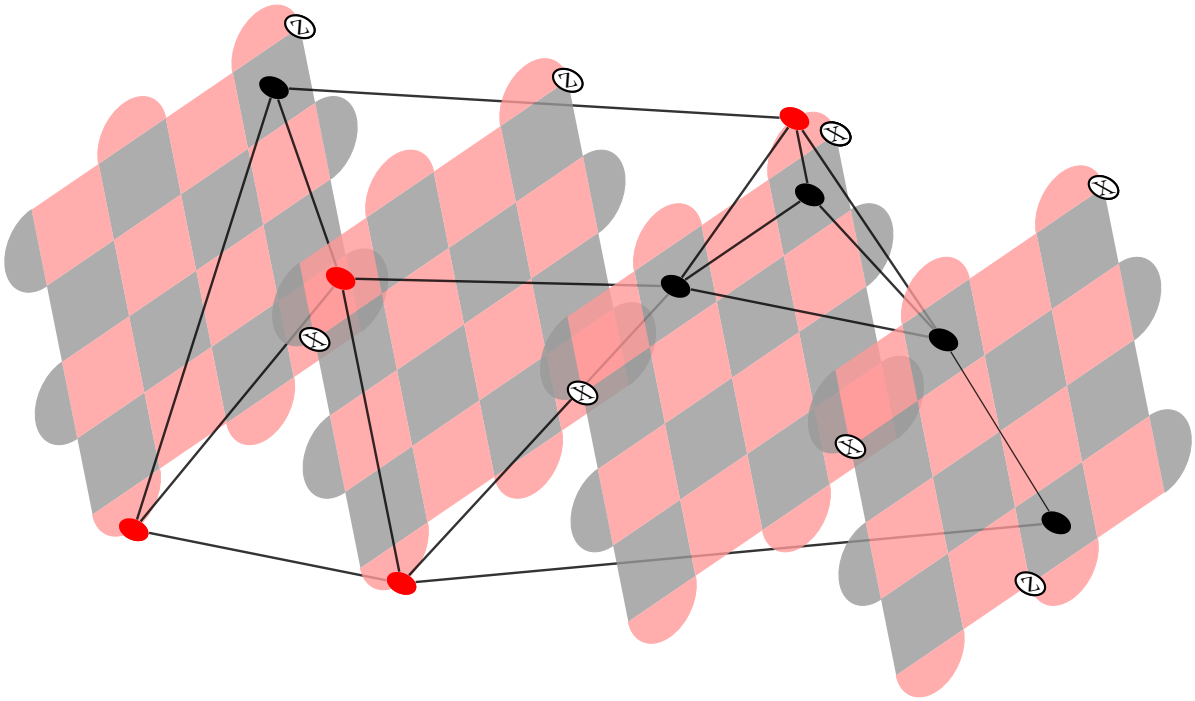
UNIVERSITY OF GOTHENBURG



Decoding the surface code using graph neural networks

Master's thesis in Master's Programme in Physics

MORITZ LANGE

DEPARTMENT OF PHYSICS

UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023
www.gu.se

Master's thesis 2023

Decoding the surface code using graph neural networks

MORITZ LANGE

UNIVERSITY OF
GOTHENBURG

Department of Physics
Quantum Computing and Quantum Machine Learning
University of Gothenburg
Gothenburg, Sweden 2023

Decoding the surface code using graph neural networks
MORITZ LANGE

Supervisor: Basudha Srivastava, Department of Physics, University of Gothenburg
Examiner: Mats Granath, Department of Physics, University of Gothenburg

Master's Thesis 2023
Department of Physics
University of Gothenburg
SE-412 96 Gothenburg

Cover: Mapping of surface code detection events to a graph representation.

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

Quantum error correction is essential to achieve fault-tolerant quantum computation in the presence of noisy qubits. Among the most promising approaches to quantum error correction is the surface code, thanks to a scalable two-dimensional architecture, only nearest-neighbor interactions, and a high error threshold. Decoding the surface code, i.e. finding the most likely error chain given a syndrome measurement outcome is a computationally complex task. Traditional decoders rely on classical algorithms, which, especially for larger systems, can be slow and may not always converge to the optimal solution. This thesis presents a novel approach to decoding the surface code using graph neural networks. By mapping the syndrome measurements to a graph and performing graph classification, we find that the graph neural networks can predict the most likely error configuration with high accuracy. Our results show that the GNN-based decoder outperforms the classic minimum weight perfect matching (MWPM) decoder in terms of accuracy. With a phenomenological noise model with depolarizing noise and perfect syndrome measurements, our networks beat MWPM up to code-size 15 across all relevant error rates. Furthermore, the GNN is capable of surpassing MWPM under circuit-level noise up to code size 7. We also show that training the network on repetition code data from a recent experiment [Google Quantum AI, Nature **614**, 676 (2023)] produces per-step error rates comparable to those achieved with a matching decoder specifically adapted to the error rates of the physical qubits. This indicates that graph neural network decoders are capable of learning the underlying error distribution on the qubits. Our findings advance the field of quantum error correction and provide a promising new direction for the development of efficient decoding algorithms.

Keywords: Quantum error correction, surface code, graph neural networks.

Acknowledgements

First and foremost, I want to express my gratitude towards Mats Granath who supported me throughout the project. I also thank Pontus Havström for explaining the details about his decoder. Furthermore, I am grateful for the discussions with Basudha Srivastava and Evert van Nieuwenburg as well as for the technical support from Hampus Linander and Viktor Rehnberg.

Computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at Chalmers Centre for Computational Science and Engineering (C3SE), partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

Moritz Lange, Gothenburg, June 2023

Contents

List of Figures

1
Introduction

Quantum computing is a rapidly evolving field with the potential to revolutionize the way we solve complex problems. In contrast to classical computing, which relies on bits that can either be 0 or 1, quantum computing uses quantum bits (qubits), which exist in a superposition of both 0 and 1. Quantum effects such as superposition, entanglement and interference give quantum computers an advantage over classical computers for certain types of problems, the most famous of those being Shor's algorithm that promises an exponential speedup for integer factorization [1] and simulating quantum systems [2]. However, this advantage is offset by the fragility of the qubits which are highly sensitive to their environment. Running algorithms on a quantum computer thus requires some sort of error correction technique.

Quantum error correction (QEC) is a technique used to counteract the effects of noise and other errors that occur during the operation of a quantum computer. Quantum error correction involves encoding the information of one logical qubit into multiple physical qubits in a way that allows for the detection and correction of errors. One of the most promising approaches to quantum error correction is the surface code. It is a practical implementation of a topological code, a class of error correction schemes that were introduced by Aleksei Kitaev and collaborators more than 20 years ago [3]. The surface code comprises a two-dimensional array of qubits to encode one logical qubit. Incomplete, so-called stabilizer measurements of the system are performed to detect errors. A decoding algorithm then finds the most likely error based on the measurement outcomes. The surface code has the advantage of being both fault-tolerant and scalable. It has recently been realized experimentally [4, 5], a promising proof of concept and an important milestone towards fault-tolerant quantum computation.

However, decoding the surface code is a computationally complex task that has to be solved fast and with high accuracy. Traditional decoding methods rely on approximate algorithms, such as minimum weight perfect matching (MWPM), which may not always converge to the optimal solution [6]. On the other hand, maximum-likelihood decoders achieve optimal accuracies but come at the expense of slow execution time [7–10]. A variety of machine learning-assisted decoding algorithms have been explored in the last decade [11–24], all displaying different strengths and weaknesses. In this thesis, we present a novel, data-driven approach to decoding the surface code using graph neural networks (GNNs). GNNs have been proven to be effective in a variety of tasks, including node classification, link prediction, and graph classification. The benefit of decoding the surface code with a neural network is that the prediction is fast after training, involving only a forward pass through the network. GNNs in particular scale favorably with the code size, i.e.

they are, in principle, suitable to decode surface codes of arbitrary size. By mapping surface measurement outcomes to a graph representation, the GNN can learn the underlying structure and patterns in the code and make more accurate predictions compared to traditional decoding methods. Our research shows that the GNN-based decoder outperforms MWPM in terms of accuracy for both perfect and imperfect stabilizer measurements. Our research advances the field of quantum error correction and provides a promising new direction for the development of efficient decoding algorithms. The use of GNNs for decoding the surface code represents a step towards the integration of quantum computing and deep learning, two of the most exciting and rapidly developing fields in physics and computer science.

This thesis is organized as follows: In Chapter 2, the principles of quantum error correction with stabilizer codes and the surface code are introduced. Furthermore, the basic concept of neural networks in general and graph neural networks, in particular, is described. Chapter 3 explains the architecture of the graph neural network explored in this thesis, defines the mapping from stabilizer measurements to a graph representation, and illudes the data generation and training setup. The decoding time and accuracy under the assumption of both perfect and imperfect stabilizer measurements are presented in Chapter 4. Chapter 5 concludes with a discussion and an outlook.

