encoded qubits, which are visible to the testing back-end but not to the decoder. The test on emulated qubits may be useful to study the error induced by the stochastic approximations.

8.4.2 Crossover experiment

Defining a crossover experiment pipeline is challenging. With faulty syndromes, determining when a logical error has occurred is unclear, because we cannot know with certainty when the qubits are in the code space. Furthermore, repeating the experiment for multiple code distances requires expensive retraining. For these reasons, we restrict our evaluation to break-even experiments.

8.4.3 Break-even experiment

A break-even experiment can be performed without relying on syndromes. For break-even analysis, we benchmark the encoded pair against an unencoded pair of physical qubits arranged in a 2×1 lattice with open boundary conditions, subject to the same background noise as the error-corrected pair.

We define the *lifetime* for both the error-corrected pair and the unencoded pair as the Monte Carlo average time before *death* (see below) of the simulated qubits, which we can access directly. We test lifetime against changing background noise parameters. The noise parameters affecting the surface code cycle must be kept as parameters, because the raw qubits are not affected by them. In principle we have two background rates, one for environment noise and one for crosstalk. To keep things simple we can keep on as parameter and find the break-even point relative to the other, by starting from high noise values and lowering down until error-correction doesn't offer a net lifetime benefit. We end by defining the death events:

Death of an error-corrected pair

The error-corrected pair dies when:

- the decoder incorrectly switches on the deliverability toggle while the pair is in an error state.

- the decay timeout has elapsed since deliverable state.

Death of an unencoded pair

For an unencoded pair, the death time is defined as the first time at which the scalar product with the initial state becomes zero.

9 Conclusions

We defined the framework in abstract form without providing a full implementation, as the effort required to formalize the concepts was already substantial for the scope of this thesis. We designed a modular framework that allows variation of every component. The idea was to build a factory, not just a product. The work is far from finished! There are indeed several steps to take to develop this project to its full potential.

9.1 Next Steps

- Develop an actual implementation as a proof of concept. Our prototypes will be gradually uploaded to the this repository.
- Test against stochastic and coherent noise.
- Clarify how to exploit symmetries to impose equivariance and invariance constraints for reinforcement learning decoders.
- Introduce these constraints in the score generator factor graph neural network.
- Choose an appropriate polynomial-time approximation of process tomography for the update workflow.
- Generalize the performance metrics to a more general query model.
- Generalize the performance metrics to more general utility functions.
- Adapt the reward system to these more general performance metrics.
- Extend the whole framework to different codes, starting from 2D planar codes, then other surface codes (like 3D codes), eventually pushing toward a general framework for multiple codes.

9.2 Final words

We hope that the idea of treating decoding as a POMDP will establish a new line of research and attract interest from other researchers. We hope that this line of research will be able to develop a general theory of POMDP-based decoding, so that POMDP-based decoder can emerge like a new industrial standard. Fault-tolerant quantum computers might represent a historic revolution, and we are proud to have tried to contribute to their progress.

Bibliography

[1] Sergey B Bravyi and Alexei Yu Kitaev. "Quantum codes on a lattice with boundary". In: *arXiv preprint quant-ph/9811052* (1998).

[2] Ryan Sweke, Friederike Anna Wilde, and Jens Eisert. "Reinforcement Learning Decoders for Fault-Tolerant Quantum Computation". In: *npj Quantum Information* 7.1 (2021), p. 14. DOI: 10.1038/s41534-020-00318-5.

[3] Chuang Nielsen. *Quantum Computation and Quantum Information*. 7th. The Edinburgh Building, Cambridge CB2 8RU, UK: Cambridge University Press, 2010.

[4] Daniel Gottesman. "Stabilizer codes and quantum error correction". In: *arXiv preprint quant-ph/9705052* (1997).

[5] Keisuke Fujii. *Quantum Computation with Topological Codes: from qubit to topological fault-tolerance*. 2015. arXiv: 1504.01444 [quant-ph]. URL: https://arxiv.org/abs/1504.01444.

