



# Politecnico di Torino

Computer Engineering

A.a. 2025/2026

Graduation Session July 2026

## Reinforcement learning models for decoding the surface quantum error correction code

Supervisors:

Bartolomeo Montrucchio  
Giacomo Vitali

Candidate:

Lorenzo Stefanoni

# Summary

Quantum computing is expected to transform computational methods by enabling the solution of extremely challenging problems in areas such as cryptography, materials science, and the simulation of complex physical systems. Nevertheless, the development of large-scale, fault-tolerant quantum computers is fundamentally limited by the high sensitivity of physical qubits to environmental disturbances and control errors. Among existing Quantum Error Correction (QEC) schemes, the Surface code is widely regarded as one of the leading approaches due to its high fault-tolerance threshold and its reliance on local qubit interactions. A key obstacle in using the Surface code is the decoding task: given a sequence of syndrome measurements, the decoder must identify the most probable error pattern in order to determine the correct recovery operations.

The conventional solution to this task is the Minimum Weight Perfect Matching (MWPM) algorithm, which builds a graph representing error probabilities and searches for the lowest-weight matching. Although computationally efficient, MWPM is based on simplified noise assumptions that treat errors  $X$  and  $Z$  as independent and identically distributed (i.i.d.). Practical quantum devices, however, exhibit significantly more complex noise processes, including correlated  $Y$ -errors, which introduce long-range dependencies that standard graph-based methods struggle to represent. In addition, MWPM does not explicitly account for code degeneracy, where several distinct error configurations generate the same syndrome, and therefore cannot fully exploit the redundancy available within the quantum code, and cannot manage readout errors, one of the hardest error types to detect.

This thesis addresses these limitations by expressing the decoding task within the framework of Partially Observable Markov Decision Processes (POMDPs). By treating decoding as a sequential decision-making problem, we make it possible to apply Reinforcement Learning (RL) techniques to learn decoding policies directly from syndrome measurements. This perspective replaces static graph matching with an adaptive, policy-driven inference process that can capture complex, non-Markovian noise correlations.

The proposed solution is based on a Heterogeneous Graph Neural Network

(HeteroGNN) architecture. Unlike homogeneous GNNs, the model explicitly distinguishes among data qubits, stabilizer checks and belief-factor nodes, allowing it to learn the directional propagation of errors across the surface code lattice. We further improve the model by incorporating Bayesian local marginals, providing a strong inductive bias regarding the probability distribution of errors. By integrating these marginals into the neural message-passing process, the architecture effectively combines global structural information with local physical constraints, leading to a decoding policy that is both more reliable and more scalable.

A substantial part of this research focuses on understanding the dynamics of the learning process. Our experiments reveal a phenomenon that we refer to as the minimum-weight attractor. In many sequential RL training settings, the learned policy naturally converges toward something similar to (and never better than) the behavior of the traditional MWPM decoder. We show that this occurs because minimum-weight syndrome nulling is locally optimal at every step: any sequential policy trained on this objective, whether initialized by imitation or trained from scratch, is pulled toward the behavior of MWPM and cannot escape it through local improvements. We argue that this attractor prevents the model from discovering alternative decoding strategies that take advantage of code degeneracy.

To address this issue, we introduce a two-phase training pipeline. During the first phase, DAgger is used to initialize the policy through imitation learning, providing a stable starting point. During the second phase, the policy is optimized through unconstrained reinforcement learning, encouraging exploration beyond the limitations of classical decoding heuristics.

Our evaluation under multiple noise regimes shows that the sequential RL-based decoders, including those trained with the two-phase pipeline, never outperform MWPM: they converge to its level of performance and stop there, as the attractor predicts. The margin is instead recovered by the supervised joint-class decoder, which exploits the  $Y$ -error correlations discarded by MWPM, reducing the logical error rate by 35% (0.0104 vs 0.0159 at  $d = 5$ ) and matching maximum-likelihood decoding; under noisy readout measurements, a lookahead variant roughly halves the error rate of the memoryless MWPM baseline in closed loop. These findings draw an experimentally verified map of where learning genuinely helps quantum decoding, and lay the foundation for future studies on adaptive decoders capable of supporting real-time decoding in next-generation quantum hardware.

# Table of Contents

|                                                     |     |
|-----------------------------------------------------|-----|
| <b>Acronyms</b>                                     | VII |
| <b>List of Tables</b>                               | IX  |
| <b>List of Figures</b>                              | X   |
| <b>1 Introduction</b>                               | 1   |
| 1.1 Quantum Error Correction . . . . .              | 1   |
| 1.2 Fault tolerant system . . . . .                 | 2   |
| 1.3 Scalable quantum computing . . . . .            | 3   |
| 1.4 State of the art . . . . .                      | 4   |
| 1.5 Goal of the thesis . . . . .                    | 5   |
| 1.6 Thesis Outline . . . . .                        | 6   |
| <b>2 Quantum Information</b>                        | 7   |
| 2.1 Qubit . . . . .                                 | 7   |
| 2.1.1 Dirac's notation . . . . .                    | 7   |
| 2.1.2 Bloch's sphere . . . . .                      | 8   |
| 2.1.3 Hilbert Space . . . . .                       | 8   |
| 2.2 Quantum gates . . . . .                         | 11  |
| 2.2.1 Hadamard gate . . . . .                       | 11  |
| 2.2.2 CNOT . . . . .                                | 12  |
| 2.2.3 Pauli X . . . . .                             | 13  |
| 2.2.4 Pauli Z . . . . .                             | 14  |
| 2.2.5 Pauli Y . . . . .                             | 14  |
| 2.3 Key Properties of Multi-Qubit Systems . . . . . | 15  |
| 2.3.1 Entanglement . . . . .                        | 15  |
| 2.3.2 Multiple qubits . . . . .                     | 17  |
| 2.3.3 Non clonability . . . . .                     | 17  |
| 2.3.4 Quantum Parallelism . . . . .                 | 18  |

|          |                                                                         |           |
|----------|-------------------------------------------------------------------------|-----------|
| <b>3</b> | <b>Surface Code</b>                                                     | <b>20</b> |
| 3.1      | Structure and parameters . . . . .                                      | 20        |
| 3.1.1    | Noise models . . . . .                                                  | 21        |
| 3.2      | Encoding and the measurement cycle . . . . .                            | 22        |
| 3.3      | Stabilizers . . . . .                                                   | 23        |
| 3.4      | Ancillae and the syndrome . . . . .                                     | 24        |
| 3.5      | Homology classes . . . . .                                              | 24        |
| 3.6      | Degeneracy: minimum weight is not maximum likelihood . . . . .          | 25        |
| 3.7      | The role of the decoder . . . . .                                       | 26        |
| 3.8      | The Logical Error . . . . .                                             | 26        |
| <b>4</b> | <b>Decoding as a Sequential Decision Problem</b>                        | <b>29</b> |
| 4.1      | Reinforcement Learning . . . . .                                        | 29        |
| 4.1.1    | Reinforcement Learning Terminology . . . . .                            | 30        |
| 4.2      | Markov decision processes . . . . .                                     | 31        |
| 4.3      | Partially Observable Markov Decision Process . . . . .                  | 32        |
| 4.4      | Why a POMDP fits decoding . . . . .                                     | 32        |
| 4.5      | Mapping the surface code onto a POMDP . . . . .                         | 33        |
| 4.6      | Foundational Learning Paradigms . . . . .                               | 35        |
| 4.6.1    | Imitation Learning’s policy . . . . .                                   | 35        |
| 4.6.2    | Deep Q-Learning and Double DQN . . . . .                                | 36        |
| 4.6.3    | Supervised Classification . . . . .                                     | 36        |
| 4.7      | The role of Exploration . . . . .                                       | 37        |
| 4.8      | The minimum-weight attractor . . . . .                                  | 38        |
| <b>5</b> | <b>Introduction to Graph Neural Networks</b>                            | <b>39</b> |
| 5.1      | Advantages of using a graph . . . . .                                   | 39        |
| 5.2      | How a GNN works . . . . .                                               | 40        |
| 5.3      | A tour of message-passing layers . . . . .                              | 41        |
| 5.4      | Expressiveness of GNNs based on message passing . . . . .               | 44        |
| 5.5      | Heterogeneous graphs and factor graphs . . . . .                        | 46        |
| 5.6      | Why GNNs fit the problem . . . . .                                      | 48        |
| 5.7      | Going deep: over-smoothing, over-squashing, and stabilization . . . . . | 49        |
| 5.8      | Building GNNs in practice: PyTorch Geometric . . . . .                  | 50        |
| <b>6</b> | <b>Practical Implementation and Architecture</b>                        | <b>52</b> |
| 6.1      | Fast Syndrome Generation and Temporal Compression . . . . .             | 52        |
| 6.2      | Auxiliary Data Structures: Beliefs and Local Marginals . . . . .        | 54        |
| 6.3      | The Heterogeneous Graph Neural Network (HeteroGNN) . . . . .            | 58        |
| 6.4      | The Interactive Decoding Environment . . . . .                          | 61        |
| 6.5      | Imitation Learning Pipeline and Policy Bootstrapping . . . . .          | 61        |

|           |                                                                    |           |
|-----------|--------------------------------------------------------------------|-----------|
| 6.5.1     | Training loop and configuration . . . . .                          | 62        |
| 6.5.2     | Score metric and loss design . . . . .                             | 62        |
| 6.5.3     | Advanced Curriculum: Top-k Transition Strategy . . . . .           | 65        |
| 6.6       | The Lookahead Window . . . . .                                     | 66        |
| <b>7</b>  | <b>Alternative Decoders and Training Extensions</b>                | <b>68</b> |
| 7.1       | Atomic Model . . . . .                                             | 69        |
| 7.1.1     | Deep Q-Network in the Atomic model . . . . .                       | 69        |
| 7.1.2     | Atomic Model’s Description . . . . .                               | 70        |
| 7.2       | Supervised Model . . . . .                                         | 71        |
| 7.2.1     | Supervised Classification Framework . . . . .                      | 71        |
| 7.2.2     | From predicted class to physical correction . . . . .              | 71        |
| 7.2.3     | Supervised Model’s Description . . . . .                           | 72        |
| 7.3       | Reinforcement Refinement of the Imitation Decoder . . . . .        | 73        |
| 7.4       | Lookahead model . . . . .                                          | 74        |
| 7.4.1     | Syndrome generation with readout errors . . . . .                  | 75        |
| 7.4.2     | Lookahead training loop . . . . .                                  | 75        |
| <b>8</b>  | <b>Experiments and Results</b>                                     | <b>77</b> |
| 8.1       | Evaluation protocol . . . . .                                      | 77        |
| 8.2       | Single-shot code capacity . . . . .                                | 79        |
| 8.3       | Multi-round without measurement noise: the attractor regime . . .  | 81        |
| 8.4       | Reinforcement refinement . . . . .                                 | 83        |
| 8.5       | The phenomenological regime: readout errors . . . . .              | 84        |
| 8.5.1     | Training dynamics of the lookahead decoder . . . . .               | 84        |
| 8.5.2     | Diagnosis: compounding, not calibration . . . . .                  | 85        |
| 8.5.3     | Logical error rate under readout noise . . . . .                   | 86        |
| <b>9</b>  | <b>Discussion</b>                                                  | <b>90</b> |
| 9.1       | One principle, two margins . . . . .                               | 90        |
| 9.2       | The minimum-weight attractor as a design constraint . . . . .      | 90        |
| 9.3       | Closed-loop compounding: the real frontier of the readout regime . | 91        |
| 9.4       | Relation to prior work . . . . .                                   | 92        |
| 9.5       | Limitations . . . . .                                              | 92        |
| <b>10</b> | <b>Conclusions and Future Work</b>                                 | <b>94</b> |
| 10.1      | Conclusions . . . . .                                              | 94        |
| 10.2      | Future work . . . . .                                              | 95        |



# Acronyms

**BCE** Binary Cross-Entropy

**CNNs** Convolutional Neural Networks

**CNOT** Controlled-NOT

**DDQN** Double Deep Q-Network

**DQN** Deep Q-Network

**GAT** Graph Attention Network

**GCN** Graph Convolutional Network

**GIN** Graph Isomorphism Network

**GNN** Graph Neural Network

**GNNs** Graph Neural Networks

**IL** Imitation Learning

**LER** Logical Error Rate

**MDP** Markov Decision Process

**MWPM** Minimum Weight Perfect Matching

**POMDP** Partially Observable Markov Decision Process

**PyG** PyTorch Geometric

**Qubit** Quantum Bit

**RL** Reinforcement Learning

**RNNs** Recurrent Neural Networks

**WL** Weisfeiler-Lehman

# List of Tables

|     |                                                                                                                                                                                                                                                                                                                                                            |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.1 | Instantiation of the POMDP tuple on the surface code. . . . .                                                                                                                                                                                                                                                                                              | 34 |
| 5.1 | Main message-passing layer families and their fit to the decoding problem. . . . .                                                                                                                                                                                                                                                                         | 45 |
| 6.1 | Node types of the heterogeneous factor graph, their input features, and the shape of the feature matrix passed to the GNN at each step. $n = d^2$ is the number of data qubits, $n_{\text{anc}} = (d^2-1)/2$ the number of ancillae of each type; the <b>stab</b> feature width is 1 in the code-capacity regime and 3 with the lookahead window . . . . . | 59 |
| 8.1 | Single-shot logical error rate at distance $d = 5$ , physical error rate $p = 0.05$ under depolarizing noise . . . . .                                                                                                                                                                                                                                     | 79 |
| 8.2 | Multi-round LER at distance $d = 5$ with perfect measurements. Baselines: 2000 episodes per point; imitation decoder: 1000, 700 and 200 episodes respectively, on the same generator. . . . .                                                                                                                                                              | 82 |
| 8.3 | Multi-round LER under readout noise at distance $d = 5$ , evaluated over 20 rounds. . . . .                                                                                                                                                                                                                                                                | 86 |
| 9.1 | Out-of-distribution ablation: LER of the lookahead decoder and the plain imitation decoder, both evaluated at $p_{ro} = 0$ , 20 rounds. Despite sharing the same backbone, the lookahead decoder loses one to two orders of magnitude of performance outside its training regime                                                                           | 93 |

# List of Figures

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |    |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | The rotated surface code at distance $d = 5$ . Data qubits sit on the vertices; $X$ -type and $Z$ -type stabilizer plaquettes alternate in a checkerboard pattern. The two logical operators $\bar{X}$ and $\bar{Z}$ run as horizontal and vertical strings of single-qubit operators connecting opposite boundaries. . . . .                                                                                                                                                                                         | 27 |
| 6.1 | Visualization of the temporal compression strategy executed by the <code>create_detector_events_and_truth_fast_compressed()</code> function. Unchanged, static rounds (grey) are filtered out, forcing the environment to advance only when a dynamic syndrome change occurs (orange) . . . . .                                                                                                                                                                                                                       | 53 |
| 6.2 | The four operations of the belief life-cycle . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 57 |
| 6.3 | Mapping of a distance-3 patch onto the heterogeneous factor graph built by <code>graph_builder</code> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                       | 60 |
| 6.4 | One step of the imitation training loop. The environment produces the residual syndrome and the belief structures, assembled into the factor graph observation; the HeteroGNN maps it to per-qubit logits. The MWPM oracle, queried on the same state, supplies both the supervision targets for the weighted BCE loss and the expert action; the DAgger gate decides whether the action executed by the environment is the model's or the expert's, sliding from teacher-forced to autonomous over training. . . . . | 64 |
| 6.5 | Training dynamics of the imitation decoder (run 39). Top: per-episode agreement score against the MWPM expert under the six-level noise curriculum ( $p = 0.0005 \rightarrow 0.016$ ); the score plateaus at 0.83–0.88. Bottom: training loss over the same 150 episodes. . . . .                                                                                                                                                                                                                                     | 65 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |    |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 7.1 | Training dynamics of the RL fine-tuning phase (run 41, $d = 5$ , $p = 0.005$ , noiseless measurements). <b>Top:</b> agreement score (moving average window 20, with interquartile band); the orange dashed line marks the imitation baseline ( $\approx 0.63$ ) from which the run was initialized. <b>Bottom:</b> training loss over the same episodes. The score oscillates in a narrow band around the initialization for the full 800-episode budget, never breaking above the baseline, the defining signature of the stall failure mode. The slow downward drift of the loss indicates that the optimizer is active but unable to move the policy: the gradient signal from the sparse terminal reward is too weak to overcome the local minimum induced by the minimum-weight attractor . . . . . | 74 |
| 8.1 | MWPM logical error rate versus physical error rate at $d = 3, 5, 7$ (code capacity, depolarizing noise, 20,000 samples per point). The curves cross at $p_{\text{th}} \approx 0.14$ (shaded band); the operating point $p = 0.05$ of the headline comparison sits well below threshold, where increasing distance suppresses the LER and a margin over MWPM is meaningful.                                                                                                                                                                                                                                                                                                                                                                                                                               | 79 |
| 8.2 | Comparison of Logical Error Rates (LER). Left: Exact ML decoding at $d = 3$ shows that the gain is strictly due to joint $X - Z$ reasoning. Right: The $d = 5$ supervised model achieves a similar relative reduction. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 81 |
| 8.3 | The cost of non-minimality: MWPM versus a random-pairing decoder that silences the same syndromes with heavier corrections ( $d = 5$ , $p = 0.05$ , 6000 shots, same error samples). A 16% average weight excess translates into a $5\times$ higher logical error rate: the extra weight means a higher probability of pairing defects along the wrong homology class . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                          | 83 |
| 8.4 | Learning curve analysis. (a) Stagnation due to under-fitting/learning rate too low. (b) Catastrophic forgetting due to aggressive gradient updates. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 84 |
| 8.5 | Smoothed per-episode score of the lookahead decoder. Green dotted lines: DAgger events; red dotted line: end of the top- $k$ constraint (episode 300). The three regimes discussed in the text are clearly visible. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 85 |
| 8.6 | LER comparison . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 87 |
| 8.7 | LER versus number of rounds at $(p, p_{ro}) = (0.01, 0.01)$ , $d = 5$ , log scale. Annotations report the naive-to-model ratio at each horizon: the lookahead advantage decays from $2.2\times$ at 5 rounds to $1.5\times$ at 40, the closed-loop compounding signature. Baselines: 2000 episodes per point; lookahead model: 2000 episodes (500 at 30 and 40 rounds).                                                                                                                                                                                                                                                                                                                                                                                                                                   | 89 |

# Chapter 1

## Introduction

The field of quantum computing has witnessed remarkable advancements in recent years, promising to revolutionize various domains by taking advantage of quantum mechanics' unique properties. However, the practical realization of quantum computers faces significant challenges, particularly in the realm of error correction. Quantum systems are inherently fragile and susceptible to errors caused by environmental noise and operational imperfections. To address these issues, **Quantum Error Correction codes** have been developed, with the **Surface code** being one of the most prominent and promising candidates for fault-tolerant quantum computation.

In this chapter we will introduce the basic concepts of quantum error correction, fault tolerance and the state of the art in decoding algorithms, setting the stage for the core arguments of this work. The thesis is divided into two main sections: the first one is dedicated to the introduction of the problem and to the theoretical background, while the second one is dedicated to the description of the original work, the results obtained and the conclusions.

### 1.1 Quantum Error Correction

Every computing system has to deal with random bit flips. It consists of random change of bit's value, from 0 to 1 or viceversa, caused by cosmic rays or electromagnetic interference. This problem is usually solved through redundancy and error-correcting codes, which guarantee that the information is valid every time the user needs it.

What is a minor inconvenience in classical computing becomes a fundamental obstacle in quantum systems. First of all, in quantum systems we cannot duplicate the information stored in a Qubit, then we cannot read the value either, because it would mean to lose the probability linked to the read value, which is the most

important part of the information. For roughly a decade after the discovery of quantum algorithms, it was widely suspected that these two obstacles, no cloning and the destructiveness of measurement, made quantum error correction impossible in principle. The turning point came in 1995, when Shor exhibited a nine-qubit code that protects one logical qubit against any single-qubit error, by measuring parities of groups of qubits instead of the qubits themselves [1]. Gottesman then added the stabilizer formalism, in which a code is defined by the set of commuting Pauli operators that leave its states unchanged [2]. The final ingredient was topology: Kitaev showed that stabilizers could be made local by arranging qubits on a two-dimensional lattice, so that every check involves only a handful of neighbouring qubits and the logical information is stored in a global, topological property of the whole surface [3, 4]. The surface code studied in this thesis is the direct descendant of this line: it combines the algebra of stabilizers with the locality of a planar lattice, which is what makes it implementable on real hardware, where qubits can only interact with their neighbours.

This is the starting point of the thesis: in order to pave the way for the future of computing and for quantum systems, we need to have the knowledge to create a fault-tolerant quantum system.

## 1.2 Fault tolerant system

A system is said to be fault-tolerant if it is able to operate without interruptions or data loss. In classical computing it means creating a system that can continue to function despite hardware or software failures, while in a quantum system it means being able to perform complex, long-running calculations without its fragile quantum states being destroyed by environmental noise. Unlike classical systems, where bits are physically robust, qubits are extremely sensitive; therefore, a quantum system becomes fault-tolerant when it can detect and correct errors faster than they accumulate.

The theoretical foundation that makes fault tolerance achievable is the **threshold theorem**[5, 6]. It establishes a quantitative relationship between the physical error rate of the hardware and the logical error rate of the encoded computation:

$$p_L \leq C \left( \frac{p}{p_{th}} \right)^{\lfloor (d+1)/2 \rfloor}. \quad (1.1)$$

The quantities involved are:

- $p_L$ : the logical error rate, i.e. the probability that the encoded information is corrupted after a correction cycle;
- $p$ : the physical error rate, determined by the quality of the hardware;

- $p_{th}$ : the threshold error rate, the critical value below which error correction becomes effective;
- $d$ : the code distance, a parameter that controls how many physical qubits are used to encode a single logical qubit;
- $C$ : a positive constant that depends on the specific code and noise model.

For a system to be fault-tolerant,  $p_L$  must be kept small. The formula shows exactly when this is possible: the fraction  $p/p_{th}$  must be less than one, which requires  $p < p_{th}$ . When this condition holds, two levers are available to push  $p_L$  down. The first is to reduce  $p$ , i.e. to build better hardware. The second, and more powerful, is to increase  $d$ : since  $p/p_{th} < 1$ , raising the exponent drives the entire expression exponentially toward zero, allowing arbitrarily reliable logical qubits to be obtained simply by encoding information across more physical qubits[7].

Crucially,  $p_{th}$  is not determined by the code alone: it also depends on the decoder, the part of the architecture whose primary function is to monitor the system's status and correct any anomalies. A decoder that is too inaccurate introduces errors of its own, and one that is too slow allows uncorrected errors to accumulate between correction cycles. Both effects reduce the effective threshold the system can achieve. So, to design a decoder that keeps  $p_{th}$  as high as possible, while remaining efficient enough to scale with  $d$  is a central requirement for any fault-tolerant quantum system.

### 1.3 Scalable quantum computing

The threshold theorem has a direct implication for scalability. Below threshold, the logical error rate decreases exponentially with  $d$ , meaning that arbitrarily reliable logical qubits can be obtained by simply increasing the code size. This is what makes large-scale quantum computation conceivable: a system operating below threshold can, in principle, be scaled to any target reliability.

However, fault tolerance alone is not sufficient for scalability. The decoder must also be able to keep up with an increasing code size; in fact from a computational complexity perspective, finding the exact optimal correction, known as maximum-likelihood decoding, is an **NP**-hard problem [8]. In order to let the decoder operate efficiently, it must reduce this task to the class **P** [9], having its runtime grow at most polynomially with the number of physical qubits, otherwise the correction overhead will behave as a bottleneck [9]. Fault tolerance and scalability are therefore not independent goals: a system that corrects errors reliably but cannot scale its decoder to larger codes is no more useful than one that does not correct errors at all.

## 1.4 State of the art

The most widely adopted decoding algorithm is Minimum Weight Perfect Matching (MWPM) [10, 11, 7], which interprets the syndrome defects as nodes of a graph and returns the pairing of defects whose total weight (i.e. physical error count) is minimal. Modern implementations such as PyMatching [12, 13] decode millions of rounds per second, and MWPM remains the de facto baseline for the surface code. In this thesis, it plays a dual role: teacher in the imitation learning phase and benchmark in the results.

MWPM is not the only classical option. Union-Find decoding trades a small amount of accuracy for almost-linear worst-case complexity, growing clusters around defects and correcting them when they become neutral [14]; belief propagation with ordered-statistics post-processing (BP-OSD) applies iterative message passing on the code’s factor graph and is the reference decoder for general quantum LDPC codes [15]. Another decoder we should mention is the correlated matching, which partially addresses the  $Y$ -error blindness of MWPM: after a first matching pass in one sector, the edge weights of the other sector are re-weighted to account for the  $Y$ -error correlations exposed by the first pass. It requires no learning and serves in this thesis as an additional classical baseline in the multi-round experiments. All these algorithms compete under a constraint that shapes the entire field: decoding must eventually run in real time. On superconducting hardware a full stabilizer measurement cycle lasts on the order of a microsecond [16, 9], and a decoder that falls behind lets errors accumulate faster than they are removed, effectively lowering the threshold. Accuracy per round and time per round are therefore the two axes on which every decoder, classical or learned, must be judged.

MWPM, however, is suboptimal in two documented ways. First, it decodes the  $X$  and  $Z$  sectors on two **separate graphs**, so it cannot exploit the fact that, under depolarizing noise, a  $Y$  error is a single likely event rather than two independent ones. Second, it returns the minimum-weight correction rather than the maximum-likelihood homology class: it ignores the degeneracy of the code, i.e. the fact that many distinct physical corrections are logically equivalent and their probabilities should be summed.

These limitations have motivated a growing body of learned decoders, classifying this as an active area of research: no learned decoder has yet established itself as a definitive replacement for MWPM across all noise regimes, and, crucially for this thesis, the conditions under which a learned decoder can beat MWPM, and those under which it provably cannot, are rarely made explicit.

## 1.5 Goal of the thesis

This thesis investigates the specific conditions under which a learned decoder can outperform Minimum Weight Perfect Matching (MWPM) on the surface code, providing an architectural design tailored to exploit these margins. Rather than assuming that increased learning capacity automatically yields a better decoder, we systematically map the boundaries of the problem. First, we argue that any sequential policy trained to imitate the minimum-weight behavior of MWPM cannot outperform it, and in practice does not even reach its performances, nor can it be improved by reinforcement fine-tuning: the policy inherited from MWPM sits at a local optimum, so any perturbation large enough to change its behaviour also destroys the minimum-weight discipline it had acquired (Section 7.3). Second, we isolate the  $X - Z$  correlations of  $Y$  errors, where MWPM is provably suboptimal, and the temporal signature of readout faults.

The primary contributions of this work are organized as follows:

1. A Partially Observable Markov Decision Process (POMDP) formulation of multi-round decoding, implemented via a Heterogeneous Graph Neural Network operating on a factor graph enriched with explicitly maintained Bayesian local marginals;
2. A two-phase training pipeline having a DAgger-style imitation learning phase based on MWPM, followed by a reinforcement learning finetuning stage;
3. A systematic comparison of three distinct learning paradigms against MWPM in the code-capacity regime: an atomic Double-DQN agent, a supervised joint-class decoder, and the imitation-based sequential decoder. We show that the supervised decoder outperforms MWPM by leveraging  $Y$  correlations to reach the estimated maximum-likelihood optimum, whereas the atomic agent matches MWPM and the sequential decoder remains bounded by its teacher;
4. An extension to the phenomenological noise regime with measurement uncertainty, utilizing a fixed-lookahead observation window to capture the temporal signatures of readout faults;

The ultimate objective of this thesis is to demonstrate that machine learning does not surpass classical benchmarks through a generalized, abstract capability, but precisely where traditional algorithms discard exploitable structural information. Establishing this theoretical and practical map provides a more robust foundation for learned quantum error correction than isolated benchmarking results.