2

Theory

2.1 Quantum computation

The fundamental building block of each quantum computing device is the quantum bit, which is a two-level quantum system $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ defined on a two-dimensional Hilbert space with computational basis vectors:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad \text{and} \quad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}. \tag{2.1}$$

One then defines a set of unitary, hermitian and self-inverse qubit operators (or quantum *gates*) $X = \sigma_x$, $Y = \sigma_y$ and $Z = \sigma_z$ in terms of the Pauli operators σ_i acting on the two-level system:

$$X|\Psi\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}|\Psi\rangle = \alpha|1\rangle + \beta|0\rangle \tag{2.2}$$

$$Y|\Psi\rangle = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix}|\Psi\rangle = i\alpha|1\rangle - i\beta|0\rangle \tag{2.3}$$

$$Z|\Psi\rangle = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}|\Psi\rangle = \alpha|0\rangle - \beta|1\rangle \tag{2.4}$$

Including prefactors ± 1 and $\pm i$, the Pauli operators a group, usually referred to as the Pauli-group:

$$\mathcal{P} = \{\pm\mathbb{I}, \pm i\mathbb{I}, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}, \tag{2.5}$$

generated by X, Z and $i\mathbb{I}$. Furthermore, any two Pauli operators either commute or anti-commute:

$$[\sigma_i, \sigma_j] = 2i\epsilon_{ijk}\sigma_k \quad \text{and} \quad \{\sigma_i, \sigma_j\} = 2\delta_{ij}\mathbb{I}. \tag{2.6}$$

The actions of those operators are visualized in so-called *quantum circuits*, where time flows from left to right, horizontal wires stand for individual qubits and operators are indicated with boxes [25]. The following example applies an X operation to the ground state, thus exciting the state:

$$|0\rangle \text{ —}\boxed{X}\text{— } |1\rangle \,. \tag{2.7}$$

Two qubits interact with each other via the CNOT gate. Similar to the classical controlled-NOT, it flips the *target* qubit $|b\rangle$ if the *control* qubit $|a\rangle$ is in the excited state $|1\rangle$. This action is summarised by $|a\rangle\,|b\rangle \rightarrow |a\rangle\,|(a+b) \bmod 2\rangle$ and enables the entangling of two qubits. For instance, the first Bell state may be created by applying the CNOT gate to $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\,|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$. In the computational basis, the CNOT gate has the following matrix and circuit representation, with the top and bottom line representing the control and target qubit, respectively:

$$\mathrm{CNOT} = \frac{1}{\sqrt{2}}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix} \qquad \begin{array}{ll}|a\rangle & |a\rangle\\ |b\rangle & |(a+b) \bmod 2\rangle\end{array} \tag{2.8}$$

One further defines the Hadamard gate H and the $\pi/8$-gate T which are related to the Pauli operators by $X = HZH$ and $T^4 = Z$:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix}1&0\\0&\exp^{i\pi/4}\end{bmatrix}. \tag{2.9}$$

Together with the entangling CNOT gate, the Hadamard and the T-gate form a universal set of gates, meaning that any unitary operation on a quantum computer can be written as an infinite sequence of gates from this set. The Solovay-Kitaev theorem guarantees that for a finite number of qubits, the unitary operation can be efficiently approximated with arbitrary precision and only a finite number of gates [26].

In the last two decades, different physical implementations of qubits have been explored, such as superconducting circuits [27], trapped ions [28], and spins in semiconductors [29]. After many years of substantial efforts, all those technologies still have a fundamental shortcoming: The qubits are highly sensitive to their environment. Even preserving their state for several milliseconds is challenging, not to mention performing quantum gates or measuring them flawlessly. For this reason, any future quantum computation device will require quantum error correction.

2.2 Quantum error correction

Whenever processing information on a computer, may it be classical or quantum, one has to deal with noise. The aim is to protect the computational state such that it can be restored correctly, even after an error occurs. The basic idea is to encode the piece of information into a (larger) block of information by introducing additional bits [25]. The simplest example is the classical repetition code. Here, each bit is repeated n times, i.e. a *logical bit* 0 or 1 is encoded by a repetition of several *physical bits*: $0 \rightarrow 00...0$ and $1 \rightarrow 11...1$. After the encoded bit has been corrupted by noise, i.e. several of the physical bits have been flipped, one simply performs a majority vote to retrieve the correct logical bit. One can hereby correct up to $\frac{n-1}{2}$ errors on the physical bits. This technique is familiar to us from our daily experience: If your dialog partner doesn't get what you were just saying, you simply repeat the phrase.

In a quantum computer, there are, however, three major obstacles that prevent one from simply adopting classical error correction schemes like the repetition code [25]: Firstly, according to the *no-cloning theorem*, it is impossible to copy an unknown quantum state. Secondly, observation (measurement) of the quantum state collapses it to a basis state of the measured observable. Thirdly, a qubit is defined on a continuous Hilbert space over the complex numbers, implying that an error acting on it, in general, is continuous as well. Even if the error was known, it would take infinite precision to keep track of it. Luckily, quantum error correction can deal with all of those problems.

2.2.1 The three-qubit code

The smallest possible quantum error correction code that can detect and correct a bitflip error (i.e. an action of the Pauli operator X) is the three-qubit code [30]. Here, the information stored in a single-qubit state $|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ is distributed across an entangled three-qubit state, the *logical state* $|\Psi\rangle_L$:

$$|\Psi\rangle \longrightarrow |\Psi\rangle_L = \alpha\,|000\rangle + \beta\,|111\rangle\,. \tag{2.10}$$

It is important to note that this is not a threefold cloning of state $|\Psi\rangle$, instead, the information is stored in the fully entangled state $|\Psi\rangle_L$ with *code words* $|0\rangle_L = |000\rangle$ and $|1\rangle_L = |111\rangle$. After encoding, the two-dimensional Hilbert space of the encoded physical qubit is promoted to an eight-dimensional Hilbert space. It is instructive to divide this space into four mutually orthogonal, two-dimensional subspaces:

$$\begin{aligned} \mathcal{C} &= \mathrm{span}\{|000\rangle, |111\rangle\}, & \mathcal{F}_1 &= \mathrm{span}\{|100\rangle, |011\rangle\}, \\ \mathcal{F}_2 &= \mathrm{span}\{|010\rangle, |101\rangle\}, & \mathcal{F}_3 &= \mathrm{span}\{|001\rangle, |110\rangle\}. \end{aligned} \tag{2.11}$$

The logical state $|\Psi\rangle_L$ lies within the *codespace* $\mathcal{C}$ spanned by the two code words $|0\rangle_L$ and $|1\rangle_L$. Suppose one of the qubits is prone to a bitflip error X_i. This will rotate the state into one of the so-called *error spaces* $\mathcal{F}_i$: $X_i|\Psi\rangle_L \in \mathcal{F}_i$. Because the code and error subspaces are mutually orthogonal, one can distinguish between those by performing a projective measurement that does not destroy the quantum information stored in the logical state. The measurement outcome of such a projective measurement is called the *syndrome*; a string of classical bits from which one can deduce which error occurred. For the three-qubit repetition code, the two projective measurements are the operators Z_1Z_2 and Z_2Z_3. Both of them have eigenvalue $+1$ when acting on the logical state:

$$Z_1Z_2|\Psi\rangle_L = Z_1Z_2(\alpha|000\rangle + \beta|111\rangle) = (+1)|\Psi\rangle_L = Z_2Z_3|\Psi\rangle_L. \tag{2.12}$$

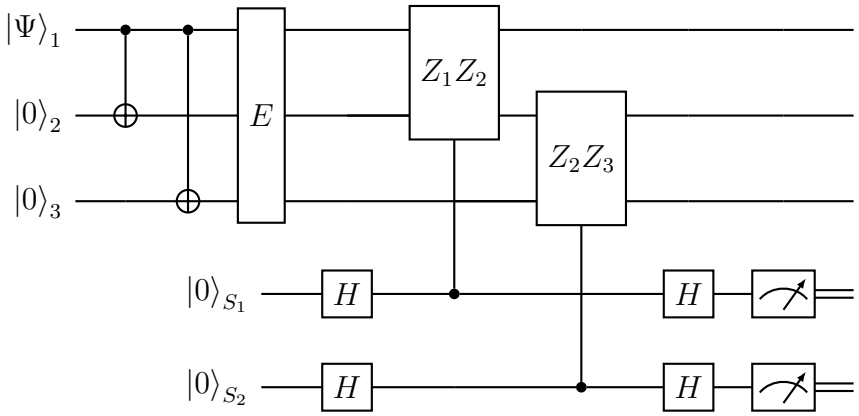
Because the logical state remains unchanged under the action of Z_1Z_2 and Z_2Z_3, the two operators are said to *stabilize* the logical state [31]. If, for instance, a bitflip error on the first qubit occurs, measurement of the first *stabilizer* Z_1Z_2 projects the state onto a different eigenspace ($\mathcal{F}_1$) with eigenvalue -1:

$$Z_1Z_2(X_1|\Psi\rangle_L) = Z_1Z_2(\alpha|100\rangle + \beta|011\rangle) = (-1)X_1|\Psi\rangle_L \tag{2.13}$$

The measurement outcome of the stabilizer Z_2Z_3 does not change, the corrupted state is projected onto the +1 eigenspace of this stabilizer:

$$Z_2Z_3(X_1|\Psi\rangle_L) = Z_2Z_3(\alpha|100\rangle + \beta|011\rangle) = (+1)X_1|\Psi\rangle_L \tag{2.14}$$