[6] Sergey Bravyi et al. "Correcting coherent errors with surface codes". In: *npj Quantum Information* 4.1 (Oct. 2018). ISSN: 2056-6387. DOI: 10.1038/s41534-018-0106-y. URL: http://dx.doi.org/10.1038/s41534-018-0106-y.

[7] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Physical Review A* 86.3 (Sept. 2012). ISSN: 1094-1622. DOI: 10.1103/physreva.86.032324. URL: http://dx.doi.org/10.1103/PhysRevA.86.032324.

[8] Anthony Ryan O'Rourke and Simon Devitt. *Compare the Pair: Rotated vs. Unrotated Surface Codes at Equal Logical Error Rates*. 2024. arXiv: 2409.14765 [quant-ph]. URL: https://arxiv.org/abs/2409.14765.

[9] Joydip Ghosh, Austin G. Fowler, and Michael R. Geller. "Surface code with decoherence: An analysis of three superconducting architectures". In: *Physical Review A* 86.6 (Dec. 2012). ISSN: 1094-1622. DOI: 10.1103/physreva.86.062318. URL: http://dx.doi.org/10.1103/PhysRevA.86.062318.

[10] Cupjin Huang et al. *Alibaba Cloud Quantum Development Platform: Surface Code Simulations with Crosstalk*. 2020. arXiv: 2002.08918 [quant-ph]. URL: https://arxiv.org/abs/2002.08918.

[11] Kevin Murphy. *Reinforcement Learning: A Comprehensive Overview*. 2025. arXiv: 2412.05265 [cs.AI]. URL: https://arxiv.org/abs/2412.05265.

[12] L.R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[13] Antonio deMarti iOlius et al. "Decoding algorithms for surface codes". In: *Quantum* 8 (Oct. 2024), p. 1498. ISSN: 2521-327X. DOI: 10.22331/q-2024-10-10-1498. URL: http://dx.doi.org/10.22331/q-2024-10-10-1498.

[14] Jack Edmonds. “Paths, trees, and flowers”. In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467.

[15] Eric Dennis et al. “Topological quantum memory”. In: *Journal of Mathematical Physics* 43.9 (2002), pp. 4452–4505. DOI: 10.1063/1.1499754.

[16] Nissim Ofek et al. *Demonstrating Quantum Error Correction that Extends the Lifetime of Quantum Information*. 2016. arXiv: 1602.04768 [quant-ph]. URL: https://arxiv.org/abs/1602.04768.

[17] Acharya et al. “Suppressing quantum errors by scaling a surface code logical qubit”. In: *Nature* 614.7949 (Feb. 2023), pp. 676–681. ISSN: 1476-4687. DOI: 10.1038/s41586-022-05434-1. URL: http://dx.doi.org/10.1038/s41586-022-05434-1.

[18] Zhen Zhang et al. *Factor Graph Neural Networks*. 2023. arXiv: 2308.00887 [cs.LG]. URL: https://arxiv.org/abs/2308.00887.

[19] Craig Gidney. “Stim: a fast stabilizer circuit simulator”. In: *Quantum* 5 (July 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497. URL: http://dx.doi.org/10.22331/q-2021-07-06-497.

Acknowledgements

The first big thanks go to the Links Foundation for the opportunity to work on such an interesting project. Without them, I would never have considered exploring error correction, nor appreciated its importance.

I want to thank my supervisor and mentor, Giacomo Vitali, for his constant guidance and support throughout this work.

I am also grateful to my colleague Vincenzo Pianese for his enthusiasm and genuine interest in my work, which has been a great source of motivation.

I want to thank my internal supervisor, Davide Girolami, for his patience and for introducing me to the fascinating world of quantum computing.

I am grateful to the organizers of my master's program. This thesis would have been impossible without the interdisciplinary education that prepared me to integrate multiple fields in this work.

Finally, I want to thank my family for their support during these challenging years.