## 1.6 Thesis Outline

The thesis is structured as follows:

- Chapter 2 reviews the foundations of quantum information such as qubits, quantum gates and multi-qubit properties such as entanglement and non-clonability, establishing the formalism used throughout the rest of the thesis;
- Chapter 3 introduces the surface code: its stabilizer formalism, the encoding and measurement cycle, homology classes and the degeneracy argument showing why minimum-weight decoding is not maximum-likelihood decoding;
- Chapter 4 explains why decoding is a sequential decision problem, formulating it as a POMDP and formalizing the minimum-weight attractor that any greedy syndrome-nulling policy converges to;
- Chapter 5 introduces the Graph Neural Network (GNN) paradigm and the architectural ingredients, such as heterogeneous factor graphs, attention, spatio-temporal mapping, adopted in this work;
- Chapter 6 details the practical implementation: fast syndrome generation, the auxiliary belief structures, the Heterogeneous Graph Neural Network, the interactive decoding environment, and the imitation learning pipeline used to bootstrap the policy;
- Chapter 7 presents alternative decoding paradigms; an atomic Double-DQN (DDQN) agent and a supervised joint-class decoder, and compares their performance against MWPM in the code-capacity regime, showing that the supervised decoder reaches the maximum-likelihood optimum;
- Chapter 8 provides the experimental evaluation across the code-capacity and phenomenological regimes, comparing all decoders against MWPM;
- Chapter 9 discusses the results, relating them to prior work and to the theoretical predictions of Chapter 4, and states the limitations of this study;
- Chapter 10 concludes and outlines future work, including the corrected readout run, recovery-oriented training, and scaling to larger code distances;

## Chapter 2

# Quantum Information

This section has the aim to review the basics of quantum error correction, starting from qubits to arrive at the description of the problem to solve.

### 2.1 Qubit

The classical bit is defined as a unit of information that can take on one of two values, typically represented as 0 and 1. In contrast, a quantum bit, or qubit, which is the fundamental unit of quantum information, is defined as a vector in a complex vector space, known as a Hilbert space, and can be represented as a linear combination of two basis states, conventionally  $|0\rangle$  and  $|1\rangle$ . A generic vector in this space can be written as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (2.1)$$

where  $\alpha$  and  $\beta \in \mathbb{C}$ , where the condition is  $|\alpha|^2 + |\beta|^2 = 1$  and where  $|\alpha|^2$  and  $|\beta|^2$  represent the probabilities of measuring the qubit in the states  $|0\rangle$  and  $|1\rangle$ , respectively. When  $\alpha$  and  $\beta$  are both nonzero, the qubit does not occupy a definite state: it exists simultaneously in both  $|0\rangle$  and  $|1\rangle$ , with the coefficients encoding how much of each. This is called **superposition**, and it is only resolved at the moment of measurement, which irreversibly collapses the qubit into one of the two basis states with the corresponding probability. This property is one of the key features that distinguishes qubits from classical bits and is fundamental to the power of quantum computing.

#### 2.1.1 Dirac's notation

Working with quantum states requires introducing the notation used to represent quantum states and operations. The most common one is the Dirac's notation, also known as Bra-Ket, that allows to represent quantum states as a vector in a

complex vector space, and quantum operations as linear transformations on these vectors. A generic vector in this space can be written as:

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = |\mathbf{v}\rangle \quad (2.2)$$

where the column vector  $|\mathbf{v}\rangle$  is called "ket-v", while the dual of it is represented as a "bra-v":

$$\langle v| = |v\rangle^\dagger = (v_0^* \quad v_1^* \quad \cdots \quad v_n^*) \quad (2.3)$$

where  $v_i^*$  denotes the complex conjugate of  $v_i$ , so that  $\langle v|$  is the conjugate transpose of  $|v\rangle$ . The inner product between two vectors  $|\mathbf{v}\rangle$  and  $|\mathbf{w}\rangle$  is denoted as  $\langle \mathbf{v}|\mathbf{w}\rangle$ , while the outer product is denoted as  $|\mathbf{v}\rangle\langle \mathbf{w}|$ .

### 2.1.2 Bloch's sphere

The state of a qubit can be visualized on the Bloch sphere, where the north and south poles represent the basis states:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (2.4)$$

and any point on the surface of the sphere is represented by a linear combination of these states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.5)$$

### 2.1.3 Hilbert Space

Operating with qubits requires a mathematical environment that satisfies specific physical and geometric properties. As previously established, a single qubit is not confined to a single definite classical state, but it is expressed as a linear combination, or superposition, of the two computational basis states  $|0\rangle$  and  $|1\rangle$ :

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where  $\alpha, \beta \in \mathbb{C}$  represent the probability amplitudes. To comply with the conservation of total probability, the coefficients must satisfy the normalization condition, meaning that the sum of their squared magnitudes must equal one:

$$|\alpha|^2 + |\beta|^2 = 1$$

However, a simple vector space that allows for linear combinations is insufficient to capture the full physics of a quantum system; in fact, it has to be guaranteed the existence of an **inner product**  $\langle \cdot | \cdot \rangle : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{C}$ , which is essential for the following reasons:

- **Mathematical Definition of Probability (Norm):** The inner product induces a norm  $\|\psi\| = \sqrt{\langle \psi | \psi \rangle}$  that measures the geometric length of a state vector. Without an inner product, it would be impossible to formalise the normalization condition ( $|\alpha|^2 + |\beta|^2 = 1$ ). Requiring the inner product ensures that all physical state vectors have a norm of exactly one ( $\langle \psi | \psi \rangle = 1$ );
- **Physical Distinguishability (Orthogonal Bases):** The inner product defines the angle between states, where two vectors are orthogonal if  $\langle \psi | \phi \rangle = 0$ . In quantum mechanics, orthogonality is synonymous with perfect physical distinguishability during a measurement. It guarantees that the computational basis states  $|0\rangle$  and  $|1\rangle$  can be unambiguously read out;

In addition to this, the vector space must also guarantee the property of **completeness**. A space is complete if every Cauchy sequence of vectors converges to a limit that is itself an element of that same space, ensuring the absence of any mathematical "gaps". Completeness guarantees that as a state vector undergoes continuous rotations, it remains within the boundaries of a valid state space at every infinitesimal point in time, preventing the system from becoming mathematically undefined.

The union of these requirements, a complex vector space that supports linear superpositions, equipped with an inner product and completed, leads directly to the definition of a **Hilbert space**  $\mathcal{H}$  [17]. For a single qubit, this space corresponds to  $\mathbb{C}^2$ . However, the vector representation within a Hilbert space introduces a mathematical redundancy: multiplying a state vector by an overall global phase  $e^{i\theta}$  does not alter any measurable probability. This means that two mathematically distinct vectors, such as  $|\psi\rangle$  and  $e^{i\theta}|\psi\rangle$ , describe the exact same physical state.

For instance, the vectors  $|0\rangle$  and  $-|0\rangle$  (where the global phase is  $e^{i\pi} = -1$ ) point in opposite directions within the abstract Hilbert space. Yet, if measured in a laboratory, both vectors yield the outcome 0 with a definitive 100% probability, making them physically indistinguishable.

While the true space of physically unique states is the projective Hilbert space  $\mathbb{P}(\mathcal{H})$ , which removes this phase redundancy and maps a single qubit onto the surface of the Bloch sphere, the redundant vector notation within  $\mathcal{H}$  is strictly maintained in practice because linear algebra is far more straightforward to calculate than projective geometry.

The algebraic architecture of this Hilbert space yields three direct consequences that serve as the operational building blocks for both quantum algorithms and fault-tolerant architectures:

- **Physical measurements are represented by Hermitian operators:**

In classical physics, measuring a system means simply reading a value. In quantum mechanics, a measurement is an active process that alters the state vector. For this reason, a measurement is represented mathematically by an **operator** (a matrix) that acts on the Hilbert space.

We specifically require these matrices to be **Hermitian** ( $A = A^\dagger$ ) due to a fundamental physical constraint: any outcome measured in a real-world laboratory must be a real number (such as a voltage or a physical count), as complex numbers cannot be read by instruments.

To understand why this mathematical property guarantees a real outcome, we must look at the definition of the adjoint operator ( $\dagger$ ), which performs both a matrix transposition and a complex conjugation. In the complex field, the only numbers that remain completely unchanged under complex conjugation are real numbers ( $z = z^* \iff z \in \mathbb{R}$ ). By enforcing the condition  $A = A^\dagger$ , we impose this exact symmetry on the entire operator. When we calculate the expectation value of a state  $|\psi\rangle$  under this operator, the definition of the inner product ensures that:

$$\langle\psi|A|\psi\rangle = \langle\psi|A^\dagger|\psi\rangle = \langle\psi|A|\psi\rangle^*$$

Because the number  $\langle\psi|A|\psi\rangle$  must be equal to its own complex conjugate, its imaginary part is mathematically forced to be zero. As a direct consequence of this algebraic restriction, the eigenvalues of a Hermitian operator, which represent the definitive measurable states of the system, are mathematically stripped of any imaginary component, leaving exclusively real numbers;

- **Time evolution is governed by unitary operators:** When a closed quantum system changes over time, whether it is naturally evolving or being manipulated by an external control, its movement is dictated by the Schrödinger equation. A fundamental consequence of this physical law is that the transformation must be executed by a **unitary operator**  $U$ , which satisfies the algebraic condition  $UU^\dagger = U^\dagger U = I$ .

The physical necessity for unitarity comes back to the conservation of probability. As a qubit evolves, the total probability of all its possible states combined must strictly remain 1 at every moment of the calculation. Geometrically, a unitary matrix acts as a rigid rotation inside the Hilbert space: it alters the direction of the state vector to perform a calculation, but it is mathematically forbidden from changing its length, because it represents the total probability, which must remain equal to 1. Every ideal quantum gate that handles information is a consequence of this unitary rule;

- **Composite systems scale via the tensor product:** When  $n$  independent qubits are combined to form a larger register, their joint state space is the tensor product of their individual spaces,  $\mathcal{H}^{\otimes n} \cong \mathbb{C}^{2^n}$ , whose dimension scales exponentially as  $2^n$ . This exponential explosion is the structural origin of quantum computing's power, as a classical computer requires  $2^n$  complex coefficients to track the state vector [18]. More importantly for QEC, the tensor product space allows for the existence of **entangled states**, global states that cannot be factored into individual qubit states that we will see later, that allow QEC to exploit this non-local structure, embedding logical information into highly entangled multi-qubit states so that localized, single-qubit physical errors can be detected and isolated without information loss;

## 2.2 Quantum gates

As in classical computing, quantum computing is equipped with a set of elementary operations, called quantum gates, that can be applied to qubits to perform computations. These gates are represented by unitary matrices that act on the state vectors in the Hilbert space. In this section we will review the most common quantum gates, which will be used in the rest of the thesis.

### 2.2.1 Hadamard gate

The Hadamard gate  $H$  is a single-qubit unitary operator that maps each computational basis state into an equal superposition of  $|0\rangle$  and  $|1\rangle$ . Its matrix representation in the computational basis is

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad (2.6)$$

and its action on the basis states is

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle, \quad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle. \quad (2.7)$$

The Hadamard gate is both Hermitian and unitary,  $H = H^\dagger$  and  $H^2 = I$ , which means that applying it twice in a row returns the qubit to its original state [17].

$H$  in quantum computation has a key role: it is the elementary gate used to create superpositions starting from a deterministic classical input. In practice almost every quantum algorithm begins by applying a layer of Hadamard gates to a register prepared in  $|0\rangle^{\otimes n}$ , producing the uniform superposition

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle, \quad (2.8)$$

where:

- $|0\rangle^{\otimes n}$ : Represents the initial state of the quantum register, consisting of  $n$  qubits all initialized to the  $|0\rangle$  state via a tensor product (e.g.,  $|000\rangle$  for  $n = 3$ );
- $H^{\otimes n}$ : Denotes the parallel application of the Hadamard gate ( $H$ ) to each of the  $n$  individual qubits simultaneously;
- $\sum_{x \in \{0,1\}^n} |x\rangle$ : Represents the sum over all possible  $2^n$  binary strings of length  $n$ , effectively generating a linear combination of all computational basis states;
- $\frac{1}{\sqrt{2^n}}$ : Acts as the normalization factor in the Hilbert space, ensuring that the total probability of all states sums up to 1 since each state has a measurement probability of  $\left(\frac{1}{\sqrt{2^n}}\right)^2 = \frac{1}{2^n}$ ;

### 2.2.2 CNOT

The Controlled-NOT, or CNOT, is a two-qubit gate which flips the state of a target qubit conditionally on the value of a control qubit. It can be written compactly in projector form as

$$\text{CNOT} = |0\rangle\langle 0|_c \otimes I_t + |1\rangle\langle 1|_c \otimes X_t, \quad (2.9)$$

so that for a computational basis input  $|q_0\rangle|q_1\rangle$  it acts as

$$\text{CNOT} |q_0\rangle|q_1\rangle = |q_0\rangle|q_1 \oplus q_0\rangle, \quad (2.10)$$

where  $\oplus$  denotes addition modulo two, making it the quantum generalisation of the classical XOR operation. Considering the computational basis  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$  its matrix form is

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (2.11)$$

Since the matrix is real and symmetric, CNOT is both unitary and Hermitian, which implies  $\text{CNOT}^2 = I$ : applying the gate twice returns the system to its original state.

A first key property of CNOT is its ability to generate entanglement. Preparing the control qubit in the superposition  $|+\rangle$  via a Hadamard gate and applying CNOT yields

$$\text{CNOT} (H \otimes I) |00\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad (2.12)$$

which is one of the four Bell states, a maximally entangled two-qubit state that cannot be written as any tensor product of single-qubit states [17, 19].

A second key property is that CNOT belongs to the Clifford group: it maps every element of the two-qubit Pauli group to another Pauli element under conjugation [2, 20]. Two representative identities are

$$\text{CNOT}(X_c \otimes I_t) \text{CNOT}^\dagger = X_c \otimes X_t, \quad \text{CNOT}(I_c \otimes Z_t) \text{CNOT}^\dagger = Z_c \otimes Z_t. \quad (2.13)$$

This conjugation structure shows how Pauli operators transform when passing through a CNOT gate, where the subscripts  $c$  and  $t$  denote the control and target qubits, respectively, separated by the tensor product  $\otimes$ . Because the CNOT gate distinguishes between the roles of the two qubits, the propagation of Pauli errors is asymmetric: a bit-flip  $X_c$  on the control qubit propagates forward to the target qubit resulting in  $X_c \otimes X_t$ , whereas a phase-flip  $Z_t$  on the target qubit propagates backward to the control qubit via phase kickback, resulting in  $Z_c \otimes Z_t$ .

These relations also describe how errors propagate through the gate: an  $X$  error on the control spreads to the target ( $X_c \rightarrow X_c X_t$ ), while a  $Z$  error on the target back-propagates to the control ( $Z_t \rightarrow Z_c Z_t$ ) [17]. Together with arbitrary single-qubit gates, CNOT forms a universal gate set for quantum computation [17, 21]. Both of these properties will play a central role on the surface code; on one hand, the entangling ability of CNOT is used in every syndrome extraction circuit: an ancilla qubit is prepared in  $|0\rangle$  (or  $|+\rangle$ ), entangled with its neighbouring data qubits via a sequence of CNOT gates, and then measured to extract the stabilizer eigenvalue without disturbing the logical state. On the other hand, the error-propagation rules determine how a single physical fault in the measurement circuit can spread to multiple qubits, and therefore directly shape the structure of the syndrome graph that the decoder must interpret [7].

### 2.2.3 Pauli X

The Pauli  $X$  gate, often called the **bit-flip** gate, is the quantum analogue of the classical NOT gate. Its matrix representation in the computational basis is

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad (2.14)$$

and it swaps the two computational basis states:

$$X|0\rangle = |1\rangle, \quad X|1\rangle = |0\rangle. \quad (2.15)$$

Geometrically,  $X$  corresponds to a rotation by  $\pi$  radians around the  $\hat{x}$  axis of the Bloch sphere [17].

In quantum error correction, the  $X$  gate is one of the two elementary error operators that one needs to be able to detect and correct. Any single-qubit error can be expanded as a linear combination of the identity and the three Pauli operators,

so being able to correct  $X$  and  $Z$  errors (and therefore also  $Y = iXZ$  errors) is sufficient to correct **any** single-qubit error. This is the basis of the celebrated Pauli decomposition of arbitrary noise and motivates the entire stabilizer formalism that underlies the surface code [2, 17].

### 2.2.4 Pauli Z

The Pauli  $Z$  gate is also known as the **phase-flip** gate. Its matrix representation is

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad (2.16)$$

and its action on the computational basis states is

$$Z|0\rangle = |0\rangle, \quad Z|1\rangle = -|1\rangle. \quad (2.17)$$

Although  $Z$  leaves the populations of the computational basis states unchanged, it flips the relative phase of any superposition involving  $|1\rangle$ . For example,

$$Z(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle, \quad (2.18)$$

which is a transformation with no classical counterpart. Geometrically  $Z$  is a rotation by  $\pi$  around the  $\hat{z}$  axis of the Bloch sphere [17].

In the surface code,  $Z$ -type errors are detected by a different set of stabilizers from those used to detect  $X$ -type errors, which is the fundamental reason why the code can correct both classes of errors independently. The interplay between bit-flip and phase-flip errors is one of the central themes of the rest of this thesis.

### 2.2.5 Pauli Y

The Pauli  $Y$  gate combines a bit-flip and a phase-flip in a single operation, plus an overall imaginary phase. Its matrix form is

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad (2.19)$$

and it can be written as the product

$$Y = iXZ. \quad (2.20)$$

Its action on the computational basis is

$$Y|0\rangle = i|1\rangle, \quad Y|1\rangle = -i|0\rangle. \quad (2.21)$$

Geometrically  $Y$  is a rotation by  $\pi$  around the  $\hat{y}$  axis of the Bloch sphere [17].

The three Pauli matrices  $X$ ,  $Y$ ,  $Z$ , together with the identity  $I$ , form a basis for the space of  $2 \times 2$  complex matrices and they generate the single-qubit Pauli group. This group plays a central role in the stabilizer formalism of quantum error correction [2].

## 2.3 Key Properties of Multi-Qubit Systems

The great advantage of quantum computing is that the state space of a multi-qubit system grows exponentially with the number of qubits, allowing for the representation and manipulation of complex quantum states that can encode vast amounts of information. However, this exponential growth also introduces unique phenomena that have no classical analogue, such as entanglement and non-local correlations, which are essential for the power of quantum algorithms and quantum error correction. This section presents the key properties that will be used and will explain the theory behind the rest of the thesis.

### 2.3.1 Entanglement

Quantum entanglement is a phenomenon in which two or more particles become linked in such a way that their properties become interdependent: measuring the state of one of them instantly determines the state of the others, regardless of the distance between them. It is a purely quantum mechanical correlation between two or more subsystems that has no classical analogue. Formally, a pure state  $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$  is called **separable** if it can be written as a tensor product

$$|\psi\rangle = |\phi_A\rangle \otimes |\phi_B\rangle \quad (2.22)$$

If no such factorization exists, the state is called **entangled** [17]. This is one of the fundamental properties that enables the development of quantum teleportation, quantum cryptography, and the generation of non-classical correlations through two-qubit operations like the CNOT gate.

The simplest and most important examples of maximally entangled two-qubit states are the four **Bell states**

$$|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), \quad (2.23)$$

$$|\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle), \quad (2.24)$$

which form an orthonormal basis of the two-qubit Hilbert space. They are created by the canonical circuit of a Hadamard gate on the first qubit followed by a CNOT:

$$|\Phi^+\rangle = \text{CNOT}(H \otimes I)|00\rangle = \text{CNOT} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \quad (2.25)$$

The remaining three Bell states are obtained by applying Pauli corrections to one of the two qubits before or after the circuit. More generally, any two-qubit unitary that maps at least one separable input to an entangled output is called an **entangling gate** [17].

Entanglement is a fundamental resource in quantum information for three distinct reasons:

- **Quantum teleportation:** when a qubit is measured, its quantum state collapses immediately. If two qubits are entangled, a measurement on the first causes both to collapse at the exact same moment, completely independent of how far apart they are in the universe. Quantum teleportation is a protocol that exploits this property to transfer an unknown quantum state from one location to another without physically moving the qubit itself; if the sender measures the unknown qubit together with their half of a shared Bell state, the measurement will trigger an immediate collapse of the entire system, destroying the original qubit while instantaneously forcing the distant partner into a correlated state that holds the hidden quantum information. Then, by transmitting two classical bits, the sender provides the missing key needed to unlock and reconstruct the original state, precisely replicating an unknown qubit at a remote location and bypassing the need to physically transport the quantum information carrier[17];
- **Quantum error correction:** entanglement is the mechanism that makes QEC possible without violating the no-cloning theorem. The smallest example is worth working through explicitly, because the surface code repeats its logic at scale. To protect  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  against bit flips, two ancillary qubits prepared in  $|0\rangle$  are entangled with it through two CNOT gates (data qubit as control):

$$\alpha|0\rangle + \beta|1\rangle \xrightarrow{\text{CNOT}_{12}, \text{CNOT}_{13}} \alpha|000\rangle + \beta|111\rangle. \quad (2.26)$$

Note what this state is **not**: it is not three copies of  $|\psi\rangle$ , which would violate the no-cloning theorem, indeed  $(\alpha|0\rangle + \beta|1\rangle)^{\otimes 3}$  contains cross terms like  $|001\rangle$  that the encoded state lacks. It is one logical qubit smeared over three physical ones. Suppose now a bit flip strikes the second qubit:  $\alpha|010\rangle + \beta|101\rangle$ . Measuring the two parity operators  $Z_1Z_2$  and  $Z_2Z_3$  via ancillae, as in the previous section, returns  $(-1, -1)$ : both parities are violated, which identifies qubit 2 as the culprit **without measuring any individual qubit**. The Born rule confirms that the superposition survives: both branches  $|010\rangle$  and  $|101\rangle$  have odd parity on the same pairs, so the parity measurement does not distinguish them and  $\alpha, \beta$  are untouched. Applying  $X_2$  restores the encoded state exactly. Every ingredient of the surface code is already here; encoding by entanglement, parity checks through ancillae, a syndrome that locates errors without reading data, but with one limitation: this code detects only bit flips. Protecting against phase flips too requires checks of both types, and arranging both families consistently on one lattice is precisely what the surface code achieves [1, 17].

- Entanglement underlies the **exponential** dimension of the multi-qubit Hilbert space: the state space of  $n$  qubits has dimension  $2^n$ , and the overwhelming majority of states in it are entangled in a non-trivial way. It is precisely this structure that quantum algorithms exploit and that makes the surface code, where the logical qubit is stored in a highly entangled state of  $O(d^2)$  physical qubits, capable of suppressing local errors exponentially in the code distance  $d$  [7].

### 2.3.2 Multiple qubits

A system of  $n$  qubits is described by a state vector in the tensor product Hilbert space  $\mathcal{H}^{\otimes n} = (\mathbb{C}^2)^{\otimes n}$ , whose dimension is  $2^n$ . The computational basis of this space is formed by the  $2^n$  states

$$|x_1 x_2 \cdots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle, \quad x_i \in \{0,1\}, \quad (2.27)$$

and a generic pure state is a linear superposition

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle, \quad \sum_x |\alpha_x|^2 = 1. \quad (2.28)$$

While a classical  $n$ -bit register can be in only one of  $2^n$  states at any given time, a quantum register exists in a superposition characterised by  $2^n$  complex amplitudes (constrained only by the normalisation condition and by an overall phase). This exponential growth of the state space is the source of both the computational power of quantum computers and the extreme difficulty of simulating them classically [17, 18].

It is important to stress that not every state in  $\mathcal{H}^{\otimes n}$  can be written as a tensor product of single-qubit states: indeed, the overwhelming majority of  $n$ -qubit states are entangled in a non-trivial way. The peculiar geometry of this exponentially large but highly constrained state space is exactly what quantum error-correcting codes exploit in order to protect a small logical subspace from the action of local noise.

### 2.3.3 Non clonability

The **no-cloning theorem**, proved independently by Wootters and Zurek [22] and by Dieks [23] in 1982, states that there exists no unitary operator  $U$  capable of producing a perfect copy of an arbitrary, unknown quantum state. More formally, there is no  $U$  acting on  $\mathcal{H} \otimes \mathcal{H}$  such that

$$U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle \quad (2.29)$$

for every  $|\psi\rangle \in \mathcal{H}$ .

The proof is a short exercise in linearity. Suppose such a  $U$  existed, and consider two distinct states  $|\psi\rangle$  and  $|\phi\rangle$ . By hypothesis

$$U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle, \quad U(|\phi\rangle \otimes |0\rangle) = |\phi\rangle \otimes |\phi\rangle. \quad (2.30)$$

Taking the inner product of the two equations and exploiting the unitarity of  $U$  one obtains

$$\langle\psi|\phi\rangle = \langle\psi|\phi\rangle^2, \quad (2.31)$$

which is satisfied only if  $\langle\psi|\phi\rangle$  equals 0 or 1, that is, only if the two states are either identical or orthogonal. A universal cloning machine therefore cannot exist.

The consequences of this seemingly innocuous result are dramatic for quantum computation. The most natural classical strategy for error correction, triplicating each bit and taking a majority vote in order to establish which bit was faulty, is simply not available in the quantum setting, because we can never make three copies of an unknown qubit, and if we measure it we will obtain a deterministic value and will lose the probability linked to that unknown value. This is the starting point of this thesis: quantum computing needs to be able to store a qubit and the solutions found consist in encoding the logical information into the entangled state of many physical qubits in order to create redundancy; in this way errors can be detected and corrected without ever measuring, and so collapsing, the logical state [1, 17].

### 2.3.4 Quantum Parallelism

The expression **quantum parallelism** refers to the ability of a quantum computer to evaluate a function  $f$  simultaneously on a superposition of inputs. Given a classical reversible function  $f : \{0,1\}^n \rightarrow \{0,1\}^m$ , one defines the corresponding quantum oracle  $U_f$  as the unitary operator acting on  $n + m$  qubits as

$$U_f |x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle. \quad (2.32)$$

If the input register is prepared in the uniform superposition produced by a layer of Hadamard gates, applying  $U_f$  once yields

$$U_f \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \right) |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle, \quad (2.33)$$

that is, a single application of the oracle produces a state that contains information about **all** the values  $f(x)$  at once [17, 24].

This is, however, only half of the story. Direct measurement of the output register collapses the superposition and returns a single value of  $f$ , chosen uniformly at

random, so quantum parallelism by itself does not provide any computational advantage. The genuine speed-up of quantum algorithms comes from combining quantum parallelism with **interference**: by carefully designing the circuit so that the amplitudes associated with the desired outcomes interfere constructively while the others interfere destructively, one can extract global properties of  $f$  that no classical algorithm could compute in the same number of queries. The Deutsch-Jozsa algorithm [25], Shor's factoring algorithm [26] and Grover's search algorithm [27] are the most well-known examples of this design pattern.

## Chapter 3

# Surface Code

The surface code is currently the most studied approach to quantum error correction. The idea behind it is the same that makes classical computers reliable, that is redundancy: a single piece of information is stored many times, so that if one copy is corrupted the others still hold the correct value. In the quantum case copying is forbidden by the no-cloning theorem, so redundancy is obtained in a different way. A single logical qubit is encoded in the joint entangled state of many physical qubits, and the information is protected by the correlations between them rather than by any single qubit [3, 7].

The code works through three ingredients. The first are the **stabilizers**, local operators that check the parity of groups of qubits without reading their actual state. The second are the **ancillae**, auxiliary qubits that are measured at every cycle to evaluate the stabilizers; the collection of their outcomes is called the **syndrome**. The third is the **decoder**, a classical algorithm that reads the syndrome and decides which physical qubits to flip in order to undo the errors. The rest of this section describes each of these ingredients in turn, starting from the structure of the code itself.