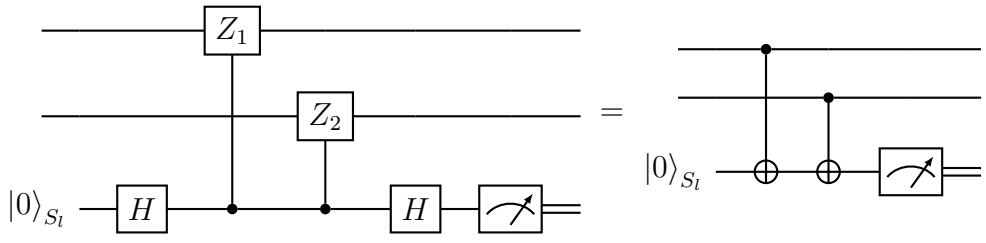
Each single-qubit bitflip error produces a unique syndrome (i.e. one classical bit from each of the stabilizers), from which one can deduce a suitable recovery operation. The three-qubit repetition code is displayed in the following circuit diagram:



In the encoding phase, the information contained in qubit $|\Psi\rangle_1$ is distributed into the logical three-qubit state $|\Psi\rangle_L$ by entangling with the two qubits $|0\rangle_2$ and $|0\rangle_3$. After an error from the set $E = \{X_1, X_2, X_3\}$, the syndrome is extracted by measuring the stabilizers Z_1Z_2 and Z_2Z_3 with the help of two ancilla qubits $|0\rangle_{S_1}$ and $|0\rangle_{S_2}$. The action of the stabilizer Z_1Z_2 controlled by the first ancilla qubit $|0\rangle_{S_1}$ transforms the corrupted state $E|\Psi\rangle_L$ to:

$$E|\Psi\rangle_L \longrightarrow \frac{1}{\sqrt{2}}\left(\mathbb{I}_1\mathbb{I}_2(E|\Psi\rangle_L)|0\rangle_{S_1} + Z_1Z_2(E|\Psi\rangle_L)|1\rangle_{S_1}\right) \tag{2.15}$$

If a bitflip error on one of the first two qubits occurs, the Hadamard gate after the controlled Z_1Z_2 projects the first ancilla qubit onto the excited state $|1\rangle_{S_1}$. The subsequent measurement of the eigenvalue -1 then indicates that the logical state was corrupted. Note that the action of the controlled Z_1Z_2 is in practice realized by two subsequent controlled Z gates with the ancilla qubit $|0\rangle_{S_l}$ as the control qubit. This is equivalent to applying a CNOT gate with the ancilla qubit as the target bit omitting the Hadamard gate:



The *logical* operators acting on the logical state have to capture the commutation relations and actions of the unencoded single-qubit operators they stand for. For the

three-qubit repetition code, the logical X_L is a tensor product of three single-qubit bitflips $X_L = X_1X_2X_3$, mapping the code words $|0\rangle_L$ and $|1\rangle_L$ to each other. The action of the Pauli operator Z is captured by its transformation of the basis vectors $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$: $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$. Following this definition, any single-qubit operator Z_i is sufficient to flip the encoded logical $|+\rangle_L$ and $|-\rangle_L$:

$$Z_i|+\rangle_L = Z_i\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = |-\rangle_L. \tag{2.16}$$

The *code distance* of a quantum code is the minimum weight of all logical operators, or in other words, the length of the smallest undetectable error. For the three-qubit repetition code, the weight of the logical X_L is three, but the weight of the logical Z_L is one, meaning that the three-qubit bitflip $X_1X_2X_3$ or any single-qubit phase flip Z_i remain undetected. This implies that the three-qubit repetition code has a code distance of one, and so it merely protects the information stored in the logical qubit from a bitflip on one of the physical qubits. The three-qubit repetition code can be extended to n qubits on a one-dimensional lattice, where n qubits together encode one logical qubit which is stabilized by $n-1$ operators Z_iZ_j acting on neighboring qubits:

$$|\Psi\rangle \longrightarrow |\Psi\rangle_L = \alpha\underbrace{|00...0\rangle}_{n} + \beta\underbrace{|11...1\rangle}_{n}. \tag{2.17}$$

This code can detect and correct up to $n-1$ and $\frac{n-1}{2}$ bitflip errors, respectively. However, it fails to detect a single-qubit phase flip Z_i. The following section extends the ideas gathered so far and explains how to protect a qubit from an arbitrary error.

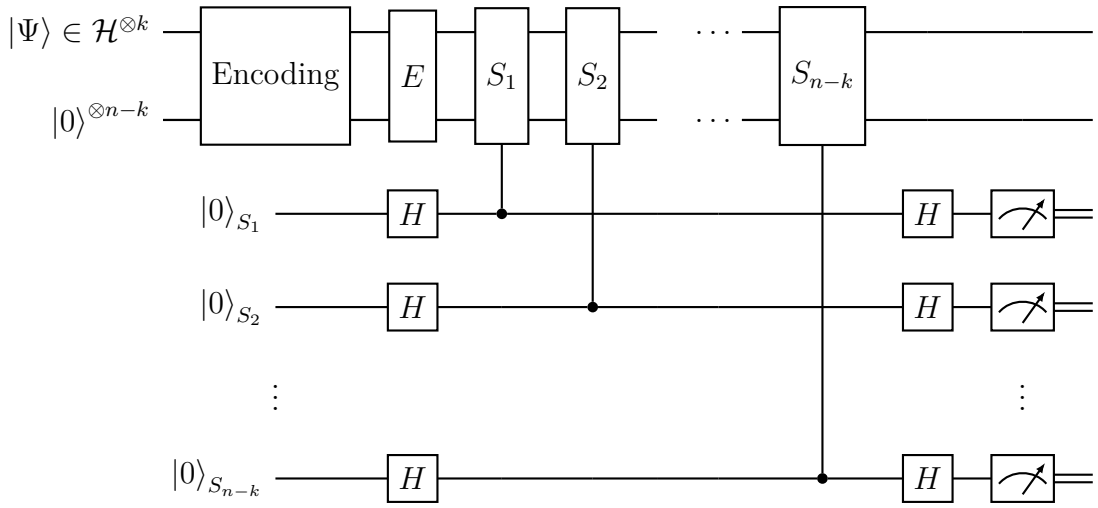
2.2.2 The stabilizer formalism

The stabilizer formalism is a technique that allows the compact description of quantum error correction codes in terms of operators from the n-qubit Pauli group $\mathcal{P}_n$ instead of the state vector of the ensemble of qubits [31]. The Pauli group on n qubits is hereby defined as the set of all n-fold tensor products of members of the single qubit Pauli group, i.e. it contains operators of type $X_1 \otimes \mathbb{I}_2 \otimes ... \otimes Z_j \otimes ... \otimes Y_n$. Just like the operators from the single-qubit Pauli group, members of the n-qubit Pauli group are unitary, hermitian, and self-inverse and have eigenvalues ± 1 or $\pm i$. Furthermore, any two operators either commute or anti-commute: two operators acting on different qubits naturally commute, whereas two operators acting on the same qubit obey the usual (anti-) commutation relations. One then studies a subset of operators with a special property: A stabilizer group $\mathcal{S}$ is a subgroup of $\mathcal{P}_n$, each element of which leaves any state in the vector space $\mathcal{C}$ (the codespace) invariant:

$$\mathcal{S} = \{S \in \mathcal{P}_n \quad | \quad S|\Psi\rangle_L = |\Psi\rangle_L, \quad \forall\, |\Psi\rangle_L \in \mathcal{C}\}. \tag{2.18}$$

From this definition follow immediately two important properties: Firstly, any two stabilizers P and P' commute (if $P|\Psi\rangle_L = |\Psi\rangle_L$ and $P'|\Psi\rangle_L = |\Psi\rangle_L$, then $PP'|\Psi\rangle_L = P|\Psi\rangle_L = |\Psi\rangle_L = P'P|\Psi\rangle_L$, thus $PP' = P'P$), implicating that the stabilizer group is Abelian. Secondly, $-\mathbb{I}$ is not part of the stabilizer group

($-\mathbb{I}\left|\Psi\right\rangle_L = \left|\Psi\right\rangle_L$ only holds for $\left|\Psi\right\rangle_L = 0$). One then constructs a stabilizer code as follows: By selecting $n-k$ commuting and independent generators generating the stabilizer group $\mathcal{S} \subset \mathcal{P}_n$, the codespace is defined as the space stabilized by this group. The following circuit displays the structure of an [[n, k, d]] stabilizer code, where n denotes the number of physical qubits that together encode k logical qubits with code distance d. A group of k qubits carrying the quantum information $\left|\Psi\right\rangle$ is encoded together with n-k redundancy qubits $\left|0\right\rangle^{\otimes n-k}$ to form the logical qubit $\left|\Psi\right\rangle_L$. The qubits encoding the logical state are usually referred to as *data qubits*. The logical state lies within the k-dimensional codespace (a subspace of the expanded n-dimensional Hilbert space $\mathcal{H}^{\otimes n}$) and contains the full information of the unencoded state $\left|\Psi\right\rangle$. After an error E occurs, the state is rotated onto one of the error spaces $\mathcal{F}_i \subset \mathcal{H}^{\otimes n}$ of dimension k. Measurement of n-k independent generators S_i of the stabilizer group gives a syndrome of n-k bits, from which a so-called *decoder* infers the best recovery operation R which rotates the state back onto the code space:



Note that it is possible to define the codespace as any of the ± 1 eigenspaces of the stabilizers by storing the measured eigenvalue of each stabilizer. For simplicity, the codespace is defined as the $+1$ eigenspace of all the stabilizers in the following discussion. The dimension of the codespace is determined by the number of stabilizer generators. One can show, that the vector space stabilized by the stabilizer group $\mathcal{S}$ generated by n-k commuting and independent operators from $\mathcal{P}_n$ is of dimension 2^k [25]. There is a total number of $2k$ logical operators acting on the codespace: one Pauli X_L^j operator and one Pauli Z_L^j operator for each logical qubit $j \in 1, ..., k$. Each logical operator L must commute with all stabilizers (if it were to anti-commute with one of the stabilizers S_i, then $L\left|\Psi\right\rangle$ would lie in the -1 eigenspace of this stabilizer: $L\left|\Psi\right\rangle = LS_i\left|\Psi\right\rangle = -S_iL\left|\Psi\right\rangle$). Furthermore, X_L^j and Z_L^j anti-commute just like their single-qubit analogs. The task of the decoder is to find the most likely error configuration giving a certain syndrome. From this information, a recovery operation R is applied to rotate the corrupted state back onto the codespace: $RE\left|\Psi\right\rangle_L = \left|\Psi\right\rangle_L$. If the recovery operation is such that a logical operator acts additionally, the resulting state is prone to a logical error: $RE\left|\Psi\right\rangle_L = L\left|\Psi\right\rangle_L$. The number of correctable errors c is related to the code distance as $c = \frac{d-1}{2}$.

A stabilizer code solves the three major problems stated in the introduction to this section: Without cloning the state one wishes to protect, the information is encoded in an entangled state of multiple physical qubits. Instead of measuring the state completely, incomplete observations of the system (the stabilizer measurements) reveal a suitable recovery operation. Lastly, any continuous error acting on each qubit can be corrected by discretizing it in terms of bitflip and phase flip errors, because the Pauli operators form a basis over all single-qubit unitaries. Note that the stabilizer formalism fails to protect from leakage, i.e. the transition into higher energy levels. Additionally, this quantum error correction scheme comes at the cost of a large overhead in the number of qubits needed to represent one logical qubit. In the next section, the surface code, one of the most promising stabilizer codes, is defined.

2.2.3 The surface code

The surface code is a *topological* stabilizer code which originally was defined on a torus [3]. It was realized that the periodic boundaries of the torus architecture can be realized by placing the qubits in a grid-like two-dimensional structure with two different types of boundaries [32]. Throughout this thesis, we work with the rotated surface code, which uses a minimal amount of physical qubits to encode and stabilize the logical qubit. We refer to this simply as the surface code. The surface code is a $[[d^2, 1, d]]$ code: It consists of a d times d grid of data qubits (white dots in fig. 2.1a) surrounded by $d^2 - 1$ ancilla qubits (black and red squares and half-circles in fig. 2.1a). The code space is stabilized by d^2-1 four-qubit (squares) and two-qubit (half-circles) X and Z stabilizers measured by the ancilla qubits. The weight-four stabilizers are products of single qubit X (Z) operators acting on four data qubits surrounding a black (red) square and are measured by the ancilla qubit in the center of the square. The weight-two stabilizers are products of X (Z) operators acting on pairs of data qubits on the boundaries of the surface code, measured by ancilla qubits in the neighboring black (red) half-circles. It is easy to check that any two stabilizers commute: If they act on different qubits, they commute trivially. Any two X or Z stabilizers also commute trivially. And finally, adjacent stabilizers Z_{ijkl} and X_{klmn} commute because they act on two joint qubits:

$$\begin{aligned} Z_{ijkl}X_{klmn} &= Z_iZ_jZ_kZ_lX_kX_lX_mX_n = -Z_iZ_jX_kZ_kZ_lX_lX_mX_n \\ &= Z_iZ_jX_kX_lZ_kZ_lX_mX_n = X_kX_lX_mX_nZ_iZ_jZ_kZ_l = X_{klmn}Z_{ijkl}. \end{aligned}$$

Figure 2.1b shows the implementation of the stabilizer operator Z_{abcd} acting on the data qubits a, b, c and d. Similarly, the stabilizer X_{efgh} acting on qubits e, f, g and h is displayed in fig. 2.1c. By measuring all $d^2 - 1$ stabilizers (a so-called surface code *cycle*), the d^2 data qubits are projected onto the simultaneous eigenstate of all stabilizers, implying that they together encode one logical qubit.

Recall that the logical operators acting on the logical qubit have to commute with all the stabilizers as well as resembling the commutation relations of their single-qubit analogs. One constructs the logical X_L by a chain of single-qubit X operators running along the western edge of the grid. The logical Z_L is defined as the product of single-qubit Z operators on the northern edge. The logical Y_L is then the product

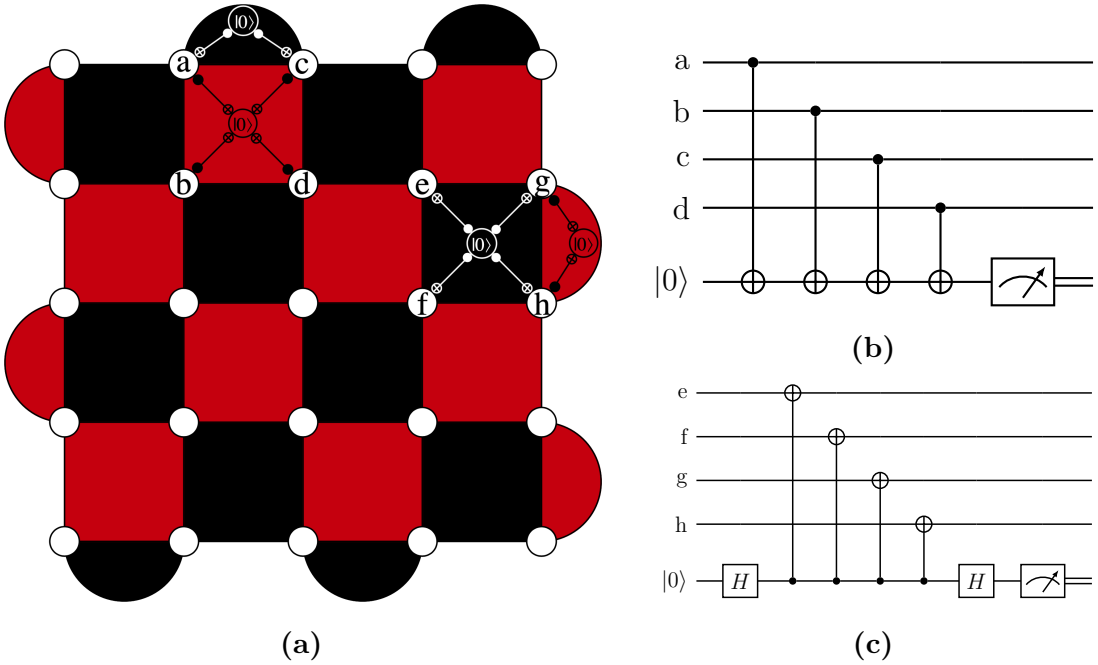
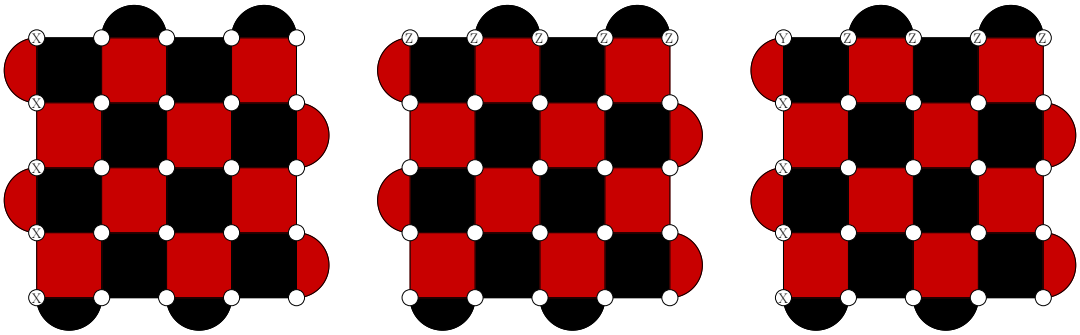


Figure 2.1: Distance 5 circuit code. **(a)** The white dots, black (red) squares and black (red) half-circles represent the data qubits, weight-four X (Z) stabilizers and weight-two X (Z) stabilizers, respectively. Exemplary quantum circuits to measure the stabilizers X_{ac}, Z_{abcd}, X_{efgh} and Z_{gh} are shown schemetically. **(b)**, **(c)** Quantum circuit for measuring the stabilizers Z_{abcd} and X_{efgh}.