### 3.1 Structure and parameters

A surface code is built on a two-dimensional square lattice of physical qubits of two kinds, arranged in an alternating pattern. The **data qubits** store the encoded information and are never measured during normal operation. The **ancilla qubits** sit between the data qubits and are used only to read out the stabilizers. Each ancilla is connected through nearest-neighbour links to the four data qubits that surround it, so the lattice forms a bipartite graph in which every link joins one data qubit to one ancilla [7].

Any error-correcting code is described by three numbers, written as a triple

$[[n, k, d]]$ , where  $n$  is the number of physical qubits,  $k$  is the number of logical qubits encoded, and  $d$  is the code distance. The distance is the minimum number of single-qubit operations needed to turn one valid encoded state into another, and it sets how many errors the code can tolerate. A surface code of distance  $d$  encodes  $k = 1$  logical qubit and can correct any error that affects fewer than  $(d-1)/2$  qubits. Increasing  $d$  means using more physical qubits but also gaining stronger protection: below a certain noise level, called the threshold, the chance of an uncorrectable error drops exponentially as  $d$  grows [7, 11]. For circuit-level depolarising noise this threshold is around 1% [7], a value that recent superconducting devices have started to reach [16].

Two variants of the lattice exist. In the original **unrotated** layout a distance  $d$  patch uses  $d^2 + (d-1)^2$  data qubits, while the **rotated** layout, obtained by cutting the same lattice along its diagonals, achieves the same distance with only  $n = d^2$  data qubits and  $d^2 - 1$  ancillae [28]. Since the two families offer comparable logical performance per physical qubit at the small distances relevant here [29], this thesis adopts the rotated layout throughout: at  $d = 5$ , the code consists of 25 data qubits and 24 ancillae, 12 per stabilizer type. Weight-four stabilizers occupy most of the space, while the two-weight stabilizers protect the boundaries, where a plaquette is cut in half; the two logical operators  $\bar{X}$  and  $\bar{Z}$  run as strings of  $d$  single-qubit Paulis connecting opposite boundaries.

### 3.1.1 Noise models

Three noise regimes of increasing realism are standard in the literature [11, 20], and this thesis uses all three names extensively:

- **Code capacity:** data qubits suffer errors (here: depolarizing noise, where each qubit independently undergoes  $X$ ,  $Y$  or  $Z$  with probability  $p/3$  each), and the syndrome is then measured **once, perfectly**. This is the simplest regime and the standard single-shot benchmark; it isolates the purely spatial part of the decoding problem.
- **Phenomenological:** data errors occur every round, and each syndrome measurement is itself flipped with probability  $p_{\text{ro}}$  (the **readout error**). The decoder now faces a temporal problem as well: a defect may be a lie.
- **Circuit level:** every gate, idle location and measurement in the stabilizer extraction circuit can fail. This is the most realistic regime; it is not addressed in this thesis and is left as future work.

Chapter 6–Chapter 8 first develop and evaluate the decoders in the code-capacity regime (Sections 8.2–8.4), then move to the phenomenological regime

Section 8.5, where the temporal structure of readout errors is precisely the margin our architecture is designed to exploit.

## 3.2 Encoding and the measurement cycle

The logical information is not stored in a single qubit but spread across the lattice through entanglement. The mechanism is the same as in the simplest quantum codes. In the three-qubit repetition code, for example, a state  $\alpha|0\rangle + \beta|1\rangle$  is encoded as  $\alpha|000\rangle + \beta|111\rangle$  by applying two CNOT gates, with the qubit to be protected as control and two fresh qubits as targets. The CNOTs entangle the ancillary qubits to the original state, and since entangling does not require measuring, redundancy is created without collapsing the state. The surface code uses the same principle on a much larger lattice.

Once the information is encoded, the code is kept alive by repeating a measurement cycle that reads the stabilizers. A single cycle can be broken into the following steps [7]:

1. All ancilla qubits are initialised in the state  $|0\rangle$ .
2. For the ancillae that read  $X$ -type stabilizers, a Hadamard gate is applied to rotate them into the  $|+\rangle$  state. The ancillae that read  $Z$ -type stabilizers are left as they are.
3. A sequence of CNOT gates entangles each ancilla with its four neighbouring data qubits. For an  $X$ -type stabilizer the ancilla is the control and the data qubits are the targets; for a  $Z$ -type stabilizer the roles are swapped. To avoid two gates acting on the same qubit at once, the four CNOTs are applied in four steps following the directions north, east, south and west. The order of these four CNOTs is relevant: it must be both fixed and carefully chosen, for two distinct reasons:
  - The correctness of the syndrome: neighbouring  $X$ -type and  $Z$ -type stabilizers share data qubits, so their measurement circuits overlap in time. With an unsuitable ordering the two circuits fail to commute, and the measured syndrome becomes non-deterministic even on an error-free encoded state, destroying the very information the decoder relies on. A consistent schedule such as north-east-south-west, applied identically to every ancilla, guarantees that all stabilizer circuits commute and that the syndrome is well defined;
  - How faults spread: the four gates act sequentially, so a fault that strikes the ancilla once some of the gates have already been applied propagates, through the CNOTs still to be executed, onto the corresponding data

qubits. A single physical fault can thus produce a weight-two correlated error on two data qubits, an event known as a **hook error**. A  $Z$  fault on a  $Z$ -type ancilla propagates backward, from target to control, onto its data qubits, whereas an  $X$  fault on an  $X$ -type ancilla propagates forward, from control to target; in both cases the outcome is a pair of same-type errors on two of the four neighbouring data qubits.

The danger of a hook error is that, from a single fault, it produces a two-step segment of an error chain, so a logical error can be completed with only about  $d/2$  faults instead of  $d$ , halving the effective code distance. The schedule must therefore orient hook errors perpendicular to the relevant logical direction, leaving them harmless and preserving the full distance  $d$ ;

The north-east-south-west schedule is the standard choice precisely because it satisfies both the commutativity requirement and this orientation constraint;

4. A second Hadamard gate is applied to the  $X$ -type ancillae, rotating them back to the computational basis.
5. Every ancilla is measured in the computational basis. The result is one classical bit per ancilla. The ancillae are then reset, ready for the next cycle.

The whole cycle entangles the ancillae with the data qubits just long enough to read the stabilizer values, then disentangles and measures them. Because the ancillae are separated from the data before measurement, reading them does not destroy the logical state.

### 3.3 Stabilizers

The stabilizers are the operators that make error detection possible. The surface code belongs to the **stabilizer formalism** [2]: the valid encoded states are exactly those that are left unchanged, with eigenvalue  $+1$ , by every stabilizer. There are two kinds, one for each type of error. An  $X$ -type stabilizer is the product of Pauli  $X$  operators on the four data qubits around an ancilla,

$$A_v = \prod_{i \in \partial v} X_i, \quad (3.1)$$

and a  $Z$ -type stabilizer is the product of Pauli  $Z$  operators around an ancilla,

$$B_f = \prod_{i \in \partial f} Z_i. \quad (3.2)$$

The two families are placed on the lattice so that each data qubit is checked by both  $X$ -type and  $Z$ -type stabilizers. This is what lets the code detect bit-flip

and phase-flip errors separately: an  $X$  error on a data qubit anticommutes with the neighbouring  $Z$ -type stabilizers and flips their value, while a  $Z$  error flips the neighbouring  $X$ -type stabilizers. All stabilizers commute with each other, so they can be measured at the same time without interfering [7, 30].

### 3.4 Ancillae and the syndrome

Stabilizers cannot be measured directly, because measuring a multi-qubit operator on the data would disturb the logical state. The ancillae solve this problem. Each stabilizer has one ancilla qubit associated with it, and the measurement cycle described above transfers the value of the stabilizer onto its ancilla, which is then measured. The classical bit obtained in this way is the value of that stabilizer for the current cycle.

When no error is present, every stabilizer is in its  $+1$  eigenstate and every ancilla returns 0. When an error hits the data qubits, the stabilizers that anticommute with it are flipped to  $-1$  and their ancillae return 1. The full set of these outcomes is the syndrome, and it is the only information the decoder receives [30, 7]. The syndrome tells where the boundaries of an error chain lie, since a chain of errors only flips the stabilizers at its two endpoints. It does not, on its own, identify which qubits were hit: many different error chains share the same endpoints and therefore produce the same syndrome.

A concrete example fixes the mechanism. Consider a single  $X$  error on a bulk data qubit. That qubit participates in exactly two  $Z$ -type stabilizers and two  $X$ -type ones; the error anticommutes with the former and commutes with the latter, so precisely two adjacent  $Z$ -checks flip to  $-1$ : the syndrome shows two neighbouring defects. Now extend the error to a chain of three collinear  $X$  errors. Every stabilizer located in the **interior** of the chain touches two errored qubits, and the two sign flips cancel; only the two stabilizers at the chain's **endpoints** see an odd number of errors and fire. The syndrome of a chain of any length is just its pair of endpoints, which is why decoding is a matching problem between defects, why the error's interior path is invisible, and why a chain that terminates on a boundary where no stabilizer waits beyond the edge produces a single lonely defect.

### 3.5 Homology classes

To sum up, the information of the stored logical qubit is distributed across the entire surface code, so that when random errors occur, they do not immediately destroy the logical state, because the logical information is redundant; the only

thing that happens is that the local parity of the data qubits changes, and so do the respective ancilla qubits' values and the resulting syndrome.

The central challenge of the decoding problem arises from the topological nature of these errors. A chain of errors only flips the stabilizers located at its two endpoints, as the intermediate stabilizers experience an even number of error actions and thus return a trivial outcome. Consequently, many distinct error chains can share the exact same syndrome. Two error chains that share the same syndrome fall into one of two cases:

- In the first case, they differ by a stabilizer operator. Since stabilizers leave the logical state untouched, these two chains have the identical physical effect, meaning that correcting either one successfully rectifies the error; they are physically equivalent;
- In the second case, they differ by a logical operator, a non-trivial chain that spans from one boundary of the lattice to the opposite one. Such a chain commutes with all stabilizers, leaving the syndrome completely empty, but it changes the value of the logical state initially stored in the surface code; this is the case that we want to absolutely avoid; [11, 20].

This splits the set of error chains with a given syndrome into **homology classes**: chains in the same class differ by a stabilizer and are equivalent, while chains in different classes differ by a logical operator and are not. The consequence is crucial for the execution of the decoder. It is not required to find the exact chain of errors, which is in general impossible, but only to identify the correct homology class. As long as the correction lands in the right class, the logical state is preserved, regardless of the individual physical qubits it flips [30, 20].

### 3.6 Degeneracy: minimum weight is not maximum likelihood

A given syndrome does not identify a unique error: it identifies an entire set of physically distinct but syndrome-equivalent corrections, partitioned into homology classes. The decoder's true task is therefore not to find the error, but to find the most probable **homology class**. The probability of a class is the sum of the probabilities of all corrections it contains, a quantity known as the maximum-likelihood decoding criterion [11].

MWPM approximates this result in two steps, with some loss of information. First, it replaces the sum over a class with its single largest term, returning the minimum-weight correction; when a class contains many corrections of comparable weight, their collective probability can exceed that of a lighter but lonelier class.

Second, by decoding the  $X$  and  $Z$  sectors on separate graphs, it treats a  $Y$  error, a simultaneous  $X$  and  $Z$  on the same qubit that has a probability  $p/3$  under depolarizing noise, as two independent events of joint probability  $\sim p^2/9$ , badly underestimating it.

The size of this mispricing is not subtle. At the operating point of this thesis,  $p = 0.05$  under depolarizing noise, a  $Y$  on a given qubit occurs with probability  $p/3 \approx 1.7 \times 10^{-2}$ ; treated as an independent  $X$  and  $Z$  coincidence, the same event is priced at  $(p/3)^2 \approx 2.8 \times 10^{-4}$ , an underestimate by a factor of  $\sim 60$ . Since matching weights are log-probabilities, MWPM systematically prefers explanations that split every  $Y$  into two unrelated single-sector errors, even when the joint explanation is far more likely.

A decoder that reasons **jointly** over both sectors and over whole classes can therefore beat MWPM without any appeal to learning; the gap is mathematical, not empirical.

This observation fixes the strategy of the thesis: any learned decoder that merely reproduces minimum-weight behaviour cannot beat MWPM; a learned decoder that captures class degeneracy and  $X$  -  $Z$  correlations can. The supervised joint-class decoder of Chapter 7 is built exactly on this margin, and Section 8.2 verifies the size of the gap exactly at  $d = 3$ .

### 3.7 The role of the decoder

The decoder is the part of the system that turns the syndrome into a correction. Its input is the syndrome, possibly a whole syndrome volume, and its output is a set of Pauli operators to apply to the data qubits. A correction succeeds when  $E \cdot R$  is a product of stabilizers, and fails when it is a logical operator. The quality of a decoder is measured by how often it succeeds, that is by its logical error rate at a given physical noise level.

The task is hard for two reasons that come directly from the discussion above. The number of possible error chains grows exponentially with the code distance, so the decoder cannot simply list them and pick the most likely. And the syndrome is an incomplete and noisy view of a state that can never be observed directly, so the decoder must reason under uncertainty. These two facts shape every decoding algorithm, from the Minimum Weight Perfect Matching used as a baseline in this work to the learning-based decoder developed in the following chapters.

### 3.8 The Logical Error

In the rotated surface code variant used in this work, the lattice is oriented diagonally so that  $X$ -type stabilizers span the code horizontally and  $Z$ -type stabilizers span it

vertically.

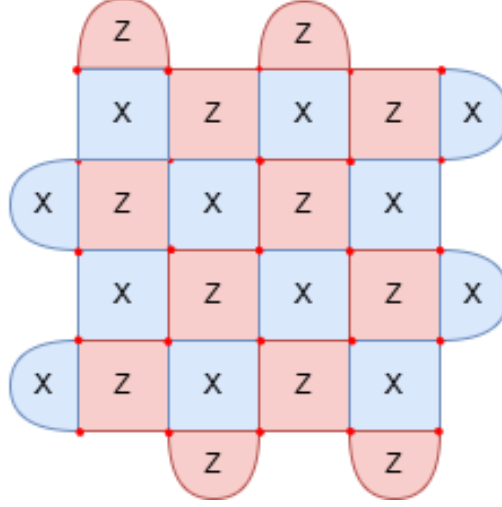


**Figure 3.1:** The rotated surface code at distance  $d = 5$ . Data qubits sit on the vertices;  $X$ -type and  $Z$ -type stabilizer plaquettes alternate in a checkerboard pattern. The two logical operators  $\bar{X}$  and  $\bar{Z}$  run as horizontal and vertical strings of single-qubit operators connecting opposite boundaries.

Each stabilizer is measured indirectly through a dedicated ancilla qubit. A sequence of CNOT gates entangles the ancilla with each of the data qubits on the corners of its plaquette: for an  $X$ -type stabilizer the ancilla acts as the control and the data qubits as targets; for a  $Z$ -type stabilizer the roles are reversed. The ancilla is then measured in the computational basis. The outcome is 0 if the number of errors of the relevant type among the corner qubits is even, including zero, and 1 if that number is odd. Two errors on adjacent qubits of the same stabilizer therefore cancel out in the measurement: the stabilizer does not fire, and the pair of errors is invisible to that check alone.

A logical error occurs when a connected chain of physical errors of the same Pauli type stretches from one boundary of the lattice to the opposite one. Looking at Figure 3.1: a chain of  $X$  errors running from the left boundary to the right boundary is a logical  $\bar{X}$  error; a chain of  $Z$  errors running from the top boundary to the bottom boundary is a logical  $\bar{Z}$  error. These chains are the only undetectable error patterns that also alter the logical state: closed loops (products of stabilizers) are equally invisible to the syndrome, but act trivially on the encoded qubit.

To see why, consider  $X$  errors on a row of data qubits in the interior of the lattice. Each isolated  $X$  error anticommutes with the two  $Z$ -type stabilizers that share that qubit as a corner, so it lights up exactly two adjacent syndrome bits. Two neighboring  $X$  errors share one  $Z$ -stabilizer: that stabilizer now has two

anticommuting terms that cancel, so it returns 0, while the two outer stabilizers each fire once, so the syndrome shows two 1s with a 0 between them. Extending the chain step by step toward the boundaries, each new error converts one firing stabilizer into a silent one and activates the next one along the chain. At the boundaries, however, the stabilizers become weight-two plaquettes that touch only two data qubits: a chain error at the boundary has one neighbor stabilizer that fires and one boundary edge with no stabilizer beyond it. Once the chain reaches both the left and right boundaries, every stabilizer along the path has been silenced, and the syndrome is all zero, so all the errors now are completely undetectable.

The decoder's task is therefore not simply to correct random errors before they form such chains; it must also ensure that its own corrections do not complete one. A syndrome pattern of two firing stabilizers with a gap between them is consistent with two different physical scenarios: a chain of errors in the interior, or two separate shorter chains near the boundaries. In the first case the correct correction bridges the interior gap; in the second case that same correction would form, together with the boundary errors already present, a complete chain that crosses the lattice. The decoder cannot observe the physical errors directly, because it only sees the syndrome, so it must infer which scenario is more likely and act accordingly. Choosing wrong means applying a correction that, combined with the true error, forms a logical operator: the syndrome goes to zero, the code space is restored, but the logical qubit has been silently flipped. This is the logical error, and its rate, the fraction of decoding attempts that end this way, is the primary metric by which every decoder in this thesis is evaluated.

## Chapter 4

# Decoding as a Sequential Decision Problem

The workflow previously described is not the whole process; it is only one cycle. The code is measured over and over, and at each cycle the decoder receives a new syndrome and has to decide what to do, knowing that its choice will affect the syndromes it sees later. Decoding is therefore not a one-shot task but a sequence of decisions taken under uncertainty, where each choice matters for the future. This chapter introduces the framework built for this kind of problem and shows how decoding fits into it.

### 4.1 Reinforcement Learning

Reinforcement learning is a branch of artificial intelligence in which a computer system called the **agent** learns to perform a task autonomously. The model proceeds by trial and error, interacting with the **environment**, and receives a **reward** for each good move while incurring a penalty for each bad one. The aim is to find the strategy that maximises the total reward over the long term, so that by setting the right reward the desired behaviour of the agent is obtained.

The decisions taken during training can affect what happens later, so when planning it is important to consider not only the immediate benefit of a decision but also its longer-term effects. A reward function that captures the goal correctly is therefore essential, so that the agent can learn that a move which does not look optimal at first may in fact lead to a higher long-term reward.

At the start of the simulation the agent knows nothing, and it only gets feedback through the rewards of the moves it makes. To discover which moves are best, it must first explore many actions and get to know the problem before trying to solve it. When possible, it can also generalise from previous experience to new tasks or

conditions, which is useful given the high dimensionality of the state space.

### 4.1.1 Reinforcement Learning Terminology

We now introduce the terminology used in the rest of the chapter.

**Reward** A reward  $R_t$  is a scalar feedback signal that measures how well the agent is doing at step  $t$ . The higher the cumulative reward, the better the performance.

**Agent and Environment** At each step  $t$  the agent executes an action  $A_t$ , then receives an observation  $O_t$  and a reward  $R_t$ ; the environment receives the action  $A_t$  and emits the next observation  $O_{t+1}$  and reward  $R_{t+1}$ . The index  $t$  increases at every environment step.

**History and State** The history is the sequence of observations, actions and rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t, \quad (4.1)$$

and it determines what happens next. The state  $S_t$  is the information used to decide what happens next, and it is a function of the history,

$$S_t = f(H_t). \quad (4.2)$$

The environment state  $S_t^e$  is the internal state of the environment at time  $t$ . It is usually not visible to the agent, and even when it is, it may contain irrelevant information. The agent state  $S_t^a$  is the internal representation held by the agent, and it is the information that the reinforcement learning algorithm actually uses.

**Markov State** A state has the Markov property when it contains all the information needed to predict the next state:

$$P(S_{t+1} \mid S_1, \dots, S_t) = P(S_{t+1} \mid S_t). \quad (4.3)$$

The future is then independent of the past given the present, because the current state is a sufficient summary of the history. This is the property that keeps the problem tractable, since the agent never has to remember how it got where it is.

**Fully and Partially Observable Environments** From the agent's point of view the environment can be fully or partially observable. Under full observability the agent observes the environment state directly,

$$O_t = S_t^a = S_t^e, \quad (4.4)$$

and the problem is formally a Markov decision process. Under partial observability the agent observes the environment only indirectly, so  $S_t^a \neq S_t^e$ , and it must build its own state representation from the history or from a belief about the environment state. This second case is the one relevant to decoding, and we treat it in detail below.

## 4.2 Markov decision processes

A Markov decision process is the interaction between an agent and an environment over discrete time steps, where at each step the environment is in some Markov state, the agent picks an action, the environment moves to a new Markov state and returns a reward [31]. Formally, an MDP is a tuple

$$(S, A, R, K_T, \gamma), \quad (4.5)$$

where  $S$  is the set of states,  $A$  the set of actions,  $K_T(s' | s, a)$  the probability of reaching state  $s'$  after taking action  $a$  in state  $s$ ,  $R(s, a)$  the reward assigned to a state, and  $\gamma \in [0, 1)$  a discount factor that controls how much future rewards count.

The behaviour of the agent is described by a policy  $\pi(a | s)$ , a rule that assigns to each state a probability over actions. The goal is to find a policy that maximises the expected return, the discounted sum of future rewards

$$G_t = \sum_{k \geq 0} \gamma^k r_{t+k+1}. \quad (4.6)$$

The discount factor makes rewards that arrive sooner count more than those far ahead, and keeps the sum finite. Optimising  $G_t$  rather than the immediate reward is what lets the agent plan: it can accept a small loss now if it leads to a better position later.

Two functions summarize how good a policy is. The **State-Value function** is the expected return starting from a state  $s$  and following a policy  $\pi$ ,

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s], \quad (4.7)$$

and the **Action-Value function**, which also conditions on the first action,

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \quad (4.8)$$

Both obey a self-consistency condition that is the engine of every RL algorithm: the value of a state equals the immediate reward plus the discounted value of what follows. For the optimal policy this becomes the Bellman optimality equation,

$$Q^*(s, a) = \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a \right], \quad (4.9)$$

whose fixed point defines  $Q^*$ , from which the optimal policy is recovered by acting greedily,  $\pi^*(s) = \arg \max_a Q^*(s, a)$  [31].

Learning  $Q^*$  from experience, however, requires visiting states and actions beyond those the current policy prefers. The simplest mechanism, used by the value-based agent of Chapter 7, is the  $\varepsilon$ -greedy policy: with probability  $1 - \varepsilon$  act greedily with respect to the current estimate, with probability  $\varepsilon$  pick a uniformly random action, with  $\varepsilon$  annealed from  $\approx 1$  toward a small floor over training, so that early experience is diverse and late behaviour is exploitative [31].

### 4.3 Partially Observable Markov Decision Process

The MDP assumes the agent can see the state of the environment. As noted above, this is often not the case: the agent only receives partial, indirect information about a state it cannot observe directly. This is the setting of the partially observable Markov decision process (POMDP) [32].

A POMDP adds two ingredients to the MDP: a set of observations  $O$  and an emission kernel  $K_O(o | s)$ , the probability of seeing observation  $o$  when the true state is  $s$ . The full tuple is

$$(S, A, O, K_T, K_O, R, \gamma). \quad (4.10)$$

Since the agent cannot see the state, it cannot act on it directly. Instead it keeps a **belief**  $b$ , a probability distribution over the possible hidden states that summarises everything it has learned from past observations. After each action and observation the belief is updated, first through the transition kernel and then through Bayes' rule on the new observation. The policy now maps beliefs to actions,  $\pi(a | b)$ , and since the reward of a hidden state cannot be read either, the agent works with its expected value over the belief,

$$r(b) = \sum_s b(s) R(s). \quad (4.11)$$

Apart from replacing the state with the belief, the loop is the same as before: observe, update the belief, act, repeat.

### 4.4 Why a POMDP fits decoding

Casting decoding as a sequential decision problem brings two main advantages, plus a practical bonus.

The first is that it matches what actually happens. The surface code is not measured once but continuously, and each round of corrections changes the state

that the next round will measure. A decoder that reasons one syndrome at a time, like Minimum Weight Perfect Matching, ignores this structure. Treating decoding as a sequence of decisions lets the decoder use the history and plan across many cycles.

The second is the change of objective. A one-shot decoder tries to undo the most likely error right now. But the real goal is not to be right at every single cycle; it is to keep the logical information intact for as long as possible. Optimising the expected return rather than the immediate correction captures exactly this: the decoder is rewarded for preserving the logical state over time, and is free to make a locally suboptimal move when it pays off later. This long-horizon view is awkward to express in a static decoder but is the natural language of MDPs. As a bonus, once decoding is written as a POMDP, the whole set of methods for solving such problems becomes available, and the decoder can be trained rather than hand-designed [30].

## 4.5 Mapping the surface code onto a POMDP

It remains to connect the framework to the actual problem. The natural candidate for the state is the error configuration of the lattice, the set of Pauli errors currently affecting the qubits. The actions are the recovery operations the decoder can apply, and the reward is tied to whether the logical state has been preserved.

The key point is that this state cannot be observed. The decoder never sees the errors directly; all it receives are the syndromes, which are a partial and noisy function of the error configuration. This is exactly why decoding is a POMDP and not an MDP. If we tried to use the syndrome itself as the state of an ordinary MDP, the Markov property would break: the syndrome does not carry enough information to predict its own future, since the same syndrome can come from different underlying errors that evolve differently. Only the hidden error state is Markovian, and because it is not observable, the correct description is a partially observable one.

The surface code cycle fits this description almost by construction. The error process plays the role of the hidden state and its transitions, while the syndrome measurements are the emissions: noisy readings that depend on the hidden state without revealing it. Sometimes only **detection events** are passed to the decoder, not the whole syndrome but a string with a 1 wherever the value changed from the previous round, the detector events, which expose only the difference between consecutive measurements.

In short, the measurement cycle is a hidden Markov model, and the decoder sitting on top of it, choosing corrections from the syndromes, turns that model into a POMDP. This also settles a subtlety: the decoder must work without

knowing the logical information it protects, so the state and its dynamics are defined independently of the encoded qubit. The belief it maintains is a belief about errors, not about the message, which is precisely what we want.

This leaves open the question of how to actually solve the resulting POMDP. In principle, with the transition and emission kernels known, an exact solution exists. In practice it is out of reach, because the state and action spaces grow far too fast with the code distance for any exact method. The way forward is to approximate the solution with reinforcement learning, training a policy on simulated cycles drawn from a known noise model.

Table 4.1 makes the correspondence explicit, item by item; every component named here has a concrete software counterpart in Chapter 6.

| POMDP object                 | Surface-code instantiation                                                                                                                          |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>State</b> $s \in S$       | The hidden Pauli error configuration on the $n$ data qubits (with the tracked correction frame, the residual error), unobservable by construction.  |
| <b>Action</b> $a \in A$      | A binary correction mask: the set of $X/Z$ flips applied to the tracking frame at this step.                                                        |
| <b>Observation</b> $o \in O$ | The measured syndrome (or its detector events), possibly corrupted by readout noise.                                                                |
| <b>Transition</b> $K_T$      | The noise process: each round, fresh depolarizing errors land on the data qubits, composing with the residual state.                                |
| <b>Emission</b> $K_O$        | The stabilizer measurement: a deterministic parity map of the hidden state, composed with readout flips of probability $p_{\text{ro}}$ .            |
| <b>Reward</b> $R$            | Terminal: determined by the homology class of error $\oplus$ correction (logical success or failure); optional shaping on residual syndrome weight. |
| <b>Belief</b> $b$            | Approximated by the factorized local marginals of Section 6.2, updated by Bayes on each observation and deterministically on each action.           |
| <b>Discount</b> $\gamma$     | Near 1: the objective is survival of the logical information over the whole episode, not greedy per-round cleanup.                                  |

**Table 4.1:** Instantiation of the POMDP tuple on the surface code.

## 4.6 Foundational Learning Paradigms

Once a problem is formulated as a POMDP, several algorithmic families can search for a good policy, and the choice among them is not neutral: each family fixes an objective, and, as the next section will argue, the objective determines the ceiling before any training begins. This thesis deliberately employs three paradigms, chosen because they occupy three distinct positions with respect to that ceiling:

- **Imitation learning** trains the policy to reproduce MWPM: it inherits both the expert’s competence and, by construction, its limits, and serves in this work as the bootstrap phase of the main decoder;
- **Value-based reinforcement learning** receives no expert guidance: its only signal is a per-step reward for reducing the number of ones in the syndrome. The agent that makes only atomic actions, therefore, has no way to imitate MWPM, but every incentive to reinvent it. This makes it the cleanest possible test of the attractor argument: if the agent converges to MWPM-level performance despite training independently and from scratch, the cause cannot be imitation, it must be the objective itself;
- **Supervised global classification** abandons sequential action selection and estimates the posterior over logical classes: it is the one formulation whose optimum is maximum-likelihood decoding rather than minimum-weight decoding, and therefore the one with a mathematical margin over the expert in a certain context;

The experimental chapters test all three against the same baseline; the theory below predicts, before any experiment, which ones can win.

### 4.6.1 Imitation Learning’s policy

Imitation Learning (IL) addresses the optimization of a policy by leveraging a set of demonstrations provided by an expert supervisor or an algorithmic oracle  $\pi^*$ . Instead of discovering an optimal behavioral pattern via autonomous environmental interaction, the learning process is cast as a supervised learning task over sequential trajectories.

Let  $\mathcal{D} = \{(s_i, a_i^*)\}_{i=1}^N$  be a dataset composed of state configurations  $s_i$  and the corresponding actions  $a_i^*$  selected by the expert. The objective of the learning agent is to parameterize a policy  $\pi_\theta(a \mid s)$  that maximizes the likelihood of reproducing the expert’s decisions. This is formally achieved by minimizing the empirical categorical cross-entropy loss over the dataset:

$$\mathcal{L}_{\text{IL}}(\theta) = -\mathbb{E}_{(s, a^*) \sim \mathcal{D}} \left[ \sum_{a \in \mathcal{A}} \mathbb{I}(a = a^*) \log \pi_\theta(a \mid s) \right], \quad (4.12)$$

where  $\mathbb{I}$  represents the indicator function.

A central theoretical challenge in sequential imitation learning is the **compounding error** phenomenon. The standard supervised formulation assumes that the states encountered during execution are independent and identically distributed, matching the expert’s state distribution ( $P_{\pi_\theta}(s) \approx P_{\pi^*}(s)$ ). However, in sequential environments, a single sub-optimal action taken by the parameterized policy shifts the system into a state outside the training distribution. Once off-track, the agent’s probability of executing further errors increases exponentially, leading to a catastrophic accumulation of drift over long horizons.

## 4.6.2 Deep Q-Learning and Double DQN

When an expert oracle is unavailable or when the goal is to discover policies that transcend existing heuristics, model-free Reinforcement Learning is employed. In value-based methods, the objective is to approximate the action-value function  $Q^\pi(s, a)$ , which describes the expected cumulative discounted return obtained by executing action  $a$  in state  $s$  and following policy  $\pi$ . The unique optimal action-value function  $Q^*(s, a)$  satisfies the fundamental Bellman optimality equation (4.9), which enforces self-consistency via dynamic programming.

In environments with high-dimensional or continuous state spaces, tabular tracking of  $Q^*(s, a)$  becomes intractable. Deep Q-Networks (DQN) overcome this restriction by approximating the function using a deep neural network  $Q(s, a; \theta)$ .

Standard DQN updates its weights by minimizing a mean-squared Bellman error. However, the presence of the maximum operator in the target formulation ( $\max_{a'} Q(s', a'; \theta)$ ) introduces a systematic **maximization bias**, causing the network to overstate value estimates due to ambient noise or function approximation errors. To mitigate this structural bias, the **Double DQN (DDQN)** architecture decouples the selection of the greedy action from its downstream evaluation. This is achieved by maintaining two distinct weight configurations: an online network  $\theta$  and a target network  $\theta^-$ . The loss function is expressed as:

$$\mathcal{L}_{\text{DDQN}}(\theta) = \mathbb{E}_{(s,a,r,s')} \left[ \left( r + \gamma Q \left( s', \arg \max_{a'} Q(s', a'; \theta); \theta^- \right) - Q(s, a; \theta) \right)^2 \right], \quad (4.13)$$

where the target parameters  $\theta^-$  are slowly updated over time to maintain optimization stability.

## 4.6.3 Supervised Classification

In specific configurations, a sequential decision problem can be mathematically reframed or simplified into a single-shot statistical inference task. Instead of treating the environment as an evolving cascade of multi-step transitions, the problem is

modeled as predicting a hidden global state or property  $\mathcal{C}$  directly from an aggregate observation vector  $x$ .

Under this supervised classification paradigm, the machine learning model functions as a conditional probability estimator. Given a discrete set of target classes  $\mathcal{C} \in \{1, 2, \dots, K\}$ , a neural network maps the input features to a score vector, which is subsequently normalized via the softmax activation function to represent the posterior probability distribution:

$$P(\mathcal{C} = c \mid x; \theta) = \frac{\exp(f_c(x; \theta))}{\sum_{j=1}^K \exp(f_j(x; \theta))}. \quad (4.14)$$

The parameters  $\theta$  are optimized end-to-end against a dataset of labeled instances by minimizing the categorical cross-entropy loss function:

$$\mathcal{L}_{\text{CE}}(\theta) = -\frac{1}{B} \sum_{i=1}^B \sum_{c=1}^K \mathbb{I}(\mathcal{C}_i = c) \log P(\mathcal{C}_i = c \mid x_i; \theta), \quad (4.15)$$

where  $B$  is the batch size.

Unlike step-by-step reinforcement learning agents that rely on localized reward signals, a global classification framework optimizes for **Maximum Likelihood Estimation (MLE)** over the entire global structure. This allows the model to process simultaneous multi-channel inputs and capture long-range statistical correlations that are structurally invisible to purely sequential, localized tracking heuristics.

## 4.7 The role of Exploration

A critical component of this Reinforcement Learning framework is managing the balance between exploitation and exploration. Exploitation occurs when the agent relies entirely on its current knowledge, selecting the action with the highest predicted value to maximize its immediate performance. However, if the agent strictly exploits its early, imperfect knowledge, it risks getting stuck in suboptimal strategies and failing to discover better, more complex solutions. To prevent this, the system incorporates an exploration mechanism, where the agent deliberately takes random actions regardless of the network's predictions. This allows the agent to experience new states and potentially uncover more effective correction pathways. In practice, this balance is managed through a dynamic, decaying probability strategy. At the beginning of the training phase, the agent acts almost entirely randomly to gather a diverse set of experiences. As the training progresses and the network's predictions become more accurate, the probability of taking random actions is gradually reduced. By the end of the optimization process, the model relies almost exclusively on its learned policy, maintaining only a minimal margin of exploration to preserve a degree of flexibility.

## 4.8 The minimum-weight attractor

Framing decoding as a sequential decision problem exposes a subtle obstacle. Consider any policy trained, by imitation or by reward, to drive the residual syndrome to zero. At every step, the correction that flips the fewest qubits compatible with the observed defects is also the one that injects the least **residual dirt** into the lattice: every unnecessary flip is itself a new physical error that future rounds must pay for. In a regime without measurement noise, greedily neutralizing the syndrome along the lightest path and so minimizing the residual dirt is therefore locally optimal at every step, and a policy that does exactly this **is** MWPM. We introduce the term minimum-weight attractor to name this phenomenon, that has two main consequences. First, MWPM is a strong local attractor of the training dynamics: an agent initialized by imitation of MWPM sits at a point where every small policy perturbation increases the amount of injected dirt before it can possibly pay off, so gradient-based refinement either stays put or degrades. Second, extra capacity does not help: the wall is the **method**, not the network size, a claim we verify empirically in Section 8.4 by doubling the hidden width with no change in logical error rate.

Escaping the attractor requires changing what is optimized, not how hard. The two exits identified in Chapter 3 are **predicting the logical class directly**, so that degeneracy and  $Y$  correlations enter the objective, and moving to a regime where minimum weight is no longer locally optimal because some defects are lies, so the regime with readout errors, where temporal reasoning has value.

## Chapter 5

# Introduction to Graph Neural Networks

Graph Neural Networks (GNNs) represent a major step forward in deep learning, designed to handle irregular data structures that traditional neural networks struggle with. Standard architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) excel at grid-like or sequential data, such as images and text, while many real-world systems are based on connections and are best represented as graphs. A graph consists of nodes, representing individual entities, and edges, which show the relationships or interactions between them. By operating directly on this structure, GNNs successfully capture patterns in irregular data that grid-based models cannot naturally represent.

This chapter builds the GNN toolbox of the thesis from the ground up. It starts from why a graph is the right representation for the surface code and how a generic message-passing layer works; it then surveys the main layer families in use today and motivates the choice made in this work, asks what message passing can and cannot compute, and generalizes the machinery to graphs with several node types, which is where factor graphs enter. The remaining sections cover the structural properties that make GNNs a good match for decoding, the pathologies of deep GNNs and their standard remedies, and the software framework in which all of this is implemented. The final section summarizes the specific combination of these ingredients adopted by the decoder of the next chapter.

### 5.1 Advantages of using a graph

We can represent the problem as a graph. The surface code is naturally described by a graph in which the nodes are the data qubits and the stabilisers, and an edge connects a qubit to a stabiliser whenever that qubit takes part in that stabiliser's

measurement. The syndrome, the marginals, and the corrections all live on this graph.

A plain multilayer perceptron would ignore this structure entirely: it would receive a flat list of numbers and would have to rediscover, from scratch, which qubit is next to which stabiliser. A convolutional network does better on regular grids, and it has been used for surface-code decoding, but it fits awkwardly with the irregular boundaries of the lattice and with the fact that qubits and stabilisers are objects of different nature. What we want is a model whose internal structure mirrors the structure of the problem, so that the relations between qubits and stabilisers are built in rather than learned.

There is also a quantitative version of this argument. A fully connected network on  $N$  qubits must learn  $O(N^2)$  pairwise relations from data. Most of these connections are irrelevant because physical errors and stabilizer checks are local. In contrast, a graph model starts with the correct sparsity pattern. It uses a Tanner-like bipartite structure [33], which means the graph is split into two types of nodes: one for qubits and one for parity checks, with edges only connecting a check to the qubits it measures. By using this structure, the model spends its entire capacity on the relations that physically exist. This prior knowledge encoded in the graph is exactly the parity-check structure of the code, which is known perfectly from the start and does not need to be learned at all.

## 5.2 How a GNN works

A GNN operates directly on a graph whose nodes carry feature vectors. Let  $h_v$  be the feature vector of node  $v$ , and let  $\mathcal{N}(v)$  be the set of its neighbours. The model works by **message passing**: each node looks at its neighbours, collects information from them, and uses it to update its own state [34, 35]. A single layer of message passing can be written as

$$h_v^{(l+1)} = \phi \left( h_v^{(l)}, \bigoplus_{u \in \mathcal{N}(v)} \psi(h_v^{(l)}, h_u^{(l)}, e_{vu}) \right), \quad (5.1)$$

where  $\psi$  is the **message function** that builds a message from a neighbour  $u$  (optionally using the feature  $e_{vu}$  of the edge between them),  $\bigoplus$  is an aggregation that combines all incoming messages, and  $\phi$  is the **update function** that produces the new state of the node. The aggregation is the crucial part: it must be a permutation invariant operation, such as a sum, a mean or a maximum, because a node has no intrinsic ordering of its neighbours and the result must not depend on the order in which they happen to be listed.

A toy computation makes the abstraction concrete. Take a path graph of three

nodes  $u - v - w$  with scalar features  $h_u = 1$ ,  $h_v = 0$ ,  $h_w = 2$ , sum aggregation, message function  $\psi(h_v, h_u) = Wh_u$  with  $W = 0.5$ , and update  $\phi(h_v, m) = \text{ReLU}(h_v + m)$ . The central node receives  $m_v = Wh_u + Wh_w = 0.5 \cdot 1 + 0.5 \cdot 2 = 1.5$  and updates to  $h'_v = \text{ReLU}(0 + 1.5) = 1.5$ ; the endpoint  $u$ , with the single neighbour  $v$ , receives  $m_u = 0.5 \cdot 0 = 0$  and keeps  $h'_u = 1$ . Two properties are visible in these two lines of arithmetic. First, after one layer,  $v$  knows about both its neighbours but  $u$  knows nothing about  $w$ : information travels one hop per layer. Second, the same  $W$  served every edge: if we had relabelled the nodes, the outputs would have been relabelled identically, which is a permutation equivariance.

One layer lets each node gather information from its immediate neighbours. Stacking  $L$  layers lets information travel up to  $L$  steps across the graph, so that after a few layers each node has aggregated a picture of a whole region around it. This is the graph counterpart of the receptive field of a convolutional network. Once the final node states are computed, a **readout** step turns them into the desired output. When the prediction is per-node, as in our case where we want one score per qubit, the readout is simply a small network applied independently to each node state; when a single graph-level output is needed, the node states are first pooled together, by a sum, a mean, or an attention-weighted combination, and the pooled vector is passed to a classifier. Both readout styles are used in this thesis: the sequential decoder of Chapter 6 scores individual qubits, while the supervised classifier of Chapter 7 pools the whole graph into a single four-way logical decision.

### 5.3 A tour of message-passing layers

Equation (5.1) is a template, not an algorithm: every concrete GNN layer is a particular choice of  $\psi$ ,  $\oplus$  and  $\phi$ . Since the choice fixes what the network can express and how much it costs, it is worth reviewing the main families before stating which one this thesis uses. The mechanism to keep in mind is how does the layer decide **how much each neighbour matters**.

**Graph Convolutional Networks (GCN)** In a GCN layer, the weighting is determined solely by the graph topology. A GCN layer [36] computes

$$h_v^{(l+1)} = \sigma \left( \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\deg(u) \deg(v)}} W h_u^{(l)} \right), \quad (5.2)$$

a degree-normalized average of the neighbourhood passed through a single shared linear map. The weighting is fixed by the node degrees: every neighbour of a given node contributes equally. This makes GCN cheap and stable, but blind to the

**content** of the neighbours; a firing stabilizer and a silent one, if equally connected, are averaged with equal weight, which is exactly what a decoder must not do.

**GraphSAGE** GraphSAGE [37] improves upon GCN by decoupling the node’s own state from the neighbourhood summary. A GraphSAGE layer computes:

$$h_v^{(l+1)} = \sigma\left(W_1 h_v^{(l)} + W_2 \text{mean}_{u \in \mathcal{N}(v)} h_u^{(l)}\right), \quad (5.3)$$

While preserving the node’s own state prevents identity loss, the uniform aggregation remains a major drawback for decoding. Because every neighbour is averaged with equal weight, the model still cannot prioritize active, firing stabilizers over silent ones, making it sub-optimal for locating localized quantum errors.

**Graph Attention Networks (GAT)** Graph Attention Networks [38] introduce the mechanism of attention. While earlier models like GCN and GraphSAGE use the graph geometry to enforce rigid, uniform weights on all neighbours, attention allows the network to dynamically scale these weights based on the actual data content inside the nodes. The topology still defines which nodes are connected, but the model learns how much to listen to each specific neighbour. Each edge receives a learned attention coefficient computed as:

$$\alpha_{vu} = \text{softmax}_{u \in \mathcal{N}(v)}\left(\text{LeakyReLU}\left(a^\top [W h_v \parallel W h_u]\right)\right) \quad (5.4)$$

In this formula, the states of the central node  $h_v$  and its neighbour  $h_u$  are first projected via a shared weight matrix  $W$  and concatenated ( $\parallel$ ). A learnable vector  $a^\top$  scores their relationship through a LeakyReLU non-linearity. Finally, the softmax function normalizes these scores across all neighbours  $u \in \mathcal{N}(v)$ , translating them into relative percentage weights  $\alpha_{vu}$  that sum to one.

To increase learning capacity, multiple **attention heads** run in parallel. Instead of relying on a single judge, the layer employs a team of independent attention mechanisms that analyze the same neighborhood simultaneously. Each head uses its own unique parameters ( $W$  and  $a^\top$ ), allowing them to specialize in different features. For instance, one head can focus entirely on tracking active, firing stabilizers (defect signals), while another head maps the surrounding quiet regions to maintain context. Their individual outputs are then concatenated together. For decoding, this is the decisive capability: attention transforms the rigid geometric graph into a flexible filter, learning to heavily prioritize critical anomalies while ignoring background silence.

**Graph Isomorphism Networks (GIN)** GIN [39] focuses on maximizing the expressive power of GNNs by making the neighbor aggregation function **injective**. Mathematically, an injective function ensures that two different sets of incoming

inputs will always produce two distinct outputs. To achieve this, GIN uses a sum aggregator rather than a mean, followed by a Multi-Layer Perceptron (MLP):

$$h_v^{(l+1)} = \text{MLP} \left( (1 + \epsilon) h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l)} \right). \quad (5.5)$$

In this formulation,  $\epsilon$  is a learnable parameter or a fixed value that weights the central node’s own state  $h_v$ , while the states of all neighbors  $h_u$  are directly summed.

The distinction between a sum and a mean aggregator is highly practical for quantum error decoding. A mean aggregator only captures the proportion of features, meaning it erases critical count information. For example, a stabilizer check surrounded by two errored qubits and another surrounded by only one errored qubit would yield identical averaged values if the ratio of quiet to active neighbors matches. On the contrary, a sum aggregator preserves the exact number of active signals. Since identifying the precise number of defects is essential for locating quantum errors, GIN’s ability to maintain these distinct defect counts makes it significantly more capable of distinguishing complex error patterns than mean-based models.

**Transformer Convolutions (TransformerConv)** The layer adopted in this work, **TransformerConv** [40], adapts the scaled dot-product attention mechanism of Transformers[41] to graph-structured data. This phrase means that TransformerConv bridges the gap between natural language processing and geometric deep learning. In standard Transformers, the scaled dot-product attention computes relationships between words in a sentence by projecting each word into three vectors: a **Query**, what the word is looking for, a **Key**, what the word contains, and a **Value**, the actual content. The network then calculates how much attention word  $A$  should pay to word  $B$  by taking the dot-product (matrix multiplication) of  $A$ ’s Query and  $B$ ’s Key, dividing it by a scaling factor  $\sqrt{d_k}$  to ensure stable training gradients.

Adapting this mechanism to graph-structured data means that instead of comparing words in a linear text, the attention is computed locally between a central node and its physical neighbours in the graph. In the context of the surface code, the central stabilizer acts as the Query, asking for error information. The surrounding data qubits act as Keys and Values, offering their error states. By multiplying the Query vector by the Key vector of a neighbour, the model computes a custom attention weight. This mathematical adaptation allows the network to calculate the precise correlation between connected components, transforming a static lattice into a dynamic, text-like sequence of quantum information exchanges. Unlike GAT, which computes attention using a rigid concatenation and a single weight vector, TransformerConv defines a much more expressive, bilinear attention score where

edge features directly modulate the interaction. The update rule is formulated as:

$$h_v^{(l+1)} = W_1 h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \alpha_{vu} (W_2 h_u^{(l)} + W_e e_{vu}), \quad (5.6)$$

where the dynamic attention coefficients  $\alpha_{vu}$  are computed via:

$$\alpha_{vu} = \text{softmax}_{u \in \mathcal{N}(v)} \left( \frac{(W_q h_v)^\top (W_k h_u + W_e e_{vu})}{\sqrt{d_k}} \right). \quad (5.7)$$

In this mechanism, the central node maps its state into a **Query** vector ( $W_q h_v$ ), while each neighbour  $u$  computes a **Key** vector ( $W_k h_u$ ). Crucially, the edge feature  $e_{vu}$  is projected by  $W_e$  and injected directly into the Key and the message computation. The dot-product between the Query and the modified Key is scaled by  $\sqrt{d_k}$  to prevent gradient saturation, and then normalized across the neighbourhood via a softmax function.

A defining structural property of this formulation is that all weight matrices are globally shared across the entire graph. The network does not allocate individual matrices for each specific qubit or stabilizer check. This weight-sharing paradigm is vital for scalability: it guarantees that the total parameter count of the decoder remains completely independent of the distance  $d$  or the size of the surface code. The model learns a universal set of decoding principles that can be evaluated locally at any node, where custom attention scores arise purely from the distinct data configurations fed into the shared operators.

For decoding a surface code, integrating **edge features** is the defining advantage of this model. The physical layout of the code relies heavily on spatial orientation. By incorporating edges  $e_{vu}$  into the attention score, the relative position of a data qubit around its stabilizer (e.g., North, East, South, West) explicitly guides the message passing. Consequently, the network can weight its neighbours based not only on **what** information they carry (active vs. silent stabilizers), but also on **where** they are physically located. This structural awareness allows the GNN to naturally learn and compensate for lattice orientation, boundary effects, and localized error distributions, making it an exceptionally robust decoder.

Table 5.1 summarizes the comparison.

## 5.4 Expressiveness of GNNs based on message passing

Before trusting a Graph Neural Network with a quantum decoding task, it is essential to investigate the mathematical limits of what message passing can actually compute. We must determine whether the network has the theoretical

| Layer                        | Neighbour weighting            | Relevance for decoding                                      |
|------------------------------|--------------------------------|-------------------------------------------------------------|
| <b>GCN</b> [36]              | Fixed by node degree           | Inefficient; blind to error content                         |
| <b>GraphSAGE</b> [37]        | Uniform average                | Sub-optimal; erases error severity                          |
| <b>GAT</b> [38]              | Learned dynamic attention      | Efficient; prioritizes active defects                       |
| <b>GIN</b> [39]              | Injective vector sum           | Strong; preserves exact defect counts                       |
| <b>Transformer Conv</b> [40] | Content and geometry attention | Ideal; position-aware weighting (the one used in this work) |

**Table 5.1:** Main message-passing layer families and their fit to the decoding problem.

capacity to distinguish different error configurations, or if it cannot distinguish patterns due to the repetitive nature of the lattice.

Historically, this expressiveness problem is mathematically equivalent to the **graph isomorphism problem**: determining whether two distinct visual or structural representations of a graph share the exact same topological connectivity. To evaluate a GNN’s ability and measure this capability, literature relies on the classical **Weisfeiler-Lehman (WL) graph isomorphism test** [42]. The benchmark works by feeding the algorithm two separate graphs that are structurally different but highly lookalike. The test operates as an iterative coloring process: each node starts with an initial color based on its local features, and in every subsequent round, it updates its color by aggregating the multiset of its neighbours’ colors. If the algorithm finishes and outputs the exact same distribution of colors for both graphs, the test is fooled, falsely declaring them identical. Because the message-passing mechanism of a standard GNN mimics this exact neighbor-aggregation process, a GNN can distinguish different graph structures at most as well as the WL test does [39, 43]. If the test yields the same color profile for two different inputs, a standard GNN will inevitably produce the same numerical output, failing to differentiate them.

From established graph theory, we know the exact structural limits where this test, and consequently any standard GNN, is mathematically proven to fail. The WL test is systematically fooled by highly symmetric, anonymous, and strongly regular graphs. In such structures, because every node shares the same degree and lacks unique identifying identifiers, the local neighborhood patterns look identical from every point, erasing global topological differences. This theoretical ceiling

represents a known hazard for quantum computing, as a bare surface-code lattice is exactly a highly symmetric, uniform regular graph. If a standard GNN were applied directly to an anonymous surface code, the structural similarities of different error chains would trigger the WL failure mode, rendering the decoder incapable of locating localized defects accurately.

So the network can only reach this theoretical ceiling if its mathematical aggregation is perfectly injective, which is the exact motivation behind GIN’s summation. Consequently, any spatial structures that confuse the WL test, such as highly symmetric, regular graphs, are provably impossible to distinguish for standard GNN architectures as well.

For the scope of this thesis, this theoretical ceiling is not a restrictive bottleneck, due to three fundamental reasons:

- **Rich Node Features:** The surface-code graph is not an anonymous, uniform regular graph. Every node carries distinctive, high-dimensional information, including dynamic syndrome bits, precomputed Bayesian marginals, and spatial coordinates. WL-style limitations are most severe when nodes lack distinguishing identifiers, whereas our graph is heavily annotated from the start;
- **Local-to-Regional Inference:** The objective of a quantum decoder is not to compute a global property of the entire graph, such as determining if two complete lattices share the same topological structure. Instead, it performs local and regional statistical inference to determine which specific qubits explain the observed defects. This targeted task is fully manageable for local message passing, provided the network depth scales with the code distance;
- **Architectural Enhancements:** The two specific areas where standard message passing faces genuine structural shortages, capturing topological counts beyond the WL limit and propagating routing information rapidly across the entire lattice, are directly solved by design in this work. This is achieved by injecting precomputed local marginals as input features and by introducing the global node architecture, which allows for long-range information flow;

In summary, identifying these exact theoretical boundaries is precisely what caused the introduction of the two non-standard ingredients in our architecture.

## 5.5 Heterogeneous graphs and factor graphs

The surface code needs different representations for different objects; data qubits, stabilizer checks, and auxiliary belief structures are objects of different nature,

carrying feature vectors of different sizes and meanings. Forcing them into one node type would either truncate information or pad everything to a common format and ask the network to learn, from data, distinctions we already know.

The clean solution is a **heterogeneous graph**, formally  $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_v, \mathcal{T}_e)$  where  $\mathcal{T}_v$  and  $\mathcal{T}_e$  are the sets of node and edge **types**. Message passing generalizes by giving every edge relation its own message function and every node type its own update: for a target node  $v$  of type  $\tau \in \mathcal{T}_v$ ,

$$h_v^{(l+1)} = \phi_\tau \left( h_v^{(l)}, \sum_{r \in \mathcal{T}_e} \bigoplus_{u \in \mathcal{N}_r(v)} \psi_r(h_v^{(l)}, h_u^{(l)}, e_{vu}) \right), \quad (5.8)$$

where  $\mathcal{N}_r(v)$  are the neighbours of  $v$  along relation  $r = (u_{\text{type}}, \text{edge}_{\text{type}}, v_{\text{type}})$ , and  $\psi_r, \phi_\tau$  are separate networks per relation and per type.

This heterogeneous graph framework serves as the exact starting point for our architectural design. In the surface code, nodes are not identical because data qubits and stabilizer checks represent entities of a completely different nature. Data qubits hold the physical error states, whereas stabilizers perform the parity measurements. Because these objects carry distinct feature dimensions and meanings, a heterogeneous graph structure is required to allow messages to travel asymmetric paths between the two types of nodes.

From a historical perspective, this specific heterogeneous network layout is not a novel invention of deep learning. Instead, this exact structure has been studied for decades in classical mathematics and statistical inference under the name of a **factor graph**. A factor graph is a bipartite network designed specifically to model probability distributions [44]. It is composed rigorously of two distinct sets of nodes: **variables**, representing the hidden states whose values are unknown, and **factors**, representing the mathematical constraints or rules that bind those variables together. The structural correspondence with our decoding problem is absolute. The variables of the factor graph match our data qubits, while the factors match our stabilizer checks enforcing the parity constraints. The heterogeneous graph of our neural network and the classical factor graph are identical structures.