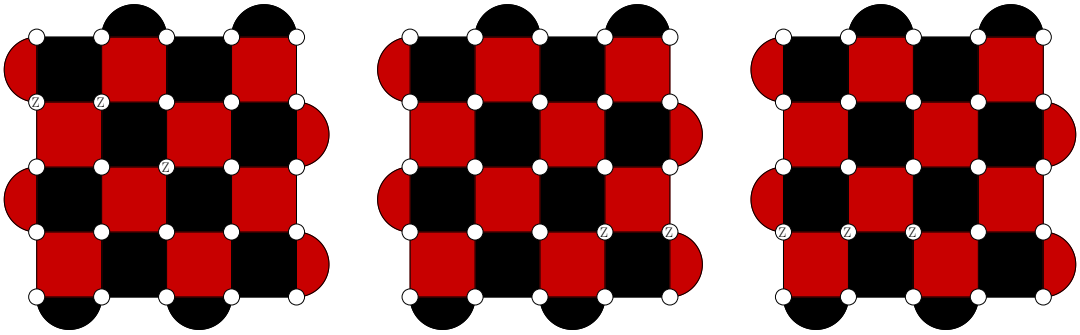
of X_L and Z_L. The logical operators of the surface code are displayed in fig. 2.2. Because the logical operators commute with the stabilizers by definition, they can be distorted and shifted around the code by multiplication with stabilizer operators. As long as there is an uneven number of chains of X (Z) operators running from north to south (west to east), the logical qubit is rotated by X_L (Z_L).

Any configuration of errors that does not commute with the stabilizers, changes the outcome of the stabilizer measurements (syndrome). The purpose of a decoder is to find the most likely chain of errors that caused the syndrome. A suitable correction operation is then applied to bring the state back into the codespace without introducing a logical error. This is a computationally complex task because each syndrome can be caused by many different error configurations. Furthermore, the number of possible syndromes grows exponentially with the number of stabilizers, implying that a look-up table with the most likely error chain for each syndrome is practically impossible. Note that the errors are corrected in the classical control unit because applying the corresponding gates on the data qubits would be an additional source of errors.



(a) **(b)** **(c)**

Figure 2.2: Logical operators X_L, Z_L and Y_L in **(a)**, **(b)** and **(c)**, respectively.



(a) **(b)** **(c)**

Figure 2.3: **(a)** Error chain from the logical coset $\mathcal{Z}$. Correction comprised of different single-qubit operators producing the same syndrome as the error chain, but from the logical coset $\mathcal{I}$ **(b)** and $\mathcal{Z}$ **(c)**.

The correction operation is split into two parts. Firstly, one needs to find a chain of Pauli operators that rotates the state back into the codespace, i.e. reproduces the anticipated stabilizer measurement outcomes of an error-free cycle. Secondly, an algorithm computes the most likely logical operator of the product of the correction and the underlying error chain. For the latter task, one defines four logical cosets capturing the commutation relations of the error chain and the logical operators: errors with an uneven parity of X (Z) operators on the northern (western) edge fall into coset $\mathcal{X}$ ($\mathcal{Z}$), errors with both uneven parities of X on the northern and Z on the western edge belong to coset $\mathcal{Y}$ and errors with an even parity of X (Z) operators on the northern (western) edge fall into coset $\mathcal{I}$. If the correction belongs to a different logical coset than the error, one needs to apply the corresponding logical operator mapping between the two cosets. Consider the following example: The chain of errors in fig. 2.3a falls into the logical coset $\mathcal{Z}$. If one corrects for the error with the correction displayed in fig. 2.3b from coset $\mathcal{I}$, the state is brought back into the codespace, but a logical Z_L acts on the logical qubit. In contrast, the product of the underlying error and the correction displayed in fig. 2.3c leaves the logical state unchanged.

So far, we only considered a simplified model by assuming that the stabilizers can be measured perfectly. In reality, however, the ancilla qubits are prone to measurement errors, yielding a wrong measurement outcome with a certain probability. Furthermore, errors can act before and after any gate of the stabilizer circuits fig. 2.1c and fig. 2.1b, usually referred to as *circuit-level noise*. The solution to this problem is to repeat the stabilizer cycle d_t times. The syndrome is then constructed from each measurement cycle and the decoder infers a suitable recovery operation by taking the information from all time steps into account.

2.3 Graph neural networks

Graphs are widely used to model and represent complex relationships and interactions among entities. However, traditional neural networks struggle to handle graph-structured data directly. Graph neural networks, a class of neural networks specifically designed for graph data, offer a promising solution to this problem, enabling effective analysis, prediction, and understanding of graph-structured data. GNNs find applications in diverse domains: In social network analysis, they can identify community structures, detect influential nodes, and predict missing links. In chemistry and bioinformatics, GNNs can predict molecular properties, drug-target interactions, and protein functions by leveraging the graph representation of molecules or biological networks [33]. In physics, they find application in nearly all disciplines, from the efficient representation of quantum many-body states [34], over materials science [35], to tasks at the Large Hadron Collider [36]. After introducing the basic principles of neural networks, the function and structure of graph convolutional neural networks are explained.

2.4 Artificial neurons and neural networks

The basic building block of neural networks is the artificial neuron. It aims at mimicking the function of biological neurons in brains: Incoming signals are accumulated and weighted according to their importance. If a certain *activation threshold* is reached, the neuron gives a non-zero output [37]. One can describe this relationship with a simple mathematical model:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{2.19}$$

The output y of the neuron is determined by its *activation function* f of the weighted sum of incoming signals x_i from neighboring neurons plus a *bias* b. The activation function typically is continuous and monotonically increasing. In an *artificial neural network* (ANN) multiple neurons are stacked in subsequent layers. Signals can be sent between the neurons in the same or different layers. Most ANNs consist of a so-called input layer, the incoming signal of which is a numerical representation of the data one wants to analyze, an output layer giving the result of the analysis, and hidden layers between those. The *feed-forward* neural network is the simplest type of ANNs. Here, information moves only forward through the layers. If each neuron in one layer $k-1$ is connected to every neuron in the subsequent layer k, the network is called *dense* or *fully connected*. By concatenating the inputs and outputs of each node in layer k to vectors $y^{(k-1)}$ and $y^{(k)}$, one arrives at the following expression:

$$y^{(k)} = f\left(W^{(k)} y^{(k-1)} + b^{(k)}\right). \tag{2.20}$$

Because the input to each neuron only depends on the output from each neuron in the preceding layer, the output vector $y^{(l)}$ of the neural network can be computed from the input vector $y^{(0)}$ and the weights $W^{(k)}$ and biases $b^{(k)}$ from each layer k:

$$y^{(l)} = f\left[W^{(l)} f\left(W^{(l-1)}\left(\dots f\left(W^{(1)} y^{(0)} + b^{(1)}\right)\dots\right) + b^{(l-1)}\right) + b^{(l)}\right]. \tag{2.21}$$

This means that a fully connected feed-forward neural network is just a map between real-valued vectors. This map is nothing more than a nested sum of activation functions with weights and biases as parameters. The flexibility of the network comes from adjusting these parameters [38]. NNs are trained using gradient descent optimization algorithms, combined with backpropagation based on the chain rule. Backpropagation computes the gradients of the *loss function* (a measure for the distance between the true value and output of the network) with respect to the network parameters. These gradients are then used to update the parameters in the opposite direction of the gradient, iteratively optimizing the model's performance (gradient descent). This procedure is repeated to get through multiple *epochs* of training.

2.4.1 Graph convolutional layers

A graph convolutional layer is a key component in the graph neural networks explored in this thesis. It aims to extract meaningful features from nodes in a graph by leveraging the underlying graph structure. In the graph convolutional layer, each node in the graph is represented by a *node feature vector*. The layer performs convolutional operations on these node features by considering the local neighborhood around each node. This is in contrast to traditional convolutional layers, which operate on grid-like structures. To achieve this, a weight matrix learns the relationships and importance between nodes. The aggregation process combines the feature vectors of neighboring nodes with their corresponding weights. Different aggregation methods can be employed, such as summing, averaging, or concatenating the neighbor features. The resulting aggregated vector captures information from the local neighborhood of each node.

After aggregation, the feature vector is transformed by applying an activation function. By applying the aggregation and transformation steps to all nodes in the graph, the graph convolutional layer produces a set of new feature vectors that encode local structural information. This enables the utilization of the learned representations for tasks such as node classification, link prediction, or graph classification.

At the heart of the networks used throughout this work lies the following graph convolutional layer. The purpose of this layer is to map the node features x_i of all nodes of the graph to a representation with different dimension x_i' by collecting information from neighboring nodes [39]:

$$x_i' = f\left(W_1 \cdot x_i + W_2 \cdot \sum_{j \in \mathcal{N}(i)} e_{ji} \cdot x_j\right). \tag{2.22}$$

After that, an activation function f is applied. Note that this layer doesn't change the structure of the graph (i.e. number of nodes and connections between them), it merely evolves the node features to a new representation, possibly of different dimensions. Because the number of nodes varies from graph to graph, the high-dimensional graph embedding obtained after multiple graph convolutional layers is pooled to get a single vector of known dimension (section 3.2).