On a classical factor graph, statistical inference is traditionally performed using a famous message-passing algorithm known as **belief propagation**. This classical algorithm calculates error locations by circulating probability values along the edges of the graph. In standard belief propagation, the mathematical functions used by the nodes to compute these updates are rigid and analytical, consisting of fixed products and summations of probabilities designed a priori by researchers. This is exactly where the Graph Neural Network introduces a fundamental shift. The GNN operates on the same underlying topology and forces messages to follow the exact same communication paths as the classical algorithm. However, it replaces the fixed analytical update rules with neural networks that learn from data. Consequently,

the heterogeneous GNN acts as an intelligent, trainable version of belief propagation that learns to decode autonomously.

This transition from fixed rules to learned functions is a critical necessity due to the presence of loops within the surface-code lattice. The classical belief propagation algorithm is mathematically exact only when the graph possesses a tree-like topology[45], meaning that connections branch out indefinitely without ever closing back on themselves. When a graph contains no cycles, probability distributions flow without distortion. However, the surface code is not a tree but a highly dense, uniform square grid. Moving along the edges from a data qubit to a stabilizer, and then to another qubit and stabilizer, eventually creates a closed geometric circuit known as a loop or cycle.

For classical analytical rules, these loops represent a severe failure mode. Because the formulas are rigid, probability messages circulate indefinitely within the closed loop and endlessly amplify one another. This cyclical feedback loop creates mathematical distortion, making the algorithm unstable and leading to corrupted output predictions. The GNN actively resolves this limitation. Because the network utilizes learned multi-layer perceptrons instead of static equations, it develops the capacity to recognize these cyclical geometric patterns during the training phase. Furthermore, the GNN can exploit localized geometric information that the classical algorithm cannot easily process, such as the relative spatial coordinates of the qubits on the physical chip.

## 5.6 Why GNNs fit the problem

Beyond matching the shape of the data, GNNs have properties that are valuable for decoding.

The first is **permutation equivariance**. Because messages are aggregated with a symmetric operation and the same functions  $\psi$  and  $\phi$  are applied at every node, relabelling the nodes of the graph simply relabels the outputs in the same way, without changing them. In practice this means the model treats equivalent parts of the lattice equivalently and automatically respects the translational and rotational symmetries of the surface code, instead of having to learn them from data.

The second, closely related, is **parameter sharing**. The same set of weights is reused at every node and every edge, regardless of where on the lattice they sit. The number of parameters therefore does not grow with the size of the code, which keeps the model compact and, more importantly, makes training far more sample efficient: every local configuration seen anywhere on the lattice contributes to learning the same shared weights.

The third is **locality and scalability**. Errors and stabilisers are local objects,

and message passing is local by construction, so the inductive bias of the model is aligned with the physics of the problem. As a bonus, the very same trained model can be applied to a larger lattice just by running it on a larger graph, which matters because increasing the code distance is the way we improve fault tolerance.

The computational profile makes the third point quantitative. One message-passing layer costs  $O(|E| \cdot d_h)$  operations for edge messages plus  $O(|V| \cdot d_h^2)$  for the node updates, where  $d_h$  is the hidden width. On the surface code both  $|V|$  and  $|E|$  grow linearly with the number of physical qubits  $N$  (every qubit has a bounded number of neighbours), so inference scales as  $O(N)$  at fixed depth and width, matching the linear-scaling requirement stated for the belief structures.

## 5.7 Going deep: over-smoothing, over-squashing, and stabilization

The receptive-field argument suggests a simple recipe for larger codes: add layers. Depth, however, is not free, and deep GNNs exhibit two characteristic pathologies that any serious architecture must engineer around.

The first is **over-smoothing**: since every layer replaces a node’s state with an aggregate of its neighbourhood, repeated aggregation acts as a diffusion process, and node representations across the graph converge toward one another until they become indistinguishable, so deep GNNs can paradoxically perform worse than shallow ones [46]. For a decoder this is fatal in a specific way: the output must sharply differentiate the one qubit to flip from its unflipped neighbours, which is exactly the distinction diffusion erases.

The second is **over-squashing**: when information from an exponentially growing receptive field must travel to a node through a fixed-width vector, long-range signals are compressed, or squashed, beyond recovery, bottlenecked by the graph’s connectivity [47]. On a distance- $d$  lattice, evidence about a chain touching the far boundary is  $O(d)$  hops away; relying on depth alone means both paying  $O(d)$  layers and squeezing that evidence through every intermediate node.

The remedies used in this work are the standard ones plus one structural addition. Residual connections let each layer learn a **correction** to the identity, so node identity is preserved by default and gradients flow through the skip path [48]; layer normalization re-centres and re-scales each node’s features, keeping activations in a stable range independently of graph size [49]. Against over-squashing, the architecture adds a **global node** connected bidirectionally to every other node: it acts as a one-hop shortcut through which lattice-wide context is broadcast, collapsing the effective diameter of the graph from  $O(d)$  to 2 for the information that genuinely is global. Local reasoning stays on the lattice edges; global reasoning takes the shortcut.

## 5.8 Building GNNs in practice: PyTorch Geometric

PyTorch Geometric (PyG) [50] is the library used to implement all the models in this thesis. The following is a minimal introduction to its three core abstractions, such as graph representation, message-passing layers and heterogeneous support.

**Storing a graph** A graph is a **Data** object. Node features are stored in **x**, a float tensor of shape  $[N_v \times d_v]$ , where each row is the feature vector of one node. Connectivity is stored in **edge\_index**, a long tensor of shape  $[2 \times N_e]$  in coordinate (COO) format: the first row lists source nodes, the second lists target nodes, and column  $k$  together identifies the  $k$ -th directed edge. Because only the  $N_e$  existing edges are stored rather than the full  $N_v \times N_v$  adjacency matrix, graphs with thousands of nodes but sparse connectivity are handled efficiently. An optional **edge\_attr** tensor of shape  $[N_e \times d_e]$  attaches a feature vector to each edge.

**Implementing a layer** Every message-passing layer inherits from the base class **MessagePassing** and overrides two methods. **message()** receives the features of the source node (**x\_j**) and the target node (**x\_i**) for each edge and returns the message  $\psi$  of Equation (5.1); if **edge\_attr** is present it is passed here as well. **update()** receives the aggregated message for each node and its own current state and returns the new node state  $\phi$ . The aggregation  $\oplus$  is declared once in the constructor as the **aggr** argument and can be 'sum', 'mean', or 'max'. Calling **self.propagate(edge\_index, x=x)** inside **forward()** triggers the full pipeline automatically: PyG distributes features along edges, calls **message()** on each one, reduces the results per target node with the chosen aggregation, and passes the output to **update()**.

**Heterogeneous graphs** When the graph has multiple node and edge types, PyG provides **HeteroData** in place of **Data**. It behaves like a nested dictionary: **data['node\_type'].x** gives the feature matrix of one node type, and **data[src, relation, dst].edge\_index** gives the connectivity of one directed relation. The **HeteroConv** container takes a dictionary that maps each relation to a layer instance and runs each layer on its own slice of the graph independently, then sums the results per destination node type. This is a direct implementation of Equation (5.8): a separate  $\psi_r$  per relation, a separate  $\phi_\tau$  per node type. Because each relation has its own layer with its own weights, the network can learn different message semantics for different kinds of connections, which is the architectural point of heterogeneous message passing.

**Mini-batching** Training on a single graph at a time would waste GPU throughput. PyG batches multiple graphs into one block-diagonal graph: feature matrices are concatenated and edge indices are offset so that no edge crosses between graphs. A single forward pass therefore processes the whole batch, and because message passing never crosses disconnected components, batched and per-graph inference are mathematically identical.

## Chapter 6

# Practical Implementation and Architecture

With the theoretical foundations established, this chapter details the practical implementation of our decoding framework. The architecture is highly modular, comprising several distinct components: the quantum circuit simulator for data generation, the auxiliary data structures representing local error beliefs, the Graph Neural Network (GNN) processing these structures, and the interactive environment that manages the sequential decoding process. We first describe these individual building blocks before detailing how they are integrated within the Reinforcement Learning (RL) environment to perform Imitation Learning.

### 6.1 Fast Syndrome Generation and Temporal Compression

To train the neural decoder, we require massive amounts of syndrome data simulating the surface code under realistic noise. For this, we utilized **Stim**, a highly optimized quantum stabilizer circuit simulator. Specifically, by leveraging Stim’s built-in circuit generation (`stim.Circuit.generated`) and its highly efficient sampling capabilities (`sampler.sample`), we were able to generate thousands of cumulative syndrome rounds almost instantaneously. This approach eliminated the severe computational bottleneck of simulating quantum tracking step-by-step, drastically accelerating the entire data-generation pipeline. To facilitate the initial training phases of the neural network, we aimed to expose the model to the simplest possible error configurations: syndromes containing isolated, single errors. Achieving this purely by tuning the physical noise on data parameter presents a dilemma: setting the noise too high yields overly complex, overlapping error

chains, while setting it too low yields an overwhelming majority of empty (all-zero) syndromes, which provide no useful learning signal. To resolve this, we employed a rejection sampling strategy combined with temporal compression creating a function called `create_detector_events_and_truth_fast_compressed()`. We set an intentionally low physical error rate (e.g.,  $p = 0.005$ ) and generated a vast excess of measurement rounds. The algorithm then scans the trajectory and filters out all rounds where the detector events remain unchanged. By discarding the empty rounds, the environment steps forward only when a meaningful syndrome change occurs. This ensures the model is consistently fed actionable, simple data, vastly improving sample efficiency and preventing the two degenerate behaviours of never learning to correct or never acting at all. In practice, at the training noise levels used here, one compressed step absorbs on average 8 – 9 physical measurement rounds; a 50-step episode thus spans roughly 400 – 450 physical rounds of the code. Note that this is the training noise level, deliberately low to expose the network to isolated errors; all decoders are evaluated at the operating point  $p = 0.05$  defined in Section 3.6 with the exception of the lookahead model which is tested in a combination of data noise and readout noise.

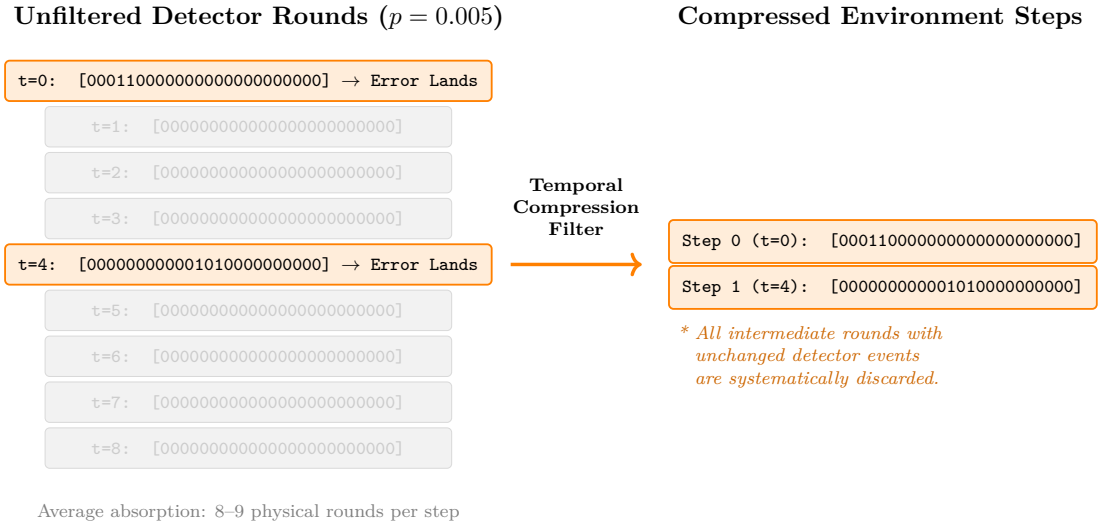


**Figure 6.1:** Visualization of the temporal compression strategy executed by the `create_detector_events_and_truth_fast_compressed()` function. Unchanged, static rounds (grey) are filtered out, forcing the environment to advance only when a dynamic syndrome change occurs (orange)

In this initial training phase, we deliberately assumed an absence of measurement noise, also known as **readout errors**. A readout error occurs when a random fault affects the ancilla qubit used to extract the syndrome, rather than the data qubits themselves. This type of error possesses distinct spatial and temporal signatures.

**Spatially** because a single readout error typically manifests as an isolated 1 in the syndrome, while a single physical data error triggers two adjacent 1s since a data qubit is checked by two stabilizers of the same type, and **temporally**, because the measurement fault does not persist on the data layer, but it is an event that appears in round  $n$ , only to disappear in round  $n + 1$ . While identifying such faults might seem theoretically straightforward, high-noise regimes severely complicate the analysis. For instance, a genuine physical data error can randomly occur at the exact spatial location of a readout error from the previous step, masking the temporal signature. Furthermore, an even number of readout errors can occur in the same vicinity, confusing the spatial parity checks. For simplicity, we initially deactivated readout noise to establish a solid training baseline. However, we explicitly recognize this phenomenon as the precise domain where our proposed framework can outperform MWPM. Traditional MWPM is strictly memoryless: it analyzes each syndrome round independently as if seeing it for the first time, with no concept of historical evolution. In contrast, our POMDP formulation is equipped with auxiliary data structures, the local marginals, that natively retain the syndrome history. This built-in temporal awareness uniquely equips the GNN to distinguish between persistent physical errors and transient readout faults, providing a fundamental theoretical advantage over static decoding heuristics.

## 6.2 Auxiliary Data Structures: Beliefs and Local Marginals

To achieve fault tolerance at scale, a decoding algorithm must exhibit computational complexity that scales linearly with the number of physical qubits. In a Partially Observable Markov Decision Process (POMDP), the agent makes decisions based on a **belief** state, a probability distribution over all possible hidden environment states. However, in the context of the surface code, tracking the exact global belief over all possible Pauli error configurations on  $N$  data qubits requires updating  $O(4^N)$  parameters. This exponential explosion makes exact tracking intractable precisely at the large code distances we target. To avoid this problem and achieve the required  $O(N)$  linear scalability, we must approximate the global belief. We accomplish this by factorizing the full joint belief into a set of overlapping, localized probability distributions known as **marginals**. While the full belief represents the exact global probability of the entire lattice state, a marginal isolates a small subset of adjacent qubits. This allows us to track local error correlations exactly while keeping the memory and computational overhead strictly bounded. As implemented in our environment, we maintain three specific auxiliary marginal structures to track the evolving quantum state:

- **Stabilizer-centric marginals (Type-5 in the implementation):** joint

distribution over an ancilla and the four data qubits it checks ( $2^5 = 32$  configurations);

- **Check-qubit marginals (Type-2 in the implementation):** A 16-entry table is created for each (check, data-qubit) pair. This table is structured around two check-bit slots (one for each stabilizer type) and the four Pauli classes I, X, Z, Y of the qubit. During emission, the system only actively conditions the bit of the paired check, while the other slot is kept simply to maintain a uniform layout;
- **Cross-check marginals (Type-3 in the implementation):** joint distribution over a data qubit together with the Z-check and the X-check that both touch it — the structure that exposes the X–Z correlation of a Y error;

Each of the three structures lives through the same four operations, repeated at every step of the episode in the order dictated by the environment loop: a deterministic relabeling when the agent acts, a noise convolution when the trajectory advances to the next round, a Bayesian conditioning when the new syndrome arrives, and a marginalization when the observation for the network is built. The mathematics of each operation is identical across the three families; only the bit layout it acts on changes.

1. **Prediction:** Between one compressed step and the next, new physical errors may have struck. The prediction step encodes this uncertainty by spreading every cell’s probability mass toward its nearest neighbours in the state space: a cell representing the absence of error on qubit  $q$  lends a fraction  $p$  of its value to the cell of the same qubit  $q$  that has an error and vice versa. The result is a distribution that is slightly flatter than before, and so less confident, by exactly as much as the noise allows.

The three families do this differently because they track different components, and not all components are equally noisy:

- In the ancilla-centric table that contains five bits, one for the ancilla’s own measurement outcome and four for the neighbouring data qubits’ error states, the spreading is applied separately and independently to each bit, using the correct rate for that component; ancilla noise for the first bit, data noise for the other four. Boundary ancillae are a special case: they touch only two data qubits, so only those two real slots are spread; the remaining two bits, which correspond to positions that do not exist on the boundary, are left untouched;
- In the cross-check table, where there are three bits, one check from each sector and one data qubit in between, all three are spread independently

with their respective rates: the two check bits with ancilla noise, the data bit with data noise;

- The pairwise table takes a simpler route: instead of spreading bit by bit, it dilutes the entire 16-entry distribution uniformly. A fraction  $p/15$  of each cell's mass is redistributed equally over all 15 other cells, producing the same net effect as a depolarizing channel on the joint Pauli state; correct for a table that tracks a full  $\{I, X, Z, Y\}$  distribution rather than individual bits;

2. **Emission:** When the new residual syndrome arrives, the tables are updated to be coherent with what was just observed. The mechanism is the same for all three: for every cell in the table, compute how likely the observed syndrome would have been if that cell's hypothesis were true, then multiply the cell's current probability by that likelihood and renormalize. Cells consistent with the observation are amplified; cells inconsistent with it are suppressed.

The key difference among the three families is how many syndrome bits a single table row is sensitive to, and which bits those are. The ancilla-centric table tracks one ancilla together with its neighbourhood. Each of its rows corresponds to a specific ancilla, and at emission time that row is sensitive to exactly one syndrome bit: the one reported by that ancilla. Each of the 32 cells is multiplied by a large factor if its ancilla bit agrees with the observed value, and by a small factor if it disagrees. The data-qubit bits in that row are never directly tested; their probabilities shift only indirectly, because they are correlated with the ancilla bit inside the same joint table. The pairwise table works identically: each row is attached to one specific ancilla, and the likelihood factor touches only the ancilla bit at the position corresponding to that ancilla's type, leaving the two data-qubit bits to move only indirectly.

The cross-check table behaves differently. Each row tracks one data qubit with one  $Z$ -type check and one  $X$ -type check. At emission time, the row is sensitive to both syndrome bits simultaneously: the likelihood applied to each cell is the product of two factors, one for the  $Z$ -ancilla outcome and one for the  $X$ -ancilla outcome. This joint conditioning is the point of the structure: a  $Y$  error on that qubit activates both checks at once through the same hidden bit, and only by testing both outcomes together can the table correctly recognize the two simultaneous firings as a single event with probability  $p/3$ , rather than treating them as two independent coincidences each with probability  $(p/3)^2$ ;

3. **Action update after agent correction:** When the agent applies a correction to a qubit, the residual error changes deterministically: the hypotheses do not become more or less certain, they simply need to be relabelled. Every cell that described the presence of an  $X$  error of the qubit  $q$  after an action on

that qubit describes the absence of that error, and vice versa. This is a pure permutation of the probability table, with no renormalization;

4. **Marginalization: from joint tables to node features** The network never sees the raw joint tables. When the observation is built, the extra components are summed out and the result is renormalized. The ancilla-centric tables sum out their own ancilla bit, leaving a 16-way posterior over the neighbourhood’s error configurations, and augmented with the two derived scalars of Section 6.3, forms the 18-dimensional `stab_fact` feature. The pairwise tables sum out both ancilla bits, leaving a four-way distribution over the qubit’s Pauli class  $\{I, X, Z, Y\}$ ; the cross-check tables sum out both check bits, leaving the qubit’s error probability. For these two families, the rows attached to the same qubit, one per neighbouring check, or per check pair, are then summed together and renormalized, producing a single per-qubit belief that feeds its `data_fact` node.

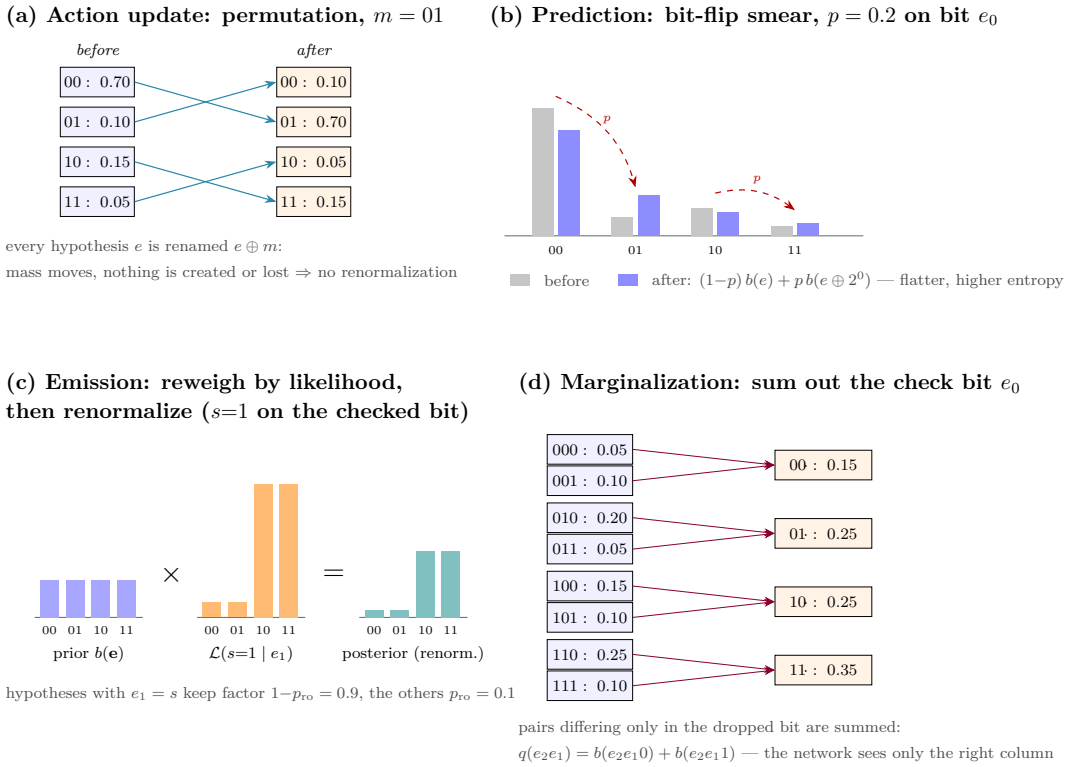


**Figure 6.2:** The four operations of the belief life-cycle

Ultimately, these actively maintained marginals serve as a fundamental support structure for the Graph Neural Network. Rather than forcing the neural network

to deduce error probabilities entirely from scratch using only raw, binary syndrome arrays, we embed these marginal distributions as continuous node features within a heterogeneous Factor Graph. By feeding the GNN these pre-computed Bayesian priors, we inject a powerful inductive bias into the network. This drastically simplifies the GNN’s optimization landscape, allowing it to focus its representational capacity on resolving complex, global topological ambiguities, such as identifying the correct homology class, rather than computing local probabilities.

### 6.3 The Heterogeneous Graph Neural Network (HeteroGNN)

The core decision-making module of our architecture is a **Heterogeneous Graph Neural Network (HeteroGNN)**, developed using the **PyTorch Geometric (PyG)** library. The surface code lattice, its syndrome, and the belief structures of the architecture are mapped onto a heterogeneous factor graph as described in the previous section, with five node types, each assigned to a specific physical or probabilistic entity:

- **data nodes (dim. 12)**: Each data qubit  $q$  is split into two component nodes, one for its  $X$  component and one for its  $Z$  component, so that the GNN can treat the two Pauli sectors with separate learned weights. Each component node carries: a one-hot sector flag ( $X$ : [1,0],  $Z$ : [0,1]), the qubit’s normalized row and column coordinates on the lattice (2 scalars), the two entries of the classical Pauli tracking frame  $c_X, c_Z$  (i.e. whether a correction has already been applied), three normalized time scalars (**step\_norm**, **delta\_rounds\_norm**, **remaining\_norm**), and three lattice-wide density scalars (mean residual  $Z$  syndrome, mean residual  $X$  syndrome, mean correction density);
- **stab nodes (dim. 1 or 3)**: One node per ancilla measurement, carrying simply the current residual syndrome bit (0 or 1). In the code-capacity regime this is a single scalar; in the phenomenological regime with the lookahead window of Chapter 7 it becomes a 3-channel vector holding the syndrome bits at the current round and the two subsequent rounds, giving the network the temporal information it needs to distinguish readout faults from data faults.
- **stab\_fact nodes (dim. 18)**: One node per ancilla, carrying the Type-5 marginal compressed to its 16-way marginalization (obtained by summing out the ancilla’s own bit from the 32-entry joint table, Section 6.2), augmented with two derived scalars: the total probability of any non-trivial configuration  $1 - b(\mathbf{0})$  and the Shannon entropy of the distribution [51], which measures the

level of uncertainty associated with a probability distribution. This node is the belief-injection point for the stabilizer neighbourhood;

- **data\_fact nodes (dim. 12)**: One node per data qubit, aggregating information from both marginal families that involve that qubit. From the Type-2 (pairwise) marginal, marginalized to a 4-way Pauli-class distribution  $(p_I, p_X, p_Z, p_Y)$ , the node receives 4 values plus three derived scalars:  $p_X + p_Y$ ,  $p_Z + p_Y$ , and the Shannon entropy of the 4-way distribution. From the Type-3 (cross-check) marginal it receives the scalar per-qubit error probability  $p_{\text{err}}$  and its entropy. Finally, an agreement score  $1 - |p_{\text{any}}^{(2)} - p_{\text{err}}^{(3)}|$  tells the network how consistent the two marginal families are on this qubit, and then there is a pad value to make the feature vector a uniform length of 12;
- **global node (dim. 9)**: A single node connected to every other node in the graph, providing a one-hop shortcut for lattice-wide context. Its feature vector contains three normalized time scalars, three density scalars and three mean-entropy scalars summarizing the current average uncertainty of the three marginal families across the whole lattice;

| Node type | Physical meaning     | Feature content                                   | Dim. (rows × cols)                |
|-----------|----------------------|---------------------------------------------------|-----------------------------------|
| data      | Data qubit           | Coords, Pauli frame                               | $2n \times 12$                    |
| stab      | Syndrome measurement | Residual syndrome bit(s)                          | $2n_{\text{anc}} \times 1$ (or 3) |
| stab_fact | Stabilizer factor    | Type-5 marginal row ( $2^5$ configs, compressed)  | $2n_{\text{anc}} \times 18$       |
| data_fact | Correlation factor   | Type-2/3 marginal rows                            | $n \times 12$                     |
| global    | Lattice-wide context | Step norm, residual densities, marginal entropies | $1 \times 9$                      |

**Table 6.1:** Node types of the heterogeneous factor graph, their input features, and the shape of the feature matrix passed to the GNN at each step.  $n = d^2$  is the number of data qubits,  $n_{\text{anc}} = (d^2 - 1)/2$  the number of ancillae of each type; the **stab** feature width is 1 in the code-capacity regime and 3 with the lookahead window

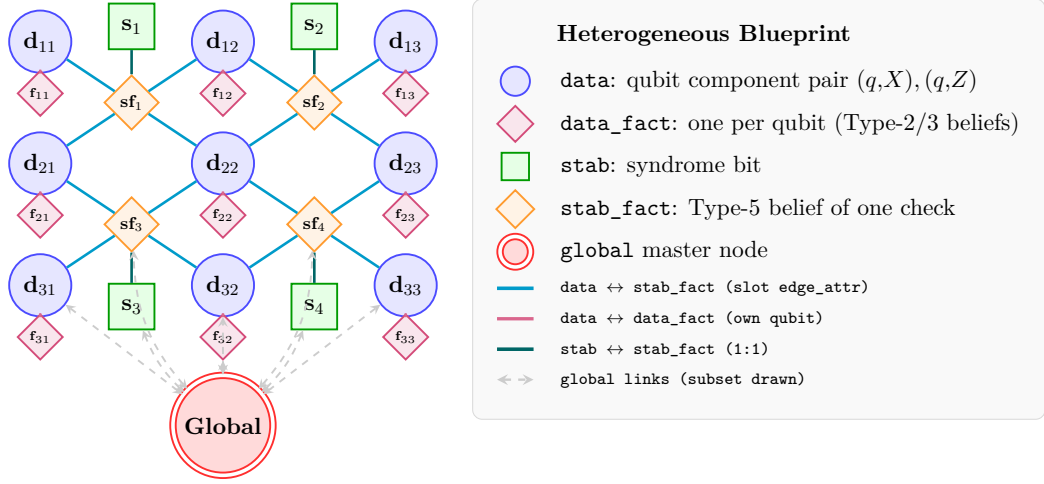


**Figure 6.3:** Mapping of a distance-3 patch onto the heterogeneous factor graph built by `graph_builder`

The communication between the statistical priors calculated in our environment and the neural network occurs via a strict, node-by-node feature mapping. Instead of passing unstructured matrices to the network, the rows of our auxiliary data structures are explicitly mapped to their corresponding graph representations. In this way, each individual row of the Type 5 marginal matrix, which contains the  $2^5$  local joint error probabilities computed via Bayesian inference, is directly injected as the feature vector of the corresponding stabilizer factor node. In this manner, that specific node instantly possesses the full localized probabilistic belief surrounding its stabilizer. At the same time, the rows of the Type 2 and Type 3 matrices are assigned to the data factor nodes, providing them with immediate awareness of the pairwise and linear spatial correlations generated by the depolarizing noise and CNOT gates. The raw binary syndrome bits (0 or 1) populate the stabilizer nodes, while the physical coordinates and the current state of the classical Pauli tracking frame populate the data nodes. To process this highly structured information, the network architecture leverages PyG’s **HeteroConv** container, which wraps around individual **TransformerConv** layers.

A distinctive architectural choice is the inclusion of a global node. In a standard GNN, information propagates locally, step-by-step, meaning that for a message to travel from one side of a distance- $d$  lattice to the other, it requires  $d$  successive layers of message passing. This creates a communication bottleneck. The **global** node circumvents this by maintaining bidirectional edges with every single node in the graph. During each forward pass, it receives aggregated features from the entire lattice, normalizes them against macro-parameters (such as the current normalized time step, `step_norm`, and the overall densities of the residual

syndromes, `residual_z_density` and `residual_x_density`), and broadcasts this global context back to all nodes instantaneously. This multi-scale processing ensures that local decoding actions are always executed with awareness of the global state of the code. Ultimately, the HeteroGNN is the component that integrates the pipeline’s heterogeneous information. At each step the environment produces binary syndromes, local Bayesian probabilities, spatial coordinates and tracking frames; the GNN organizes them according to the translational symmetries of the lattice, turning a collection of local beliefs into a coherent decoding policy.

## 6.4 The Interactive Decoding Environment

All the aforementioned components are managed by a custom environment (`PyMatchingImitationEnv`) inheriting from the standard gymnasium API[52]. The environment acts as the bridge between the quantum simulation and the learning agent. The environment maintains a strict separation between the true physical state, the actual errors sampled by Stim, which the agent cannot see, and the agent’s tracked state. During a step, the environment performs the following loop:

1. It applies the agent’s chosen action by updating the classical Pauli tracking frame;
2. It calculates the residual syndrome, which is the physical syndrome XORed with the syndrome induced by the agent’s tracked corrections;
3. It steps the quantum simulation forward to the next non-trivial round, bringing in new physical errors and measurement faults;
4. It updates the three marginals based on the new residual syndrome;
5. It reconstructs the heterogeneous factor graph to serve as the observation (obs) for the next step;

This setup perfectly embodies the POMDP nature of the problem: the agent never modifies the physical quantum state, but continually updates its internal software frame to neutralize the observed syndromes.

## 6.5 Imitation Learning Pipeline and Policy Bootstrapping

Once the individual components, such as the simulation environment, the auxiliary marginal structures and the Heterogeneous GNN, are defined, they must be integrated into an optimization pipeline. Learning a decoding strategy from scratch

using exclusively Reinforcement Learning (RL) is notoriously inefficient due to the sparsity of the reward signal; a random agent might explore millions of sub-optimal action paths before accidentally neutralizing a logical defect. To circumvent this, we implement a preparatory **Imitation Learning (IL)** phase designed to bootstrap the network’s policy by training it to mimic the expert heuristic of MWPM.

### 6.5.1 Training loop and configuration

The optimization is governed by a configuration object grouping three kinds of parameters: environmental, such as code distance  $d$ , physical error rate  $p$ , number of episodes and exploration and expert-guidance schedules, optimization, like learning rate, gradient clipping threshold, gradient accumulation steps, and loss shaping, such as class weights and outcome multipliers. Each episode is a full correction trajectory: the environment is reset with a fresh temporally-compressed Stim trajectory and the first factor graph is built; at every step the agent produces logits over the data qubits while the environment queries the MWPM oracle for the expert labels; a DAgger-style schedule [53] decides whether the action actually applied to the environment is the expert’s or the model’s own, mitigating the compounding of errors typical of sequential imitation; the episode ends at the round budget, when a terminal check classifies the trajectory as clean or as a logical failure.

### 6.5.2 Score metric and loss design

Convergence toward the expert is monitored by a per-step matching score that penalizes false positives and false negatives against the expert’s correction vector of actions, averaged over the episode and smoothed with a moving window. The learning signal itself is a Binary Cross-Entropy loss over the per-qubit logits. Because physical errors are sparse, the targets are heavily imbalanced and plain BCE collapses to the all-zero policy; a positive-class weight rebalances it.

For each step, let  $y_i \in \{0,1\}$  be the expert label for qubit component  $i$ , and let  $\hat{p}_i$  be the model’s sigmoid output. The total per-step loss  $\mathcal{L}$  balances three distinct objectives:

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \lambda_{\text{card}} \mathcal{L}_{\text{card}} + \lambda_{\text{iou}} \mathcal{L}_{\text{iou}} \quad (6.1)$$

where the individual terms are defined as follows:

- **Weighted Binary Cross-Entropy ( $\mathcal{L}_{\text{BCE}}$ ):**

$$\mathcal{L}_{\text{BCE}} = \frac{1}{n} \sum_{i=1}^n \left[ -w_+ y_i \log \hat{p}_i - (1 - y_i) \log(1 - \hat{p}_i) \right]$$

We set  $w_+ = 10$  to counter error sparsity and prevent the model from collapsing into an all-zero strategy.

- **Cardinality Penalty ( $\mathcal{L}_{\text{card}}$ ):**

$$\mathcal{L}_{\text{card}} = \sum_{s \in \{X, Z\}} \left| \sum_i \hat{p}_i^{(s)} - \sum_i y_i^{(s)} \right|$$

This term forces the network to match the total number of errors per Pauli sector ( $X$  and  $Z$ ), which standard BCE struggles to learn directly.

- **Intersection-over-Union Loss ( $\mathcal{L}_{\text{iou}}$ ):**

$$\mathcal{L}_{\text{iou}} = \sum_{s \in \{X, Z\}} \left( 1 - \text{IoU}_s \right) \quad \text{with} \quad \text{IoU}_s = \frac{\sum_i \hat{p}_i^{(s)} y_i^{(s)}}{\sum_i \hat{p}_i^{(s)} + \sum_i y_i^{(s)} - \sum_i \hat{p}_i^{(s)} y_i^{(s)}}$$

This differentiable IoU measures the alignment between predicted and true error locations, penalizing predictions that get the count right but the placement wrong.

The auxiliary regularization weights are set to  $\lambda_{\text{card}} = \lambda_{\text{iou}} = 0.5$ . The DAgger schedule that governs whose action the environment executes is a single ramp,

$$\varepsilon(\text{ep}) = \min \left( 1, \frac{\text{ep} - \text{ep}_0}{f \cdot N_{\text{ep}}} \right), \quad (6.2)$$

where  $\text{ep}_0$  is the episode at which the hand-off begins,  $N_{\text{ep}}$  the training budget and  $f$  the ramp fraction: with probability  $\varepsilon$  the environment executes the model's own thresholded action, otherwise the expert's, so the state distribution slides continuously from teacher-forced to autonomous. Gradients are norm-clipped before each optimizer update.

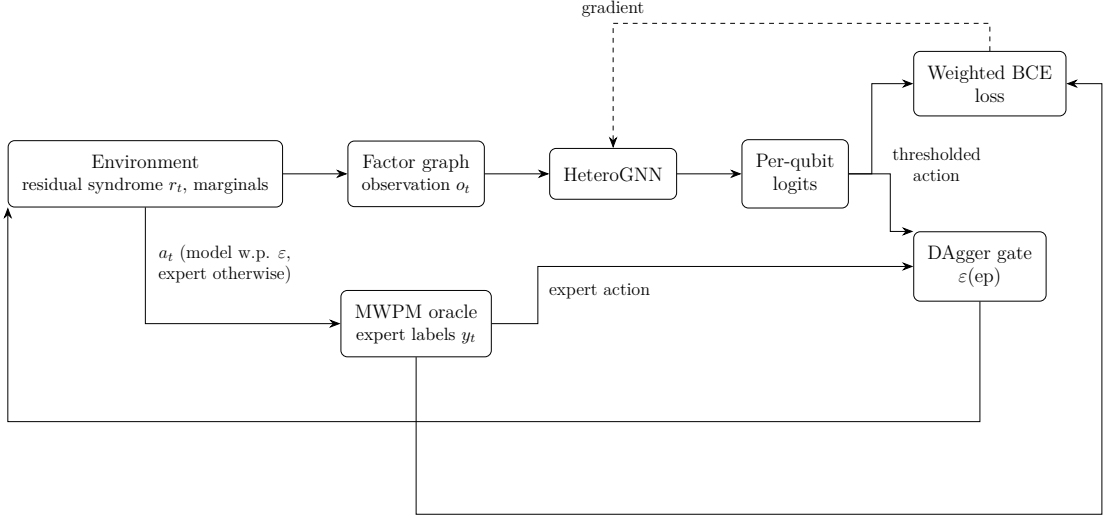


**Figure 6.4:** One step of the imitation training loop. The environment produces the residual syndrome and the belief structures, assembled into the factor graph observation; the HeteroGNN maps it to per-qubit logits. The MWPM oracle, queried on the same state, supplies both the supervision targets for the weighted BCE loss and the expert action; the DAgger gate decides whether the action executed by the environment is the model’s or the expert’s, sliding from teacher-forced to autonomous over training.

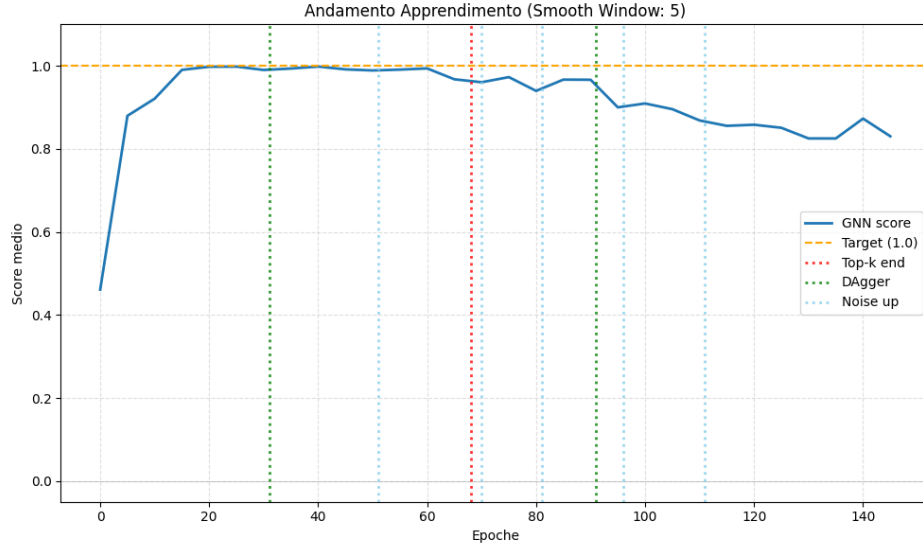


**Figure 6.5:** Training dynamics of the imitation decoder (run 39). Top: per-episode agreement score against the MWPM expert under the six-level noise curriculum ( $p = 0.0005 \rightarrow 0.016$ ); the score plateaus at 0.83–0.88. Bottom: training loss over the same 150 episodes.

Figure 6.5 shows how the model learned during training (run 39, using 150 episodes of 320 steps each). We used a training schedule with six noise levels, where the error rate ( $p$ ) ranged from 0.0005 to 0.016.

During training, the model’s success rate improved from near zero to a steady value between 0.83 and 0.88. The number of correct steps per episode rose from about 200 to 239, while the loss value dropped from 3.26 to roughly 0.7. This final version of the model is the one we test in Chapter 8.

### 6.5.3 Advanced Curriculum: Top-k Transition Strategy

A critical refinement embedded within our imitation pipeline is the **Top-k Curriculum Strategy**. In the very early stages of training, the model’s weights are randomly initialized, meaning its predictions are really chaotic. If the environment were to execute these random actions we would lose a lot of episodes only to obtain a meaningful action (not correct) from the model. During the first phase of training, the action sent to the environment is strictly constrained: the system evaluates the top- $k$  most confident predictions of the model, where  $k$  is the number of actions made by the expert. This in addition to the already implemented DAgger mechanism keep the training trajectories safe from divergence. As the episodes progress and the GNN’s loss decreases, this constraint is gradually relaxed. The environment

transitions to utilizing the model’s own thresholded predictions, exposing the GNN to its own marginal mistakes. This adaptive curriculum allows the network to learn how to recover from minor misalignments, bridging the gap between supervised imitation and independent reinforcement learning exploration.

## 6.6 The Lookahead Window

When syndrome measurements are perfect, the three marginal structures carry all the information the decoder needs: a defect that appeared at step  $t$  is still present at step  $t+1$  unless the agent corrects it, so the current residual syndrome already tells the full story. With readout errors active, this is no longer true. A measurement fault flips one ancilla’s outcome for exactly one round and then disappears spontaneously; it leaves no trace in the marginals, which are updated on the residual syndrome of the current step alone, so the decoder cannot distinguish a real defect, that should be corrected from a non defect that should be ignored. Treating every transient readout fault as a real error produces unnecessary corrections that dirty the lattice and cascade into future mistakes; ignoring it risks letting a real error persist. The marginal structures, by design, integrate information over time through the Bayesian update cycle, but the integration is lossy; a single-round readout glitch leaves only a faint residue in the ancilla table and vanishes within one prediction step. What the decoder needs is direct access to the raw temporal evidence before it is averaged away. The **lookahead window** provides exactly this. At every decision step  $t$ , the environment computes the residual syndrome not just for the current round but for the two subsequent rounds as well, under the same correction frame already applied. It then stacks the three syndrome vectors, which are rounds  $t$ ,  $t+1$ , and  $t+2$ , into a three-channel observation for each ancilla. A readout fault at round  $t$  produces a defect in the first channel and a defect in the second channel, because the measurement fault has corrected itself and the difference with the previous round is now visible, then nothing in the third. A real data error produces a defect in the first channel and none in the subsequent ones, because the error persists stably and the round-to-round difference disappears. The two patterns are distinct, and the network can learn to recognize them.

This changes the graph and the GNN in a targeted and minimal way. The stab nodes carry the three-channel temporal stack; the input feature vectors of the remaining node types are correspondingly extended in the lookahead family: their feature vector grows from a single scalar to a 3-dimensional vector (the three-channel stack), reflecting the  $t$ ,  $t+1$ ,  $t+2$  readings for that ancilla. The corresponding input projection layer inside the GNN widens from dimension 1 to dimension 3 accordingly. The lookahead information and the belief information therefore enter the network through separate channels and at separate nodes, and

the network can weight them independently at each attention head. The latency cost of the window is fixed at two measurement rounds, matching the standard delay of windowed classical decoders.

## Chapter 7

# Alternative Decoders and Training Extensions

In the previous chapter, we introduced the main architecture of this thesis, centered on a heterogeneous GNN trained by imitation of MWPM, and we talked about the presence or absence of readout error. That architecture is not the only one developed. The choice of what to imitate, what objective to optimize, and what information to give the network are not obvious, and three alternative designs were built to explore different answers to those questions, each targeting a specific theoretical gap.

The first gap is the one identified in the degeneracy argument: MWPM decodes the  $X$  and  $Z$  sectors independently, which means it systematically misprices  $Y$  errors and cannot reach the maximum-likelihood optimum. A decoder that reasons jointly over both sectors can beat MWPM on this margin without any sequential decision-making at all. The supervised model was built to test exactly this: it bypasses the step-by-step correction loop entirely and directly predicts the **logical class of the error**, optimizing a joint maximum-likelihood objective. In order to do this, during its supervised learning, the model needs to learn something that the imitation model could not: the correlations between different error patterns.

The second gap is architectural: the imitation decoder learns to replicate MWPM step by step, which raises the question of whether a purely reinforcement-driven agent, with no expert to copy, would discover the same behaviour or something different. The atomic model addresses this by learning entirely from reward, applying one correction at a time and receiving feedback only on the outcome. The comparison between the two is a direct empirical test of how much of the imitation decoder’s behaviour comes from the training signal and how much would emerge from reward alone.

In addition to these two gaps, the main architecture of Chapter 6 was designed

exclusively for the noiseless-measurement regime. The final model in this chapter extends it to the regime where readout errors are present.

An architectural remark is in order before describing them. The atomic and supervised models are not variants of the Chapter 6 architecture: they are deliberately new and far simpler networks, operating directly on the homogeneous Tanner graph of the code, with message-passing layers, no heterogeneous node types, no global node, and none of the Type-5, Type-3 and Type-2 belief structures of Section 6.2. Their simplicity is a feature of the experimental design: any margin they achieve or fail to achieve can be attributed to the training objective, not to the auxiliary machinery. The lookahead model, by contrast, is a direct extension of the main architecture: it reuses the full Chapter 6 pipeline, such as factor graph, marginals, imitation loop, unchanged, modifying only the observation and the training regime.

## 7.1 Atomic Model

This model was created to explore alternative ways for the network to decide on corrections. Given a syndrome, the main model tries to apply flips on the qubits that the network calculates as the most probable, while also learning to select the correct number of flips. This process can be highly demanding in terms of time and computational resources during training. The atomic model, instead, focuses on deciding **one single flip** at a time until all necessary flips for that round have been selected. It calculates the most likely flip and internally recalculates the syndrome without requiring a new physical measurement. If the resulting syndrome is not composed entirely of zeros, meaning errors are still present because we have no readout error here, it selects another qubit to flip. It functions as a Reinforcement Learning agent, specifically utilizing a value-based approach based on a Double Deep Q-Network [54], driven by a Graph Neural Network. Rather than predicting the entire correction mask at once, it iteratively builds the solution through a sequence of discrete, single-qubit actions. The name Atomic refers to the nature of its actions: each time it applies only one Pauli operator, so for one step it can apply more actions to reset the syndrome.

### 7.1.1 Deep Q-Network in the Atomic model

A Deep Q-Network is a machine learning framework that merges Reinforcement Learning with Deep Learning. At its core lies the traditional Q-learning algorithm[55], which evaluates the quality of an action taken in a specific state by estimating the total future reward it will yield. In simple scenarios, these values can be stored in a lookup table, but in complex environments with a practically infinite number of possible states, such as a quantum error correction grid, a tabular

approach becomes computationally impossible. To solve this, the model employs a deep neural network to approximate these values. The network takes the current state of the environment as input and outputs the predicted long-term values for all possible actions, learning to recognize structural patterns to evaluate even previously unseen states.

### 7.1.2 Atomic Model’s Description

At each step, the network observes the current state of the errors (the active syndrome) along with the history of corrections that have been applied up to that point. The underlying architecture relies on a series of message-passing layers that exchange information between the physical data qubits and the check nodes of the quantum code. After processing this structural information across several rounds, the model outputs a specific score for every possible action. Each action corresponds to applying a specific Pauli operator (either an  $X$ -flip or a  $Z$ -flip) on a single data qubit.

During the training loop, the agent generally selects the action with the highest score, while occasionally taking random actions to encourage the exploration of new strategies. Once a flip is chosen, the system internally updates the tracked corrections and mathematically calculates the new syndrome. This transition is simulated entirely in software, without interacting with the actual quantum environment or requiring costly new physical measurements.

The agent’s learning process is guided by a dynamic, step-by-step reward system. It receives intermediate feedback based on whether its chosen action reduced or increased the total number of active defects in the syndrome. The sequence continues until either a maximum step limit is reached or the syndrome is completely cleared (yielding an all-zero state). Upon clearing the syndrome, a final check determines whether the accumulated corrections successfully neutralized the errors or accidentally introduced a logical failure. A successful recovery grants a positive terminal reward, whereas a logical failure or a timeout results in a harsh penalty.

To stabilize the optimization, the model stores its past trajectories in a memory buffer [56]. During training, it samples random batches from this memory to update its internal weights, gradually improving its ability to estimate the long-term value of each single flip.

Finally, during the evaluation phase, a safety mechanism is introduced to prevent stalling. The system keeps a record of all previously visited states within a single decoding attempt. If the model’s preferred action leads to a state that has already been explored, which would cause an infinite loop of flipping and unflipping the same qubit, the agent is forced to select its next best available action, ensuring that the error-correction process continually moves forward.

## 7.2 Supervised Model

This model was developed to explore a fundamentally different approach to error correction: direct logical classification. While the main and atomic models attempt to identify the exact physical qubits that need flipping, the supervised model bypasses the step-by-step correction entirely. Instead, it reads the complete initial error signature and directly predicts which of the four possible logical classes the underlying error belongs to, and entrusts the task of generating the action to a deterministic function. This approach operates as a single-shot, maximum-likelihood decoder[57].

### 7.2.1 Supervised Classification Framework

Unlike the Reinforcement Learning setups discussed previously, this architecture relies purely on supervised learning. The network acts as a global classifier. It takes the combined initial syndromes as input and directly maps them to a discrete target class representing the logical state. To achieve this, a deep Graph Neural Network is employed, culminating in a final readout layer that outputs probabilities for four distinct outcomes. The optimization is driven by a standard loss function, which penalizes the model based on the divergence between its predicted probability distribution and the true, mathematically computed logical class of the training sample.

### 7.2.2 From predicted class to physical correction

The network outputs a probability distribution over four logical classes,  $I$ ,  $X$ ,  $Z$ ,  $Y$ , and selects the most probable one. Turning that abstract class into a concrete set of qubit flips is a two-step algebraic procedure that requires no learning and no approximation.

The first step is to find any correction that silences the observed syndrome, regardless of its logical class. This is a system of linear equations over the binary field  $\text{GF}(2)$ : given the parity-check matrix  $H$  of the code and the observed syndrome vector  $s$ , find a binary vector  $c$  such that  $Hc = s \pmod{2}$ . The solution is obtained by Gaussian elimination, the standard row-reduction algorithm applied to the augmented matrix  $[H \mid s]$ , proceeding column by column, swapping pivot rows and XOR-ing to eliminate non-zero entries above and below each pivot. The result is one particular solution  $c_0$ ; all other syndrome-compatible corrections differ from  $c_0$  by elements of the code's stabilizer group, and they are all logically equivalent to or differ by a logical operator from  $c_0$ . The procedure is run separately for the  $X$  sector, using the  $Z$ -check matrix and the  $X$ -error syndrome, and the  $Z$  sector, using the  $X$ -check matrix and the  $Z$ -error syndrome, producing a base correction

$$c_0 = (c_0^X, c_0^Z).$$

The second step checks whether  $c_0$  lands in the class the network predicted. The logical class of a correction is determined by computing its inner product with the two logical operators  $\bar{Z}$  and  $\bar{X}$  over  $\text{GF}(2)$ :  $\ell_X = \bar{Z} \cdot c_0^X \bmod 2$  and  $\ell_Z = \bar{X} \cdot c_0^Z \bmod 2$ . If the resulting class  $(\ell_X, \ell_Z)$  matches the network’s prediction,  $c_0$  is the final correction. If it does not, a single XOR with the corresponding logical operator moves  $c_0$  into the predicted class: XOR-ing  $c_0^X$  with  $\bar{X}$  flips  $\ell_X$ , and XOR-ing  $c_0^Z$  with  $\bar{Z}$  flips  $\ell_Z$ , at the cost of one extra logical-weight string of flips. The correction applied to the lattice is therefore always consistent with the syndrome, and its logical class is always the one the network chose. All the intelligence of the decoder is concentrated in the four-way class prediction; the algebra that builds the physical correction from it is exact and deterministic.

### 7.2.3 Supervised Model’s Description

The training loop of this model has no environment, no actions, and no trajectory. At every iteration, a fresh batch of errors is sampled from the depolarizing noise model: for each sample, an  $X$  and a  $Z$  error pattern are drawn independently, the two syndromes are computed from the parity-check matrices, and the true logical class of the error is determined by checking its inner product with the logical operators  $\bar{Z}$  and  $\bar{X}$ . This gives a label in  $\{I, X, Z, Y\}$  that is mathematically exact because the error is known exactly. The syndrome is fed to the GNN as input, and the network outputs a four-way probability distribution over the four classes. The cross-entropy between this distribution and the true label is the loss; the weights are updated immediately, and the next batch is generated. The model never sees the same error twice and never runs out of training data: the generator produces as many labeled examples as needed, all with exact labels.

At evaluation time, a large set of syndromes is generated in the same way, and for each one the network’s predicted class is compared against the true class. The fraction of wrong predictions is the logical error rate. The same syndromes are also decoded by MWPM so that the two methods can be compared on identical inputs under identical conditions; a wrong prediction means that the correction lands in the wrong homology class, meaning the combined effect of the true error and the applied correction is a non-trivial logical operator, which is a logical error, the qubit’s information is silently corrupted, and no subsequent measurement can detect it, because the code returns to a valid state in the wrong logical value.

## 7.3 Reinforcement Refinement of the Imitation Decoder

Once the imitation decoder has been bootstrapped by imitation, the original plan of this thesis called for a reinforcement learning phase to push it beyond its teacher. In the noiseless-measurement regime, this phase systematically failed in one of two ways:

- **Stall:** with conservative learning rates, the policy remained numerically indistinguishable from its imitation initialization. The gradient signal produced by the sparse terminal reward was too weak to move a policy sitting at a local optimum where every small deviation first increases the injected residual dirt;
- **Forgetting:** with aggressive learning rates or strong reward shaping, the policy escaped the initialization but lost the minimum-weight discipline before finding anything better, degrading the logical error rate;

We stress that this is a negative result with a positive interpretation: it is exactly the behaviour predicted if MWPM is a local optimum of the greedy syndrome-nulling objective[30]. The practical lesson is that reinforcement refinement needs a regime in which minimum weight is not locally optimal. That regime is the subject of the next section.