3
Methods

3.1 From syndrome to graph

3.1.1 Perfect stabilizer measurements

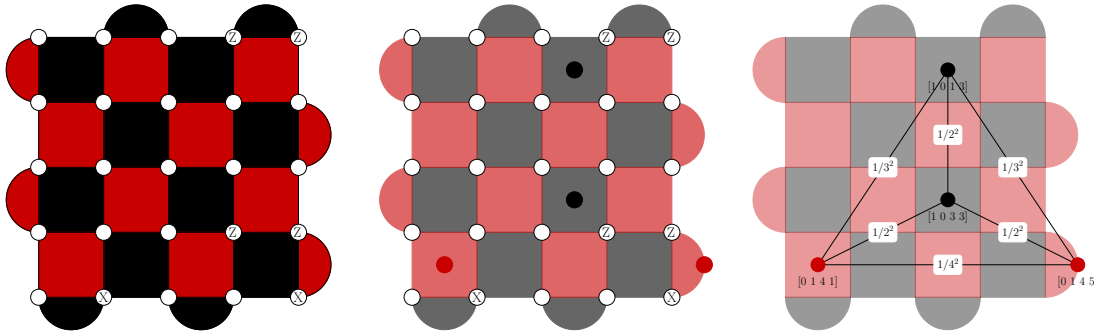
To determine the most likely logical coset of the underlying error chain, we map the syndrome measurement outcomes to an undirected graph. A graph is defined as a pair of a set of nodes N and a set of edges E, which connect between pairs of nodes. Each stabilizer measurement that differs from the error-free case sets off a so-called *detection event*. A set of detection events produces the syndrome displayed in fig. 3.1b. Each detection event is mapped to a node in a graph (fig. 3.1c). All nodes are connected with their six nearest neighbors by starting from a fully connected graph and removing the most distant neighbors of each node. Consequently, the number of edges connecting to (and from) each node is constant and so the number of operations in each network layer scales linearly with the number of detection events (eq. (2.22)). Furthermore, the key feature of the graph neural network architecture is that the total number of trainable weights is independent of the size of the graph (i.e. the number of nodes). That implies that the GNN is in principle capable of decoding syndromes from surface codes of arbitrary size. The nodes are labeled with node features indicating the stabilizer type and the distance from the northern and western boundary: $[X?, Z?, d_{\text{North}}, d_{\text{West}}]$. The edges connecting two nodes i and j with coordinates $(d^{i(j)}_{\text{North}}, d^{i(j)}_{\text{West}})$ are labeled with the square of the inverse supremum norm of the two nodes, as pictured in fig. 3.1c:

$$e_{ij} = \left(\max\{|d^i_{\text{North}} - d^j_{\text{North}}|, |d^i_{\text{West}} - d^j_{\text{West}}|\}\right)^{-2}. \tag{3.1}$$

3.1.2 Surface code under circuit-level noise

If circuit-level noise acts on all qubits in the surface code, the stabilizer measurement cycles are repeated multiple times. Here, each change in detection events is added as a new node to the graph (fig. 3.2). The temporal distance from the first measurement round, i.e. the time-wise boundary is added as a fifth node feature: $[X?, Z?, d_{\text{North}}, d_{\text{West}}, d_{\text{time}}]$. The edges connecting two nodes i and j with coordinates $(d^{i(j)}_{\text{North}}, d^{i(j)}_{\text{West}}, d^{i(j)}_{\text{time}})$ are now labeled with the square of the inverse supremum norm of the two nodes with respect to both time and space:

$$e_{ij} = \left(\max\{|d^i_{\text{North}} - d^j_{\text{North}}|, |d^i_{\text{West}} - d^j_{\text{West}}|, |d^i_{\text{time}} - d^j_{\text{time}}|\}\right)^{-2}. \tag{3.2}$$



(a) **(b)** **(c)**

Figure 3.1: Distance 5 surface code: Mapping from syndrome to graph. **(a)** Underlying error chain that causes the syndrome in **(b)**. **(c)** Graph representation of the syndrome, annotated with node features and edge weights.

3.1.3 Experimental repetition code data

To test if the GNN decoder is capable of learning the underlying error distribution of real and imperfect physical qubits of varying quality, networks were trained on real experimental data [4]. Because data for the surface code was limited (10^5 data points), this test was conducted only for the repetition code. The Google Quantum AI team ran a distance-25 repetition code experiment with 50 stabilizer cycles on their `sycamore` processor, publishing data from $5 \cdot 10^5$ shots. By subsampling smaller code sizes from the full chain of qubits, $1.15 \cdot 10^7$, $1.05 \cdot 10^7$ and $9.5 \cdot 10^6$ data points for code size 3, 5 and 7, respectively, were distilled. Changes in detection events are mapped to a node in a graph, yielding an effective two-dimensional structure similar to the surface code with perfect stabilizer measurements. Figure 3.3 displays the circuit diagram of a repetition code of distance 3 with 10 measurement cycles, including circuit-level noise acting on the qubits.

In contrast to the surface code, a single measurement of (any) single data qubit is sufficient to define the logical coset. The logical coset is just a binary label with respect to the logical X (Z) operator because the repetition code only can detect either bitflip (phase flip) errors, depending on which basis the stabilizers are measured in. Now, the nodes in the graph are annotated with three node features: the commutation relation of the final state of the first qubit with the logical operator, the temporal distance from the first measurement round, and the spatial distance from the first qubit: $[X?, d_{\text{time}}, d_{\text{space}}]$. The square of the inverse supremum norm between two nodes i and j labeling the edge between those nodes is now computed with respect to the time and space coordinates:

$$e_{ij} = \left(\max\{|d^i_{\text{time}} - d^j_{\text{time}}|, |d^i_{\text{space}} - d^j_{\text{space}}|\}\right)^{-2}. \tag{3.3}$$

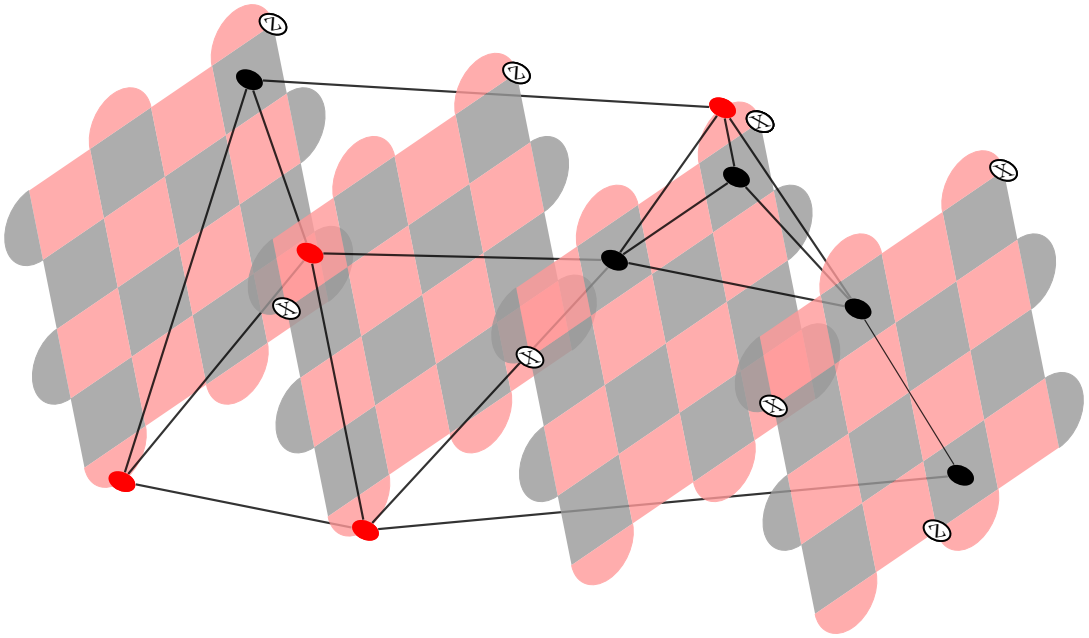


Figure 3.2: Distance 5 surface code under circuit-level noise, 4 stabilizer cycles with time progressing from left to right. Errors acting on the data qubits (indicated with the corresponding Pauli operator) cause a change in the measurement outcome of the neighboring stabilizer (black: X-type, red: Z-type). Measurement errors on the ancilla qubits also cause a change in the stabilizer measurement. Each change in stabilizer measurements is mapped to a node in a graph, which is displayed with only a few edges for clarity.

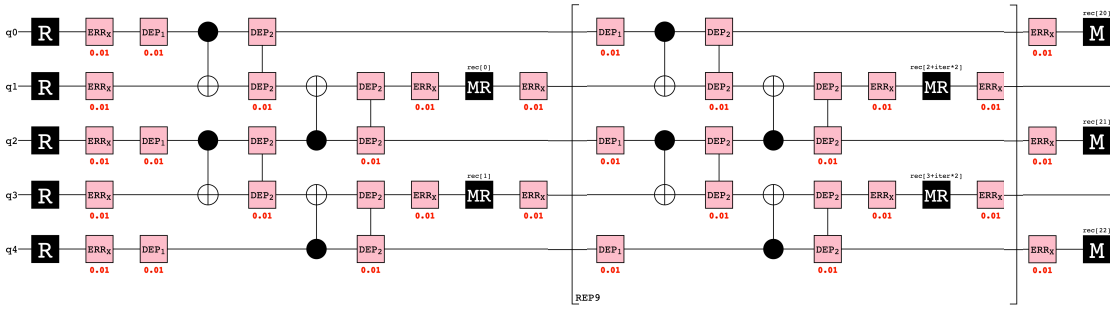


Figure 3.3: Distance 3 repetition code with 10 stabilizer cycles: Circuit diagram including circuit-level noise. The gate block in brackets is repeated 9 times. Figure generated with `stim` [40].

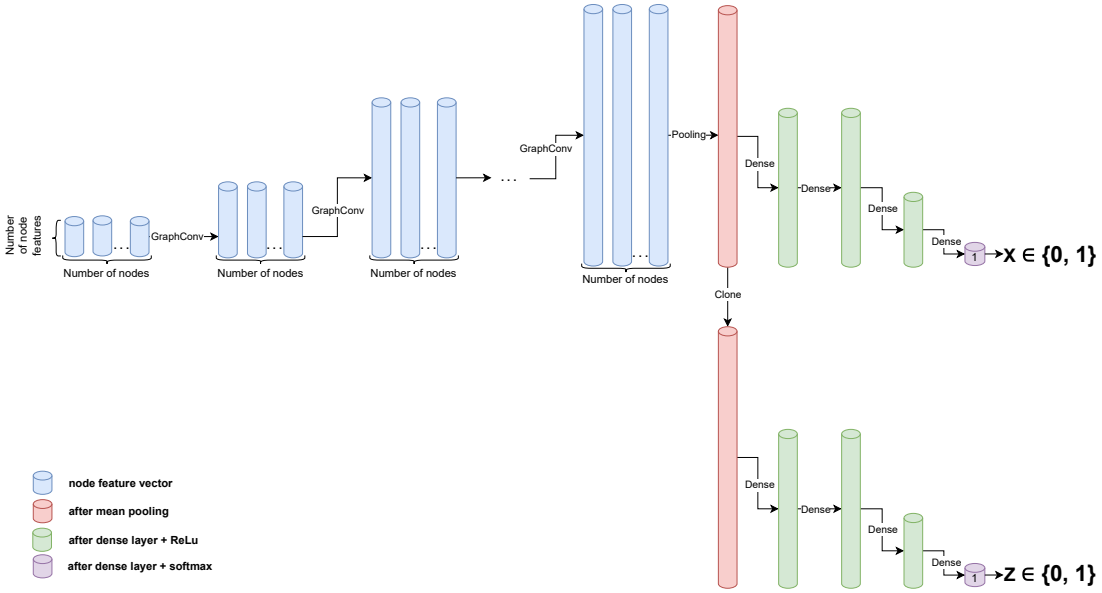


Figure 3.4: Architecture of the GNN: The node features of a graph with a given number of nodes are sent through multiple graph convolutional layers (blue). Then, they are pooled into one high-dimensional vector (red). This vector is cloned and propagated through two different sets of dense layers (green), each of which gives a binary output (purple) for logical coset $\mathcal{X}$ and $\mathcal{Z}$.

3.2 Graph neural network architecture

The graphs are fed through a graph neural network which classifies them according to the most likely logical coset of their underlying error configuration. The architecture of the GNN is displayed in fig. 3.4. The node features are mapped to a representation of higher dimension by several subsequent graph convolutional layers according to eq. (2.22). After each layer, the node features are processed by the rectified linear unit activation function which has the following form:

$$f(z) = \max(0, z). \tag{3.4}$$

After the transformation through the graph convolutional layers, the node features x_i from all nodes are pooled into one high-dimensional vector X by computing the mean across all nodes:

$$X = \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{3.5}$$

This vector is then cloned and sent to two architecturally identical dense neural networks (eq. (2.21)). Both networks (*heads*) map the pooled node feature vector down to one real-valued number which is normalized to a probability P for logical coset $\mathcal{X}$ and $\mathcal{Z}$ via a sigmoid function: $[P_{\mathcal{X}}, P_{\mathcal{Z}}]$. Rounding those probabilities to one or zero finally gives a prediction for the logical coset of the underlying error chain encoded in the following way: $\mathcal{I} \leftrightarrow [0,0], \mathcal{X} \leftrightarrow [1,0], \mathcal{Z} \leftrightarrow [0,1]$ and $\mathcal{Y} \leftrightarrow [1,1]$.

3.3 Training setup

For simulating perfect stabilizer measurements surface code experiments, samples are generated by randomly drawing a tensor product of Pauli operators acting on the data qubits. Next, the stabilizer measurement outcomes are computed and compared to the error-free case, which gives the syndrome. The syndrome is then mapped to a graph as described in section 3.1.1. Circuit-level noise was simulated using the `stim` `Python` library, a fast stabilizer circuit simulator [40]. Because `stim` simulates a real experiment, the data qubits can only be measured in the Z or X basis, implying that either the logical coset $\mathcal{Z}$ or $\mathcal{X}$ can be determined. One of the heads is then shut down and the cost function defined below is computed with respect to just one of the two binary labels Z or X. Furthermore, with the final measurement of all data qubits, one not only infers the logical coset of the chain of errors but also constructs a last round of "perfect" (up to measurement errors on the data qubits, that is) stabilizer measurements by computing their parities. This additional set of detection events at time step $d_t + 1$ is also included in the graph.

Error chains are sampled according to the depolarizing noise model, where the probabilities of X, Y and Z errors are equal and a third part of the total error probability p of a single qubit:

$$p_x = p_y = p_z = \frac{p}{3} \quad \text{and} \quad p_x + p_y + p_z = p. \tag{3.6}$$

In the case of circuit-level noise, errors act with error rate p after all gates, after resetting and before each measurement on all qubits and before each new surface code cycle on the data qubits (fig. 3.3).

For each training sample, the prediction of the network $[P_{\mathcal{X}}, P_{\mathcal{Z}}]$ is compared to the true label $[y_x, y_z]$ via the binary cross entropy cost function:

$$C_x(P_{\mathcal{X}}, y_x) = -\left[y_x \cdot \log P_{\mathcal{X}} + (1 - y_x) \cdot \log\left(1 - P_{\mathcal{X}}\right)\right] \tag{3.7}$$

$$C_z(P_{\mathcal{Z}}, y_z) = -\left[y_z \cdot \log P_{\mathcal{Z}} + (1 - y_z) \cdot \log\left(1 - P_{\mathcal{Z}}\right)\right] \tag{3.8}$$

Note that a fraction of the data is labeled wrongly: given a chain of errors (which is drawn at random according to a certain error rate), the syndrome is computed and labeled with the logical coset of this error chain. Because the mapping from error chains to syndromes is a mapping of many to one, the syndrome may also be caused by a different error chain with higher probability and possibly a different logical coset. However, we argue that in the limit of small error rates and large numbers of samples, this effect is neglectable.

Up next, the values of the two cost functions C_x and C_z are added and the gradients with respect to the parameters of the graph convolutional and dense layers are evaluated by using back-propagation. Those steps are repeated for all samples in a batch of size 1,000 thus yielding gradients for 1,000 samples. The parameters of the network are finally updated considering all samples from one batch with the Adam (short for Adaptive Moment Estimation) optimizer, a computationally efficient implementation of stochastic gradient descent [41]. The amplitude with which the parameters are updated is controlled by the *learning rate*, which was set to 10^{-4} in this thesis. This procedure is repeated for all batches of the dataset to form

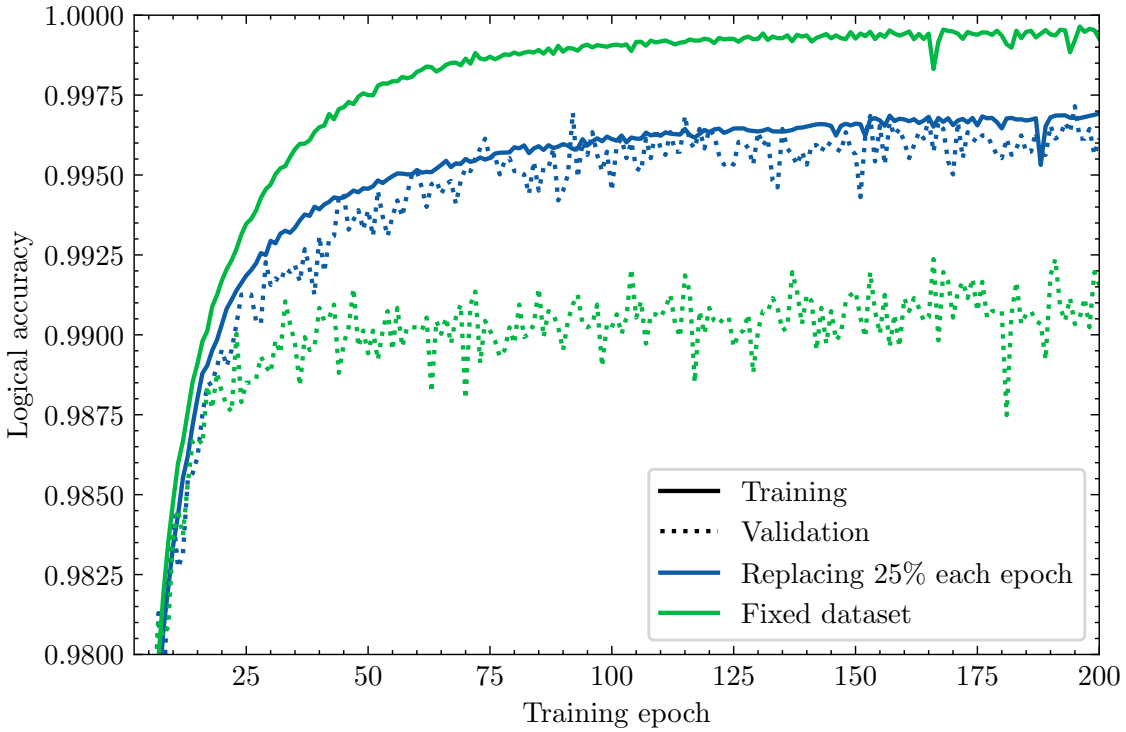


Figure 3.5: Comparison of the influence of replacing data during training. Perfect syndrome measurements, code size $d = 9$, error rate $p = 0.05$. Logical accuracy for training and validation data as a function of the training epochs. Green: fixed dataset. Blue: replacing 25% of the dataset after each epoch.

one *training epoch*. Training the networks involves many training epochs, such that the network processes each sample multiple times. The proportion of the number of correctly classified samples to the total number of samples defines the *logical accuracy*, a quantity that is evaluated on a training, validation and test dataset. The graph neural network and the training loop were implemented in the `Python` programming language using the `PyTorch` and `PyTorch Geometric` libraries [42, 43]. The RAM capacity limits the number of samples in the dataset to several million. However, this is not sufficient to train networks to high accuracy without overfitting, i.e. capable of generalizing on unseen data. To circumvent this bottleneck, a part of the dataset is replaced after each epoch with new samples generated on the fly. An optimal rate of replacement was found to be 25%, where the networks reach high accuracies without overfitting (see fig. 3.5).