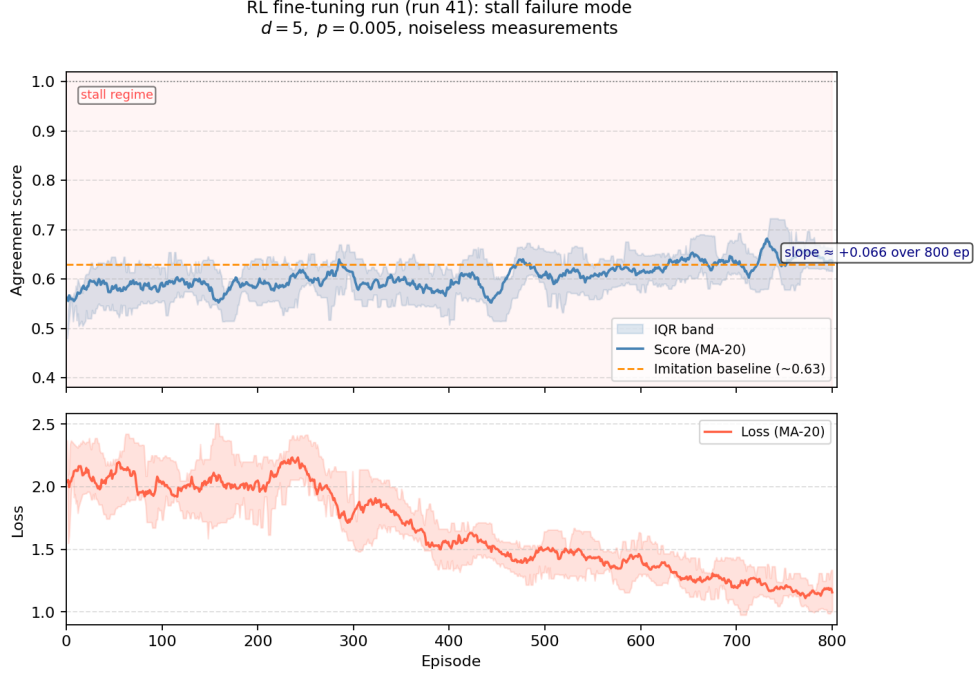


**Figure 7.1:** Training dynamics of the RL fine-tuning phase (run 41,  $d = 5$ ,  $p = 0.005$ , noiseless measurements). **Top:** agreement score (moving average window 20, with interquartile band); the orange dashed line marks the imitation baseline ( $\approx 0.63$ ) from which the run was initialized. **Bottom:** training loss over the same episodes. The score oscillates in a narrow band around the initialization for the full 800-episode budget, never breaking above the baseline, the defining signature of the stall failure mode. The slow downward drift of the loss indicates that the optimizer is active but unable to move the policy: the gradient signal from the sparse terminal reward is too weak to overcome the local minimum induced by the minimum-weight attractor

## 7.4 Lookahead model

Lookahead model is a variant of the imitation decoder that keeps the same architecture and training loop, but is trained in a regime where readout errors are present. We have talked about it before when we introduced the lookahead window, but now we can go into more details. The presence of readout errors changes the nature of the problem: the network must learn to distinguish between persistent defects, which are the result of physical errors on the data qubits, and transient defects, which are the result of readout flips. The network must learn to ignore the latter and only act on the former. This is a more complex task than the one faced

by the noiseless imitation decoder, and it requires a different training regime. It also requires more work to make the expert work: MWPM is still used, but it is queried on the clean syndrome, which is the syndrome that would be observed if there were no readout errors. The network must learn to infer the clean syndrome from the noisy measurements it receives through the lookahead window.

### 7.4.1 Syndrome generation with readout errors

The generator used in this training produces, for every episode, a sequence of measurement rounds that contains both data errors and readout faults. For each round, a depolarizing error pattern is sampled on the data qubits and the true syndrome is computed from the parity-check matrices; then, independently, each ancilla measurement is flipped with probability  $p_{ro}$ , producing the noisy measured syndrome that the environment observes. The generator records both versions: the measured syndrome, which is what the network sees through the lookahead window, and the clean syndrome, obtained by XOR-ing the measured syndrome with the readout flip mask. The clean syndrome is what MWPM would see if measurements were perfect, and the environment exposes it as a separate internal field so that the expert can query it. The compression step works differently here compared to the noiseless generator: a round is kept not only when a physical data error occurs, but also when at least one ancilla measurement is flipped; readout events are active events that the network must reason about, and discarding them would remove exactly the signal that distinguishes transient faults from persistent ones.

### 7.4.2 Lookahead training loop

The training loop is structurally identical to the pipeline of imitation model: DAgger mixing between expert and model actions, a top- $k$  constraint in the earliest episodes, the same sum of three loss functions, gradient clipping, and a smoothed score for convergence monitoring. Only four things change:

1. **The expert:** instead of querying MWPM on the residual measured syndrome, the oracle calls MWPM on the clean residual syndrome, the one with readout flips removed. The model therefore learns to imitate a teacher that already knows which defects are lies, and its job is to infer the same from the temporal evidence in the lookahead window, without ever having access to the clean syndrome at deployment;
2. **The noise schedule:** the training starts at low noise levels and promotes to harder ones only when the moving average of the episode score exceeds a

threshold, following a seven-level joint curriculum raising the data error rate from 0.003 to 0.02 and the readout rate from 0.002 to 0.015;

3. **Three stability mechanisms** absent in the base pipeline: an episode is truncated early and counted as a failure when the expert’s required correction mass grows beyond a safety threshold, signalling that the trajectory has derailed beyond recovery; the best checkpoint is saved at every new peak of the smoothed score rather than at the end of training, after an earlier run lost its best weights to subsequent degradation; and the learning rate is decayed when the DAgger coefficient reaches full autonomy, so that the network consolidates its policy rather than drifting under the higher-variance gradient signal of self-generated trajectories;
4. **A fourth loss component**: an explicit false-positive penalty that discourages corrections on qubits left untouched by the expert:

$$\mathcal{L}_{\text{fp}} = \lambda_{\text{fp}} \sum_i \hat{p}_i (1 - y_i) \quad (7.1)$$

where  $\lambda_{\text{fp}} = 0.1$ .

During the readout phase, any spurious correction causes double damage: it introduces a new physical error, and it is usually triggered by a temporary defect that would have resolved on its own. For this reason, we penalize over-correction directly in the loss function rather than relying solely on class weighting;

## Chapter 8

# Experiments and Results

This chapter evaluates imitation, atomic, supervised and lookahead decoders against MWPM, following the map drawn in the theoretical chapters: first the code-capacity single-shot regime, where the  $Y$ -correlation margin lives (Section 8.2); then the multi-round regime without measurement noise, where the minimum-weight attractor dominates (Sections 8.3-8.4); finally the phenomenological regime with readout errors, where the temporal margin is at stake (Section 8.5).

### 8.1 Evaluation protocol

**Logical error rate** The primary metric is the logical error rate (LER): the fraction of runs in which the decoder’s combined correction, together with the true error, implements a non-trivial logical operator. In the single-shot setting, a run consists of one correction of a syndrome, while the multi-round setting a run is a full episode, where errors can accumulate.

In both cases, the terminal check relies on the same algebraic criterion shared by every decoder in this work. Given the residual  $r = e \oplus c$  (true error XOR total correction) split into its  $X$  and  $Z$  components, the logical class is computed by parity against the logical operators:

$$\ell_X = \bar{Z} \cdot r_X \bmod 2 \tag{8.1}$$

$$\ell_Z = \bar{X} \cdot r_Z \bmod 2 \tag{8.2}$$

A run fails when either parity is non-trivial. This matches the class computation used by the supervised decoder of Section 7.2.2, ensuring all decoders are judged by the same notion of logical failure.

An earlier version of the pipeline used a boundary-to-boundary connectivity test (depth-first search on the residual chain) as the terminal check. The two criteria

coincide only on minimal residuals. On stabilizer-dressed residuals, the connectivity test disagrees in 10 – 28% of cases. These disagreements are almost always false negatives, meaning real logical errors are missed.

All results reported in this chapter use the algebraic criterion. The two protocols, single-shot and multi-round, remain not comparable with each other: multi-round LER grows over rounds and is systematically higher at equal  $p$ . In the multi-round tables, rounds counts compressed active steps on the filtered timeline of Section 6.1 (one step absorbs the empty physical rounds preceding it); baselines and learned decoders within the same table always share the same generator, so every comparison is internally consistent. Every table below states which protocol it uses.

**What the training score is not.** During imitation and lookahead training we monitored a per-step agreement score against the expert. This score is a diagnostic of imitation, not a valid metric of judgment: an episode containing only readout faults can achieve a high score with zero decoding value. All claims in this chapter rest on LER.

**Setup** Unless stated otherwise: rotated surface code, distance  $d = 5$ , depolarizing data noise, MWPM via PyMatching [12, 13], sampling via Stim [58]. The physical error rate for the headline comparison is  $p = 0.05$ , chosen well below the MWPM threshold ( $p_{\text{th}} \approx 0.13\text{-}0.14$ , measured on our implementation), so that a margin over MWPM is meaningful and reachable.

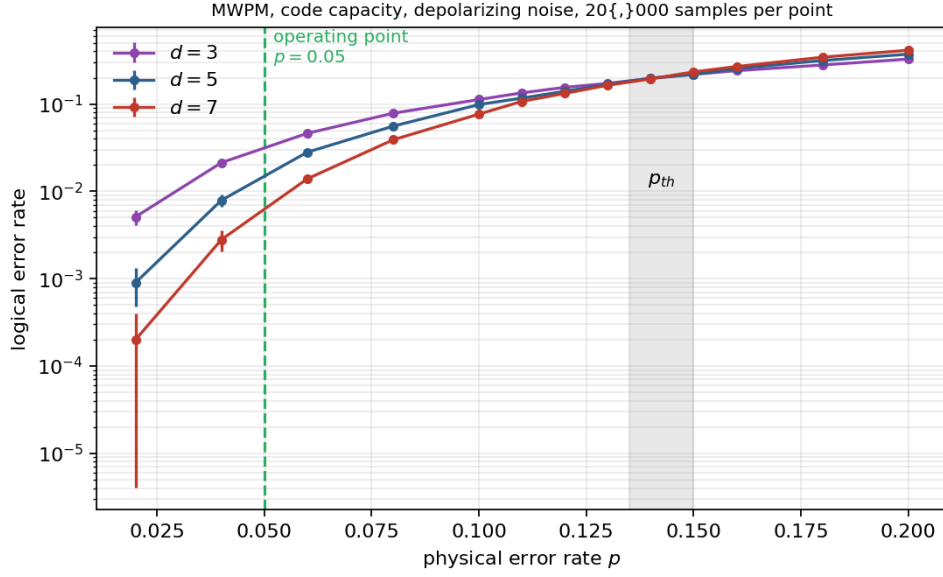


**Figure 8.1:** MWPM logical error rate versus physical error rate at  $d = 3, 5, 7$  (code capacity, depolarizing noise, 20,000 samples per point). The curves cross at  $p_{th} \approx 0.14$  (shaded band); the operating point  $p = 0.05$  of the headline comparison sits well below threshold, where increasing distance suppresses the LER and a margin over MWPM is meaningful.

## 8.2 Single-shot code capacity

Table 8.1 reports the headline comparison at  $d = 5$ ,  $p = 0.05$ , over  $5 \times 20,000$  evaluation samples per decoder with the exception of the imitation decoder which is evaluated on  $5 \times 5,000$  evaluation samples.

| Decoder           | LER           | Outcome vs. MWPM                                |
|-------------------|---------------|-------------------------------------------------|
| MWPM              | 0.0159        | Baseline                                        |
| <b>Supervised</b> | <b>0.0104</b> | <b>Beats (<math>\sim 35\%</math> reduction)</b> |
| Atomic            | 0.0181        | Ties/Slightly underperforms                     |
| Imitation         | 0.0674        | Underperforms ( $\sim 4x$ MWPM)                 |

**Table 8.1:** Single-shot logical error rate at distance  $d = 5$ , physical error rate  $p = 0.05$  under depolarizing noise

Three observations, one per row, each closing a prediction made in the theoretical chapters.

**The supervised decoder beats MWPM and reaches the maximum-likelihood optimum** The 35% relative reduction in LER is a significant result. To verify that this gain is exactly the  $Y$ -correlation margin and not an artefact, we computed the exact decoders at  $d = 3$ , where brute force is feasible: at distance  $d = 3$  the code has only 9 data qubits, so the set of possible Pauli errors is small enough to enumerate completely:  $4^9 = 262,144$  configurations. For each one we know its exact probability under depolarizing noise (each qubit is left alone with probability  $1 - p$ , or hit by  $X$ ,  $Y$  or  $Z$  with probability  $p/3$  each), its syndrome, and its logical class. This allows us to build the best possible decoder by brute force: given an observed syndrome, list every error configuration compatible with it, add up the probabilities of those belonging to each of the four logical classes  $\{I, X, Z, Y\}$ , and answer with the class that collected the most probability. No decoder, learned or classical, can do better than this on average; it is maximum-likelihood (ML) decoding by definition.

We built two versions of it. The joint ML decoder uses the true depolarizing probabilities, where a single  $Y$  error costs  $p/3$ . The per-sector ML decoder deliberately repeats MWPM’s simplification: it treats the  $X$  and  $Z$  components as two independent channels, so a  $Y$  error is priced as two separate coincidences. Comparing the two isolates exactly what is being tested: any gap between them can only come from the  $X$ - $Z$  correlations that  $Y$  errors introduce. The outcome at  $p = 0.05$  is clear-cut: joint ML achieves LER 0.0223, per-sector ML is statistically indistinguishable from MWPM (0.0346). The entire available margin at this distance therefore lives in the  $Y$  correlations, and since the learned  $d = 5$  supervised decoder obtains a comparable relative reduction over MWPM ( $-35\%$ ), we conclude it has effectively learned maximum-likelihood decoding, not a lucky heuristic.

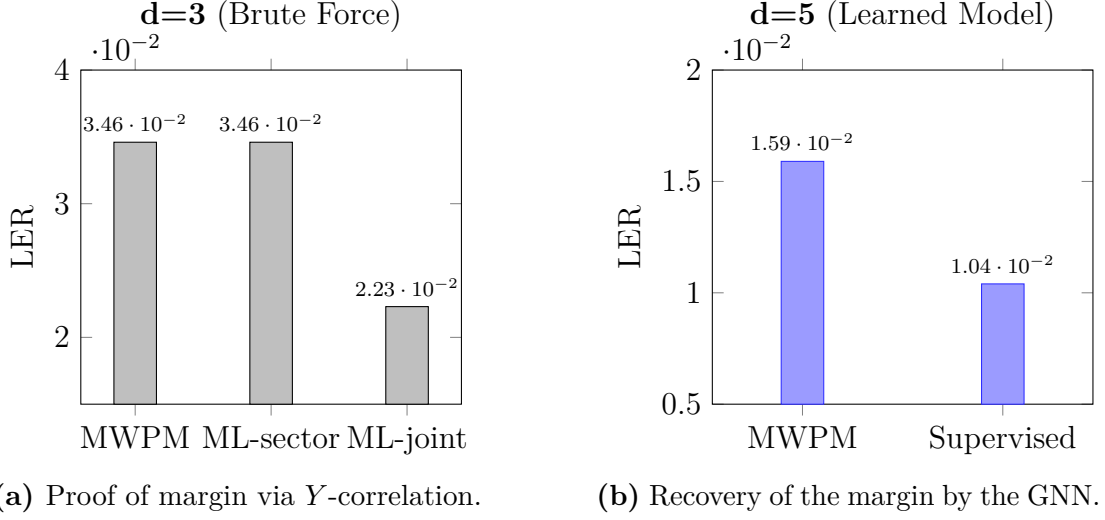


**Figure 8.2:** Comparison of Logical Error Rates (LER). Left: Exact ML decoding at  $d = 3$  shows that the gain is strictly due to joint  $X - Z$  reasoning. Right: The  $d = 5$  supervised model achieves a similar relative reduction.

**The atomic agent slightly underperforms MWPM** The atomic agent scores below MWPM (+14%). Doubling the hidden width of the Q-network ( $128 \rightarrow 256$ ) leaves the LER unchanged, confirming that the ceiling is the greedy syndrome-nulling objective, so the minimum-weight attractor of Chapter 4, and not model capacity.

**The imitation decoder is bounded by its teacher** By construction it cannot exceed MWPM, and sequential compounding of small disagreements makes it land below: in the single-shot setting it lands at roughly four times the MWPM error rate; in the multi-round setting of Section 8.3 (Table 8.2) the gap is  $\approx 2.3\times$  at 20 rounds, compounding with the horizon. This number is not a failure of the architecture, because the same backbone is the substrate of the readout-regime decoder of Section 8.5, but of the paradigm: pure imitation means that the model inherits the teacher’s ceiling.

### 8.3 Multi-round without measurement noise: the attractor regime

We now let errors accumulate over many measurement rounds (still with perfect measurements). Table 8.2 compares plain MWPM with the correlated matching

decoder introduced in Section 1.4, a  $Y$ -aware version of MWPM [59] implemented in this thesis, across three noise/round settings.

| $p$  | Rounds | MWPM LER | Correlated LER | Imitation LER |
|------|--------|----------|----------------|---------------|
| 0.05 | 20     | 0.1045   | 0.0865         | 0.2450        |
| 0.07 | 50     | 0.4350   | 0.4015         | 0.6029        |
| 0.02 | 500    | 0.5270   | 0.4655         | 0.7050        |

**Table 8.2:** Multi-round LER at distance  $d = 5$  with perfect measurements. Baselines: 2000 episodes per point; imitation decoder: 1000, 700 and 200 episodes respectively, on the same generator.

The third column shows how the imitation decoder performs in a closed-loop setting. After 20 steps, the model’s error rate is about  $2.3\times$  higher than the MWPM baseline. Small mistakes made at each step add up over time, exactly as predicted by the training analysis in Section 6.5.1.

When the number of steps increases (50 or 500), the gap between the model and the baseline shrinks. This occurs not because the model improves, but because all decoders eventually approach a random error rate of approximately 0.75. At this level, all methods fail, making their differences invisible. The imitation decoder never outperforms the baseline at any point. This confirms that a model trained only to copy an expert remains limited, especially as errors pile up during the process.

The  $Y$ -correlation margin survives the multi-round setting, and consequently, the fact that our learned sequential decoders do not beat MWPM here is not because the margin disappears, but because the sequential-flip formulation cannot reach it: an agent that emits physical flips step by step is pulled into the minimum-weight attractor, while the margin is only accessible to decoders that reason jointly over sectors and classes.

The mechanism behind the attractor deserves one concrete paragraph, because it explains every failed alternative tried during this work. A decoder that flips more qubits than strictly necessary can still silence the syndrome, but every superfluous flip is a real physical error added to the lattice. A decoder that silences the syndrome with a heavier correction than necessary differs from the minimum-weight one by closed loops of flips: those that are stabilizer products are harmless, but those that stretch between boundaries are logical operators, and the heavier the pairing, the likelier it is to have taken a defect around the wrong side of the lattice. Minimum-weight correction is precisely the policy that makes the wrong homology choice least likely per step, which is why every learned policy that discovered new correction patterns without minimizing weight ended up choosing wrong classes more often and paying a worse LER



**Figure 8.3:** The cost of non-minimality: MWPM versus a random-pairing decoder that silences the same syndromes with heavier corrections ( $d = 5, p = 0.05, 6000$  shots, same error samples). A 16% average weight excess translates into a  $5\times$  higher logical error rate: the extra weight means a higher probability of pairing defects along the wrong homology class

## 8.4 Reinforcement refinement

Section 7.3 described the two failure modes of fine-tuning in the noiseless regime; here we document them. Across learning-rate and reward-shaping sweeps, as we have already said, every run fell in one of two options: stall, policy and LER numerically frozen at the imitation initialization for the entire budget, and forgetting, training score decays, validation LER worsens monotonically, and the policy loses the minimum-weight discipline without acquiring a substitute. No configuration produced a statistically significant LER improvement over the initialization.

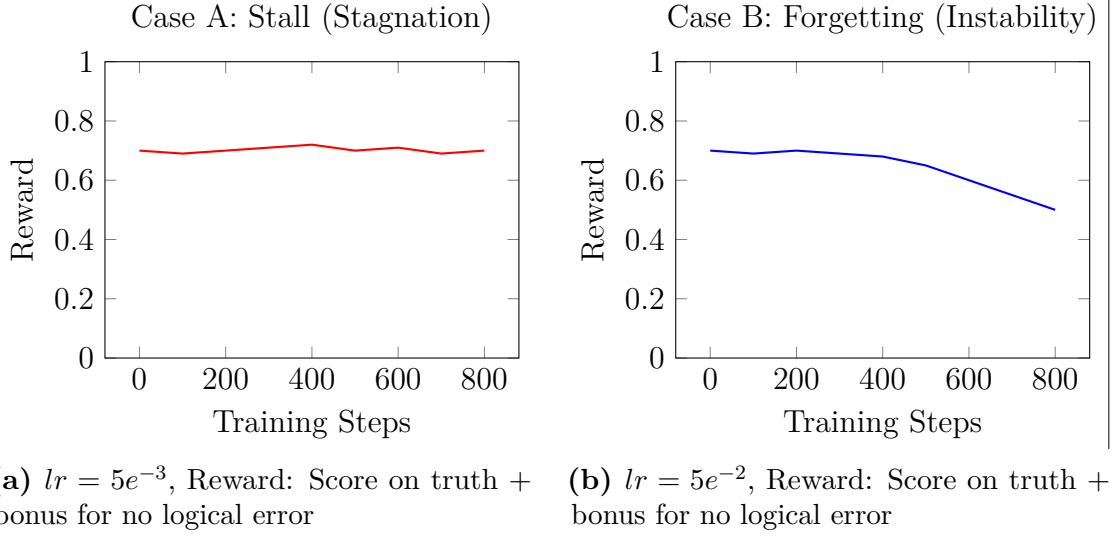


**Figure 8.4:** Learning curve analysis. (a) Stagnation due to under-fitting/learning rate too low. (b) Catastrophic forgetting due to aggressive gradient updates.

These runs constitute the empirical case for the central negative claim of this thesis: **in the absence of measurement noise, imitation of MWPM followed by policy-gradient refinement cannot beat MWPM, because MWPM is a local optimum of the objective the refinement optimizes.** The positive counterpart is Table 8.1: changing the objective (joint class prediction) beats MWPM with a smaller model and simpler training. The reward function is linked with the LER; it is the sum between the score on the truth and a bonus for not having a logical error.

## 8.5 The phenomenological regime: readout errors

### 8.5.1 Training dynamics of the lookahead decoder

Figure 8.5 shows the score curve of the reference training run of the lookahead decoder (1000 episodes,  $d = 5$ , base level of the curriculum:  $p = 0.003$ ,  $p_{ro} = 0.002$ ).

The curve has three regimes, whose boundaries coincide exactly with the training schedule:

- **Guided (episodes 1–300, top- $k$  active):** the block-average score climbs from 0.55 to 0.765, with a peak moving average (window 20) of  $\approx 0.91$ ; perfect-step matches reach 42/50 per episode and the loss is stable ( $\sigma \approx 0.09$ ).

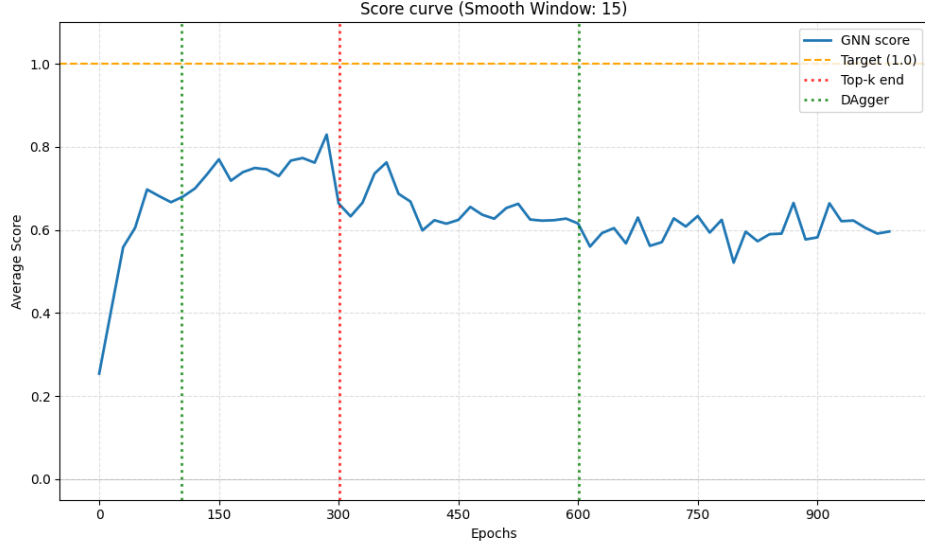


**Figure 8.5:** Smoothed per-episode score of the lookahead decoder. Green dotted lines: DAGger events; red dotted line: end of the top- $k$  constraint (episode 300). The three regimes discussed in the text are clearly visible.

Under expert guidance, the network learns the task while managing the readout errors: the counter of corrections erroneously applied on transient (readout-only) defects is essentially zero, direct evidence that the lookahead works;

- **Ramp (episodes 300–600):** as the DAGger coefficient ramps and the environment increasingly executes the model’s own actions, the score degrades from 0.69 to 0.63; full autonomy is reached at episode  $\approx 603$ ;
- **Autonomous (episodes 600–1000):** the score plateaus at 0.60–0.63 with no trend, having a linear fit slope  $-4.3 \times 10^{-5}/\text{episode}$ ,  $p = 0.26$ , loss variance quadruples ( $\sigma 0.09 \rightarrow 0.36$ ), and the regime is bimodal: good episodes near 0.7 alternate with collapses to 0.26–0.45 accompanied by loss spikes of 1.7–2.5. The curriculum never promoted past the base noise level;

### 8.5.2 Diagnosis: compounding, not calibration

Two hypotheses could explain the autonomous plateau: a miscalibrated decision threshold on the per-qubit logits, or a genuine inability to operate in closed loop. A threshold sweep on the final checkpoint (closed-loop evaluation, 20 episodes per threshold, thresholds 0.35-0.65) is flat: scores range 0.59-0.68 with no structure. Calibration is ruled out. The remaining explanation, consistent with the bimodality,

is **closed-loop compounding**: the model decodes well on the expert’s state distribution but does not recover from its own small mistakes; once a trajectory drifts off-distribution, errors of both kinds, missed persistent defects and their cascading consequences, accumulate until the episode collapses. This is the classic covariate-shift failure of sequential imitation [53], here surviving the DAgger ramp because the ramp teaches the model to act alone but never to return from bad states it has not visited under guidance.

Comparing this run with the RL fine-tuning runs of Section 8.4, they seem to share a trend, but this is not true; here we have a score that accumulates all kinds of missed flips and wrong flips, resulting in a hard metric to master. In addition to this, here the model starts from an absolute ignorance about the problem and, although the training is every time more difficult because of the DAgger ramp and the end of top-k, the model is still learning and improving its score, without declining. In the RL fine-tuning runs, instead, the model starts from a good initialization and it is not able to improve its score (reward), which is the only thing we expect from it.

### 8.5.3 Logical error rate under readout noise

The final judgement of the lookahead decoder is its LER against the worst and best MWPM baselines in this context: the naive MWPM, which does not know how to handle readout errors, and the oracle MWPM, which avoids handling readout errors because it works with the true syndrome history. Table 8.3 reports the LER of the three decoders at distance  $d = 5$  under readout noise, evaluated over 20 rounds.

| $(p, p_{ro})$  | MWPM-naive | MWPM-oracle | Lookahead GNN |
|----------------|------------|-------------|---------------|
| (0.01, 0.01)   | 0.6890     | 0.0105      | 0.3775        |
| (0.01, 0.005)  | 0.5945     | 0.0114      | 0.4056        |
| (0.003, 0.002) | 0.5941     | 0.0013      | 0.2880        |

**Table 8.3:** Multi-round LER under readout noise at distance  $d = 5$ , evaluated over 20 rounds.

The results presented in Table 8.3 and Figure 8.6 show a clear separation between the lookahead decoder, the MWPM-naive and the MWPM-oracle performances when readout noise ( $p_{ro}$ ) is introduced. The Lookahead GNN consistently outperforms the MWPM-naive approach across all tested scenarios, which validates the strategy used to let the model learn to integrate temporal information and handle the readout noise.

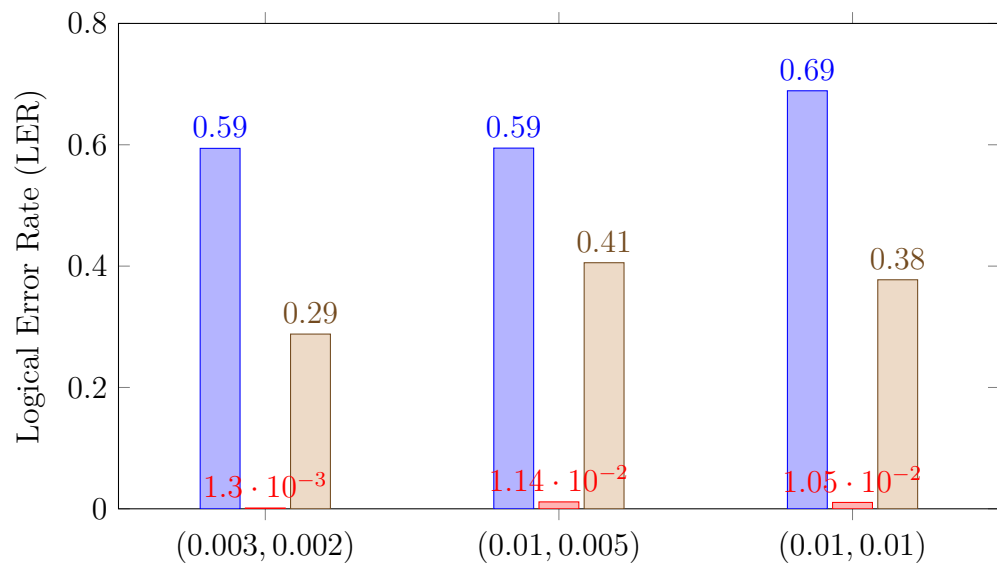


**Figure 8.6:** LER comparison

Nevertheless, the performance gap relative to the MWPM-oracle remains substantial, raising crucial questions regarding the task’s complexity:

1. **Beyond Static Processing:** The failure of MWPM-naive, which treats each round as an isolated instance, shows that with  $p_{ro} > 0$ , the evidence required for optimal correction is not contained solely within the instantaneous syndrome. The Lookahead GNN partially bridges this gap by learning to integrate the temporal history of syndromes, effectively transforming a static decoding problem into an estimation task of the hidden state;
2. **The Cost of Inference:** The remaining difference compared to the MWPM-oracle should not be viewed as a failure of the model, but as a consequence of the uncertainty inherent in the measurement process. The oracle operates on syndromes free from readout errors, with perfect knowledge, whereas the GNN must infer the system state despite the noise. The model’s success in lowering the LER, even in the presence of this ambiguity, confirms that it has learned a policy aware of error propagation, capable of distinguishing between transient noise and persistent logical errors;
3. **Compounding Effects:** The data suggests that as the error rate increases ( $p = 0.01$ ), the advantage of the Lookahead over the naive approach becomes more pronounced; yet, the distance to the oracle also tends to widen. This trend points to compounding effects: in high-noise regimes, past inference errors propagate uncertainty that the decoder, regardless of its expertise, cannot fully neutralize within the limited temporal horizon of the lookahead window;

Figure 8.7 completes the picture by sweeping the horizon at the highest noise point: the lookahead advantage over the naive baseline decays monotonically with the number of rounds, from  $2.2\times$  at 5 rounds to  $1.5\times$  at 40; a per-step competence that erodes as self-induced deviations accumulate, which is the quantitative signature of the closed-loop compounding diagnosed in Section 8.5.2.

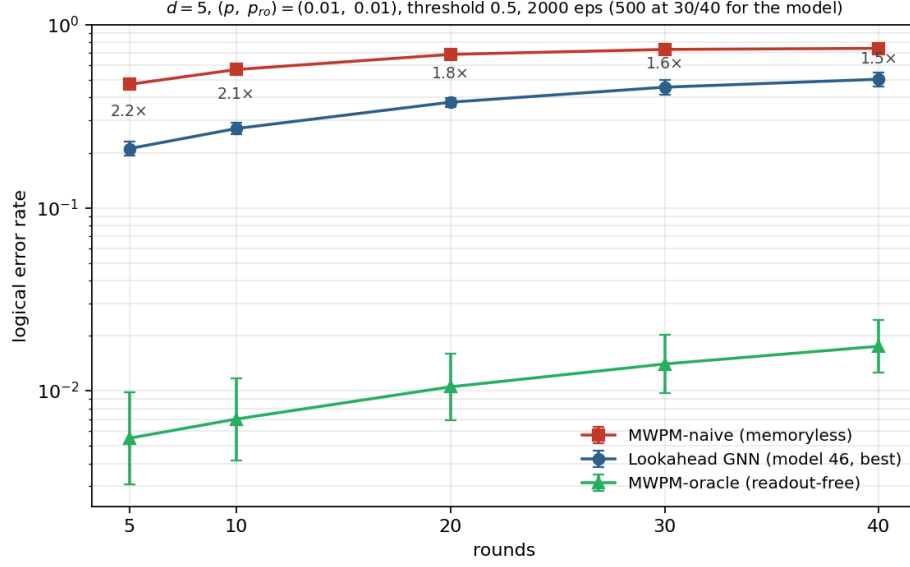


**Figure 8.7:** LER versus number of rounds at  $(p, p_{ro}) = (0.01, 0.01)$ ,  $d = 5$ , log scale. Annotations report the naive-to-model ratio at each horizon: the lookahead advantage decays from  $2.2\times$  at 5 rounds to  $1.5\times$  at 40, the closed-loop compounding signature. Baselines: 2000 episodes per point; lookahead model: 2000 episodes (500 at 30 and 40 rounds).

# Chapter 9

## Discussion

### 9.1 One principle, two margins

The results of Chapter 8 admit a single unifying reading: a learned decoder beats MWPM exactly where, and only where, MWPM discards exploitable structure. Two such margins were identified in Chapter 3 and both were confirmed empirically. The first is spatial: by decoding  $X$  and  $Z$  on separate graphs, MWPM cannot price  $Y$  errors correctly; the supervised joint-class decoder, whose objective is maximum likelihood over both sectors, recovers this margin in full (0.0104 vs 0.0159 at  $d = 5$ ,  $p = 0.05$ ), matching the exact optimum computed at  $d = 3$ . The second is temporal: by treating every defect as real, round-per-round MWPM cannot distinguish persistent data faults from transient readout faults; the lookahead decoder demonstrably reads this signature under guidance (near-zero corrections applied on transient defects throughout training). Conversely, wherever no discarded structure exists, learning gains nothing: the atomic agent never outperforms MWPM at any capacity, and reinforcement refinement of the imitation policy stalls or forgets.

This framing turns what could read as a mixed scorecard into a falsifiable statement about the geometry of the problem, and it prescribes where to invest next: not in bigger networks on solved regimes, but in objectives and observations that expose the remaining discarded structure.

### 9.2 The minimum-weight attractor as a design constraint

The negative results are as informative as the positive ones. Every sequential-flip agent trained in this work, such as imitation, imitation+RL, value-based atomic, converged to, and could not exceed, minimum-weight behaviour. Chapter 4 explains

why: under perfect measurements, the correction that silences the syndrome with the fewest flips is also the one least likely to have paired defects along the wrong homology class, so greedy syndrome nulling has MWPM as a local optimum, and gradient methods initialized there stay there or fall off. The practical design constraint follows: **the action space and objective determine the ceiling before training begins**. The supervised decoder wins not because it is a better network, in fact it is the smallest and the simplest, it wins because predicting a homology class is an objective whose optimum is maximum-likelihood decoding rather than minimum-weight decoding. We consider making this constraint explicit a contribution of the thesis, since much of the learned-decoder literature reports sequential agents matching MWPM without identifying the attractor that makes exceeding it impossible in that formulation.

### 9.3 Closed-loop compounding: the real frontier of the readout regime

In the phenomenological regime, the architecture performs effectively. The lookahead window allows the network to distinguish between readout errors and data faults. In closed-loop settings, it roughly halves the error rate of the memoryless baseline. Consequently, the main challenge has shifted: the issue is no longer about detecting errors, but about the recovery process.

Three observations suggest that the remaining failures are due to closed-loop error compounding:

1. **Performance decay over time:** The advantage of our model over the naive baseline drops as the time horizon increases, moving from a  $2.2\times$  improvement at 5 rounds to  $1.5\times$  at 40 (Figure 8.7). This gradual loss of accuracy is a clear sign that errors are accumulating, rather than a failure to perceive the state correctly;
2. **Threshold stability:** Our threshold sweep tests show that simple recalibration does not resolve the issue;
3. **Specialization:** Out-of-distribution tests show that the policy is a specialist, performing well only on states similar to those seen during training;

This indicates that the problem lies in the training distribution, not in the model’s capacity or architecture. This finding points toward clear solutions (Section 10.2): training the model on dirty but recoverable states, limiting autonomy during self-play, and selecting checkpoints based on autonomous, rather than guided, performance. Unlike the noiseless regime, where the limitation was fundamental (a local optimum), we believe this obstacle is operational and, therefore, surmountable.

## 9.4 Relation to prior work

Our conclusions align with and help explain existing literature. Previous reinforcement learning (RL) decoders on toric and surface codes [30, 60] show performance similar to MWPM. Our attractor argument explains this: because these decoders use sequential-flip designs, they naturally move toward the minimum-weight attractor, which leads to parity with MWPM rather than superior performance.

In contrast, supervised neural decoders have outperformed MWPM baselines when trained on likelihood-based objectives with complex noise models. This includes early work like restricted Boltzmann machines [61] and feed-forward classifiers [62], as well as modern graph networks [63] and AlphaQubit’s recurrent transformer [64]. All these successes support the principle of exploiting discarded structure.

Regarding classical methods, our results on correlated matching support Fowler’s re-weighting scheme [59]. Since some performance gains from  $Y$ -correlations are possible without machine learning, a learned decoder must perform significantly better to justify its development cost.

Compared to the existing literature, this thesis contributes the following:

- An explicit POMDP formulation that uses Bayesian local marginals as features, rather than relying on raw syndromes alone;
- The identification and empirical isolation of the minimum-weight attractor;
- A fixed-lookahead observation design for the readout noise regime, evaluated against properly matched baselines;
- A modular, Stim-based pipeline with temporally compressed trajectories;

## 9.5 Limitations

Honesty requires listing what this work does not establish:

- **One distance:** all experiments run at  $d = 5$  ( $d = 3$  is used only for the exact brute-force check). We do not know how the learned decoders behave as the code grows, neither their error rates nor their training cost. The architecture is built to scale (the marginals cost  $O(N)$  and the network size does not depend on  $d$ ), but built to scale is a design property, not a measured result;
- **Distance from the oracle:** beating the naive baseline is not the same as decoding well in absolute terms: the gap to the readout-free oracle remains between one and two orders of magnitude at every measured point. The perception problem is solved; the recovery problem is open;

- **Specialization:** the lookahead decoder is a specialist. Outside its training regime, with readout noise switched off, it performs far worse than the plain imitation decoder built on the same backbone. Its skill does not transfer; it is the direct imprint of its training distribution;

| $p$ (readout-free) | Lookahead GNN | Imitation decoder |
|--------------------|---------------|-------------------|
| 0.01               | 0.5550        | 0.0500            |
| 0.05               | 0.7360        | 0.2450            |

**Table 9.1:** Out-of-distribution ablation: LER of the lookahead decoder and the plain imitation decoder, both evaluated at  $p_{ro} = 0, 20$  rounds. Despite sharing the same backbone, the lookahead decoder loses one to two orders of magnitude of performance outside its training regime

- **The supervised decoder’s win is single-shot:** Its extension to the multi-round setting was not evaluated, and nothing in the present data guarantees that the  $Y$ -correlation margin it collects would survive sequential accumulation, even though the correlated-matching results suggest the margin itself does survive;

## Chapter 10

# Conclusions and Future Work

### 10.1 Conclusions

This thesis set out to determine where a learned decoder can outperform Minimum Weight Perfect Matching on the surface code, and built an architecture, a heterogeneous graph neural network over a factor graph enriched with actively maintained Bayesian local marginals, framed as a POMDP, to exploit those marginals. Its findings are four.

1. **The margin is structural, not generic:** A learned decoder beats MWPM exactly where MWPM discards exploitable structure. The supervised joint-class decoder, optimizing maximum likelihood over both Pauli sectors, reduced the single-shot logical error rate by 35% at  $d = 5$ ,  $p = 0.05$  ( $0.0104 \pm 0.0006$  vs  $0.0159 \pm 0.0012$ , significant beyond  $5\sigma$ ), matching the exact maximum-likelihood optimum verified at  $d = 3$ ;
2. **Sequential syndrome nulling has MWPM as a ceiling:** Every sequential-flip agent, such as imitation, imitation+RL, atomic, converged to minimum-weight behaviour and no further, independently of capacity. We identified the mechanism, called the minimum-weight attractor, where heavier corrections mean likelier wrong homology choices;
3. **The temporal margin of readout errors is readable:** Under measurement noise, a fixed two-round lookahead lets the network separate transient readout faults from persistent data faults, as shown by near-zero spurious corrections on transient defects throughout guided training;

4. **The current frontier is recovery, not perception:** In closed-loop operation the lookahead decoder roughly halves the error rate of the memoryless baseline, but its advantage decays with the horizon (from  $2.2\times$  at 5 rounds to  $1.5\times$  at 40) and the gap to the readout-free oracle remains over an order of magnitude. A flat threshold sweep rules out calibration and locates the cause in compounding: the policy does not recover from its own off-distribution states;

Beyond the numbers, the thesis leaves a methodological artefact: a modular, Stim-based pipeline with temporally compressed trajectories, a unified multi-decoder evaluation harness with information-matched baselines, and a documented every win and loss, because negative and corrected results are where most of this project’s understanding came from.

## 10.2 Future work

The next steps are ordered from immediate to structural.

**Completing the readout-regime evaluation:** The corrected training run and its 20-round LER grid now exist (Table 8.3); what remains is consolidation: multiple evaluation seeds at  $\sim 10^4$  episodes per point, a completed curriculum (the best checkpoint stops at  $p = 0.005$ , below the two hardest evaluation points), longer horizons, and, above all, the space-time (3D) MWPM baseline, the standard classical treatment of measurement noise, against which the lookahead advantage must ultimately be judged.

**Recovery-oriented training:** The compounding signature confirmed by the corrected run (the advantage decaying from  $2.2\times$  to  $1.5\times$  between 5 and 40 rounds) prescribes attacking the state distribution directly: initialize a fraction of training episodes from lightly corrupted states, like residual dirt sampled from the model’s own past failures, cap the DAgger coefficient below 1 so a thin stream of expert corrections keeps visiting recovery states, and slow the autonomy ramp. The goal is a policy that has practiced returning, not only acting.

**A learned syndrome cleaner:** An alternative decomposition, prototyped during this work, separates perception from correction: a standalone supervised detector predicts per-ancilla, per-round readout probabilities from the raw detector-event sequence (this component already trains successfully), the observed syndrome is cleaned at high precision, and the frozen noiseless backbone decodes the cleaned input. Integrating and benchmarking this two-stage decoder against the end-to-end

lookahead model would separate how much of the problem is detection and how much is decoding.

**Closing the multi-round question for the class decoder:** The supervised decoder’s single-shot victory raises an unanswered question: can joint-class prediction be lifted to the sequential setting, and does the  $Y$  margin then survive accumulation, as the correlated-matching results suggest it should?

**Scaling and realism:** Two axes remain untouched: distance (training and evaluating at  $d = 7, 9$  to measure how the learned decoders’ advantage and cost scale, the marginals being  $O(N)$  by design) and noise realism (circuit-level noise, where correlated gate faults create exactly the kind of structure, discarded by standard matching, that this thesis’s principle predicts a learned decoder should exploit). The latter is also the setting of the strongest published learned decoders, making it the natural arena for a direct comparison.

**Latency:** Finally, deployment on real hardware would require profiling the per-step inference cost of the HeteroGNN against the microsecond-scale round time of superconducting platforms, and reconciling the two-round lookahead latency with the sliding-window schemes used by streaming classical decoders.

# Bibliography

- [1] Peter W. Shor. «Scheme for reducing decoherence in quantum computer memory». In: *Physical Review A* 52.4 (1995), R2493–R2496. DOI: 10.1103/PhysRevA.52.R2493 (cit. on pp. 2, 16, 18).
- [2] Daniel Gottesman. «Stabilizer Codes and Quantum Error Correction». PhD thesis. California Institute of Technology, 1997. DOI: 10.48550/arXiv.quant-ph/9705052 (cit. on pp. 2, 13, 14, 23).
- [3] Alexei Yu. Kitaev. «Fault-tolerant quantum computation by anyons». In: *Annals of Physics* 303.1 (2003), pp. 2–30. DOI: 10.1016/S0003-4916(02)00018-0 (cit. on pp. 2, 20).
- [4] Sergey B. Bravyi and Alexei Yu. Kitaev. *Quantum codes on a lattice with boundary*. arXiv preprint quant-ph/9811052. 1998. DOI: 10.48550/arXiv.quant-ph/9811052 (cit. on p. 2).
- [5] Dorit Aharonov and Michael Ben-Or. «Fault-Tolerant Quantum Computation with Constant Error Rate». In: *SIAM Journal on Computing* 38.4 (2008), pp. 1207–1282. DOI: 10.1137/S0097539799359385 (cit. on p. 2).
- [6] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. «Resilient Quantum Computation». In: *Science* 279.5349 (1998), pp. 342–345. DOI: 10.1126/science.279.5349.342 (cit. on p. 2).
- [7] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. «Surface codes: Towards practical large-scale quantum computation». In: *Physical Review A* 86.3 (2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324 (cit. on pp. 3, 4, 13, 17, 20–22, 24).
- [8] Pavithran Iyer and David Poulin. «Hardness of decoding quantum error-correcting codes». In: *IEEE Transactions on Information Theory* 61.9 (2015), pp. 5209–5223 (cit. on p. 3).
- [9] Barbara M Terhal. «Quantum error correction for quantum memories». In: *Reviews of Modern Physics* 87.2 (2015), p. 307 (cit. on pp. 3, 4).

- [10] Jack Edmonds. «Paths, trees, and flowers». In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/CJM-1965-045-4 (cit. on p. 4).
- [11] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. «Topological quantum memory». In: *Journal of Mathematical Physics* 43.9 (2002), pp. 4452–4505. DOI: 10.1063/1.1499754 (cit. on pp. 4, 21, 25).
- [12] Oscar Higgott. «PyMatching: A Python Package for Decoding Quantum Codes with Minimum-Weight Perfect Matching». In: *ACM Transactions on Quantum Computing* 3.3 (2022), p. 16. DOI: 10.1145/3505637 (cit. on pp. 4, 78).
- [13] Oscar Higgott and Craig Gidney. «Sparse Blossom: correcting a million errors per core second with minimum-weight matching». In: *Quantum* 9 (2025), p. 1600. DOI: 10.22331/q-2025-01-20-1600 (cit. on pp. 4, 78).
- [14] Nicolas Delfosse and Naomi H. Nickerson. «Almost-linear time decoding algorithm for topological codes». In: *Quantum* 5 (2021), p. 595. DOI: 10.22331/q-2021-12-02-595 (cit. on p. 4).
- [15] Joschka Roffe, David R. White, Simon Burton, and Earl Campbell. «Decoding across the quantum low-density parity-check code landscape». In: *Physical Review Research* 2.4 (2020), p. 043423. DOI: 10.1103/PhysRevResearch.2.043423 (cit. on p. 4).
- [16] Rajeev Acharya et al. «Quantum error correction below the surface code threshold». In: *Nature* 638.8052 (2025), pp. 920–926. DOI: 10.1038/s41586-024-08449-y (cit. on pp. 4, 21).
- [17] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th Anniversary. Cambridge University Press, 2010. ISBN: 978-1-107-00217-3. DOI: 10.1017/CB09780511976667 (cit. on pp. 9, 11–18).
- [18] Richard P. Feynman. «Simulating Physics with Computers». In: *International Journal of Theoretical Physics* 21.6–7 (1982), pp. 467–488. DOI: 10.1007/BF02650179 (cit. on pp. 11, 17).
- [19] John S. Bell. «On the Problem of Hidden Variables in Quantum Mechanics». In: *Reviews of Modern Physics* 38.3 (1966), pp. 447–452. DOI: 10.1103/RevModPhys.38.447 (cit. on p. 12).
- [20] Keisuke Fujii. *Quantum Computation with Topological Codes: From Qubit to Topological Fault-Tolerance*. Vol. 8. SpringerBriefs in Mathematical Physics. Springer, 2015. DOI: 10.1007/978-981-287-996-7 (cit. on pp. 13, 21, 25).

- [21] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. «Elementary gates for quantum computation». In: *Physical Review A* 52.5 (1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457 (cit. on p. 13).
- [22] William K. Wootters and Wojciech H. Zurek. «A single quantum cannot be cloned». In: *Nature* 299.5886 (1982), pp. 802–803. DOI: 10.1038/299802a0 (cit. on p. 17).
- [23] Dennis Dieks. «Communication by EPR devices». In: *Physics Letters A* 92.6 (1982), pp. 271–272. DOI: 10.1016/0375-9601(82)90084-6 (cit. on p. 17).
- [24] David Deutsch. «Quantum theory, the Church–Turing principle and the universal quantum computer». In: *Proceedings of the Royal Society of London A* 400.1818 (1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070 (cit. on p. 18).
- [25] David Deutsch and Richard Jozsa. «Rapid solution of problems by quantum computation». In: *Proceedings of the Royal Society of London A* 439.1907 (1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167 (cit. on p. 19).
- [26] Peter W. Shor. «Algorithms for quantum computation: discrete logarithms and factoring». In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (cit. on p. 19).
- [27] Lov K. Grover. «A fast quantum mechanical algorithm for database search». In: *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*. ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866 (cit. on p. 19).
- [28] Yu Tomita and Krysta M. Svore. «Low-distance Surface Codes under Realistic Quantum Noise». In: *Physical Review A* 90.6 (2014), p. 062320. DOI: 10.1103/PhysRevA.90.062320 (cit. on p. 21).
- [29] Anthony R. O’Rourke and Simon Devitt. «Compare the Pair: Rotated vs. Unrotated Surface Codes at Equal Logical Error Rates». In: *Physical Review Research* 7.3 (2025), p. 033074. DOI: 10.1103/PhysRevResearch.7.033074 (cit. on p. 21).
- [30] 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 (cit. on pp. 24, 25, 33, 73, 92).
- [31] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018 (cit. on pp. 31, 32).

- [32] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. «Planning and acting in partially observable stochastic domains». In: *Artificial Intelligence* 101.1–2 (1998), pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X (cit. on p. 32).
- [33] R. Michael Tanner. «A recursive approach to low complexity codes». In: *IEEE Transactions on Information Theory* 27.5 (1981), pp. 533–547. DOI: 10.1109/TIT.1981.1056404 (cit. on p. 40).
- [34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. «The Graph Neural Network Model». In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605 (cit. on p. 40).
- [35] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. «Neural Message Passing for Quantum Chemistry». In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1263–1272 (cit. on p. 40).
- [36] Thomas N. Kipf and Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on pp. 41, 45).
- [37] William L. Hamilton, Rex Ying, and Jure Leskovec. «Inductive Representation Learning on Large Graphs». In: *Advances in Neural Information Processing Systems 30 (NeurIPS)*. 2017, pp. 1024–1034 (cit. on pp. 42, 45).
- [38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. «Graph Attention Networks». In: *International Conference on Learning Representations (ICLR)*. 2018 (cit. on pp. 42, 45).
- [39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. «How Powerful are Graph Neural Networks?» In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on pp. 42, 45).
- [40] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. «Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification». In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*. 2021, pp. 1548–1554. DOI: 10.24963/ijcai.2021/214 (cit. on pp. 43, 45).
- [41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *Advances in Neural Information Processing Systems 30 (NeurIPS)*. 2017, pp. 5998–6008 (cit. on p. 43).

- [42] Boris Weisfeiler and Andrei Lehman. «A reduction of a graph to a canonical form and an algebra arising during this reduction». In: *Nauchno-Tekhnicheskaya Informatsiya* 2.9 (1968), pp. 12–16 (cit. on p. 45).
- [43] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. «Weisfeiler and leman go neural: Higher-order graph neural networks». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4602–4609 (cit. on p. 45).
- [44] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. «Factor Graphs and the Sum-Product Algorithm». In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519. DOI: 10.1109/18.910572 (cit. on p. 47).
- [45] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. «Loopy Belief Propagation for Approximate Inference: An Empirical Study». In: *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*. 1999, pp. 467–475 (cit. on p. 48).
- [46] Qimai Li, Zhichao Han, and Xiao-Ming Wu. «Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning». In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 2018, pp. 3538–3545. DOI: 10.1609/aaai.v32i1.11604 (cit. on p. 49).
- [47] Uri Alon and Eran Yahav. «On the Bottleneck of Graph Neural Networks and its Practical Implications». In: *International Conference on Learning Representations (ICLR)*. 2021 (cit. on p. 49).
- [48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 49).
- [49] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. arXiv preprint arXiv:1607.06450. 2016. DOI: 10.48550/arXiv.1607.06450 (cit. on p. 49).
- [50] Matthias Fey and Jan Eric Lenssen. «Fast Graph Representation Learning with PyTorch Geometric». In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019 (cit. on p. 50).
- [51] Claude Elwood Shannon. «A Mathematical Theory of Communication». In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423 (cit. on p. 58).
- [52] Mark Towers et al. «Gymnasium: A Standard Interface for Reinforcement Learning Environments». In: *arXiv preprint arXiv:2407.17032* (2024). DOI: 10.48550/arXiv.2407.17032 (cit. on p. 61).

- [53] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. «A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning». In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 15. PMLR. 2011, pp. 627–635 (cit. on pp. 62, 86).
- [54] Hado van Hasselt, Arthur Guez, and David Silver. «Deep Reinforcement Learning with Double Q-Learning». In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2016, pp. 2094–2100. DOI: 10.1609/aaai.v30i1.10295 (cit. on p. 69).
- [55] Christopher J. C. H. Watkins and Peter Dayan. «Q-learning». In: *Machine Learning* 8.3–4 (1992), pp. 279–292. DOI: 10.1007/BF00992698 (cit. on p. 69).
- [56] Long-Ji Lin. «Self-improving reactive agents based on reinforcement learning, planning and teaching». In: *Machine Learning* 8.3–4 (1992), pp. 293–321. DOI: 10.1007/BF00992699 (cit. on p. 70).
- [57] Sergey Bravyi, Martin Suchara, and Alexander Vargo. «Efficient algorithms for maximum likelihood decoding in the surface code». In: *Physical Review A* 90.3 (2014), p. 032326. DOI: 10.1103/PhysRevA.90.032326 (cit. on p. 71).
- [58] Craig Gidney. «Stim: a fast stabilizer circuit simulator». In: *Quantum* 5 (2021), p. 497. DOI: 10.22331/q-2021-07-06-497 (cit. on p. 78).
- [59] Austin G. Fowler. *Optimal complexity correction of correlated errors in the surface code*. arXiv preprint arXiv:1310.0863. 2013. DOI: 10.48550/arXiv.1310.0863 (cit. on pp. 82, 92).
- [60] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath. «Quantum error correction for the toric code using deep reinforcement learning». In: *Quantum* 3 (2019), p. 183. DOI: 10.22331/q-2019-09-02-183 (cit. on p. 92).
- [61] Giacomo Torlai and Roger G. Melko. «Neural Decoder for Topological Codes». In: *Physical Review Letters* 119.3 (2017), p. 030501. DOI: 10.1103/PhysRevLett.119.030501 (cit. on p. 92).
- [62] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. «Decoding small surface codes with feedforward neural networks». In: *Quantum Science and Technology* 3.1 (2018), p. 015004. DOI: 10.1088/2058-9565/aa955a (cit. on p. 92).
- [63] Moritz Lange, Pontus Havström, Basudha Srivastava, Valdemar Bergentall, Karl Hammar, Olivia Heuts, Evert van Nieuwenburg, and Mats Granath. *Data-driven decoding of quantum error correcting codes using graph neural networks*. arXiv preprint arXiv:2307.01241. 2023. DOI: 10.48550/arXiv.2307.01241 (cit. on p. 92).

- [64] Johannes Bausch, Andrew W. Senior, Francisco J. H. Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, et al. «Learning high-accuracy error decoding for quantum processors». In: *Nature* 635 (2024), pp. 834–840. DOI: 10.1038/s41586-024-08148-8 (cit. on p. 92).