For each code size, one network is trained on samples generated from different error rates. This is to ensure that the networks reach high logical accuracies when tested across a range of different error rates. The logical accuracy of the graph neural network decoder is benchmarked against the minimum weight perfect matching (MWPM) algorithm. MWPM is based on the Blossom algorithm [6], which matches pairs of nodes (detection events) by minimizing the total weight of the edges (weighted with the inverse error rates) between those nodes. The algorithm was implemented with the `PyMatching` library [44].

4
Results

4.1 Time scaling of the GNN decoder

The motivation for this work is not only to improve upon the decoding accuracy of existing decoders but also the inference time. Because the logical state must be protected in real-time during a quantum computation, the decoder must be as fast as possible. Here, we give an estimate of the scaling of the inference time of our GNN decoder with the code size in comparison to minimum weight perfect matching. Figure 4.1 displays the average decoding time T of a syndrome from perfect stabilizer measurements sampled at $p = 0.05$ as a function of the code size for MWPM and the GNN. The dotted lines show a linear regression of the data points over the log-log scale, assuming a polynomial scaling of the inference time with the code size: $T = C \cdot d^{\alpha}$. The scaling exponents α were found at 1.89 ± 0.04 for the GNN and 2.16 ± 0.05 for MWPM. This is expected as the number of operations in the networks scales linearly with the number of nodes, i.e. quadratically in the code size at a fixed error rate, given that the number of edges to each node is kept constant. Note that to correct for the temporal overhead when distributing the samples to the GPU, multiple samples were collected in one batch in this analysis. In a real application, however, one wishes to decode one sample, i.e. one logical qubit at a time.

In addition to the fact that the GNN yields no significant improvement in terms of decoding time compared to MWPM, the time it takes to construct the graphs from a syndrome measurement scales with $\alpha = 3.81 \pm 0.11$ with the code size. However, this time could be substantially improved by implementing this part of the computation in `C` and by choosing a more sophisticated algorithm to remove edges between distant nodes. Furthermore, we argue that the scaling of the decoding time does not pose a constraint on the practicality of our decoder. In a real quantum computing device, the network could be implemented in hardware once trained, allowing for fast inference times [45].

4.2 Perfect stabilizer measurements

First, we test the capabilities of the graph neural network decoder to decode syndromes under the assumption of perfect stabilizer measurements. For each code size, networks with the same architecture are trained on a dataset containing 4 million samples from error rates $p = 0.01, 0.05, 0.01, 0.15$ (1 million samples per error rate). After each training epoch, 25% of the dataset is replaced with new samples gener-

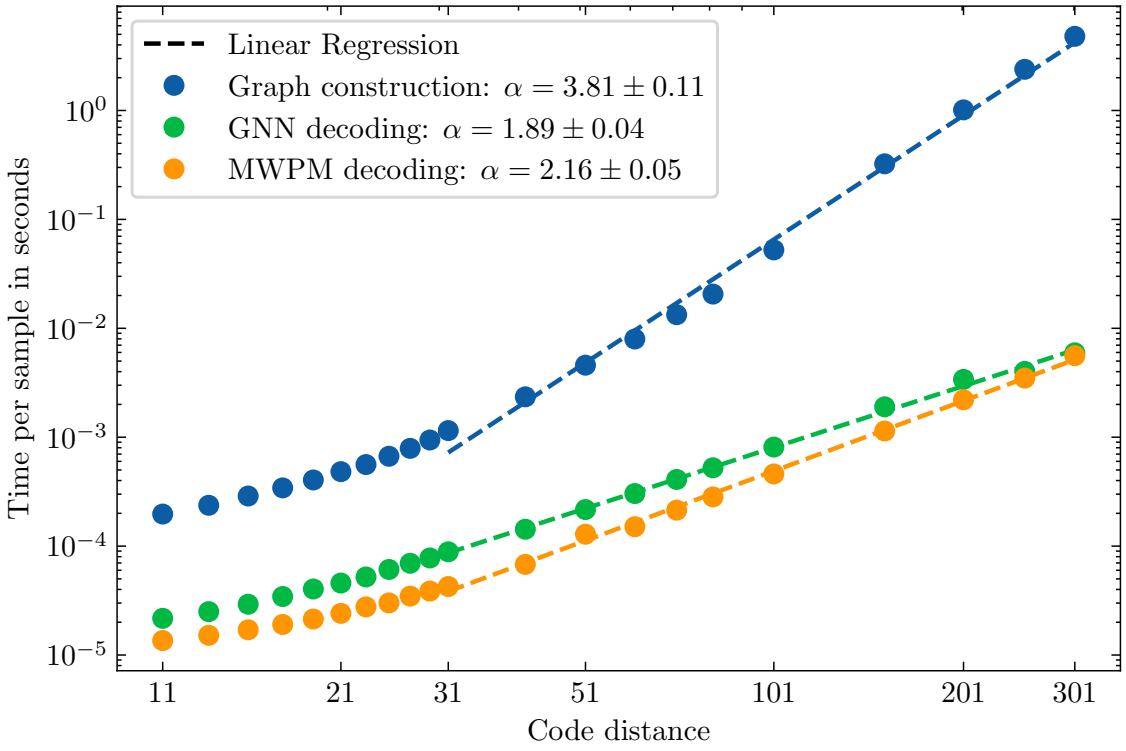


Figure 4.1: Average decoding time per syndrome from perfect stabilizer measurements sampled at $p = 0.05$ vs code size. Dotted lines show a linear regression according to the ansatz: $T = C \cdot d^{\alpha}$. Blue: graph construction time.

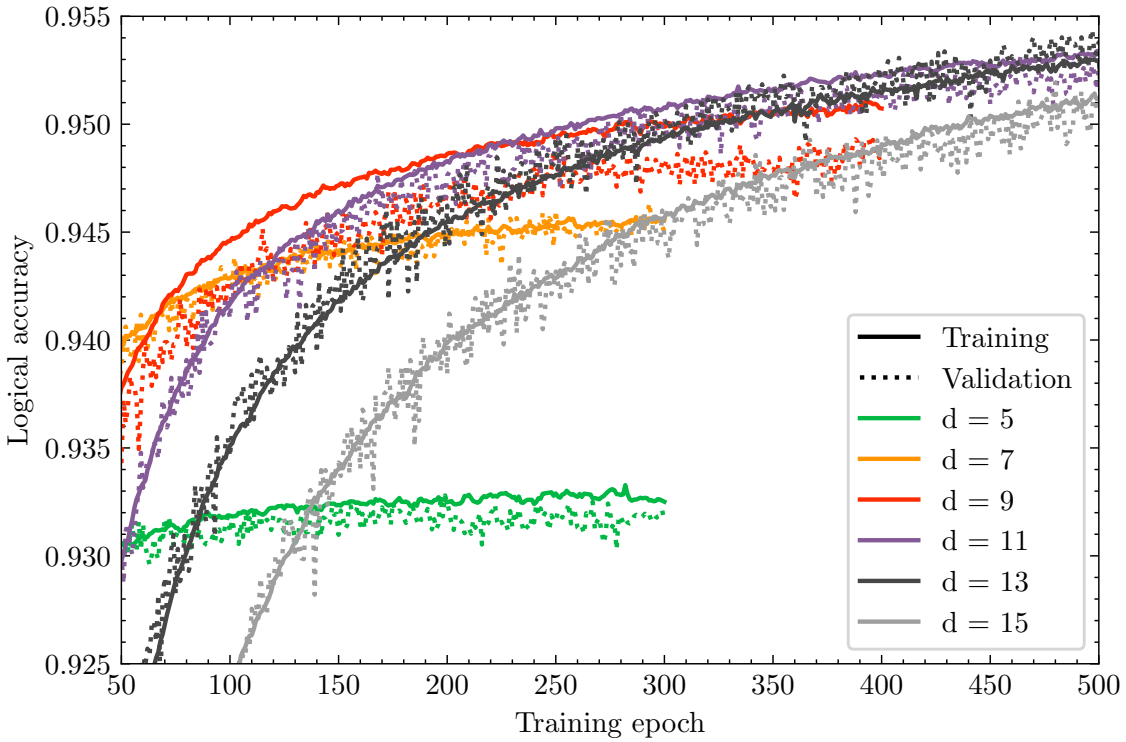


Figure 4.2: Training history of networks trained on different code sizes.

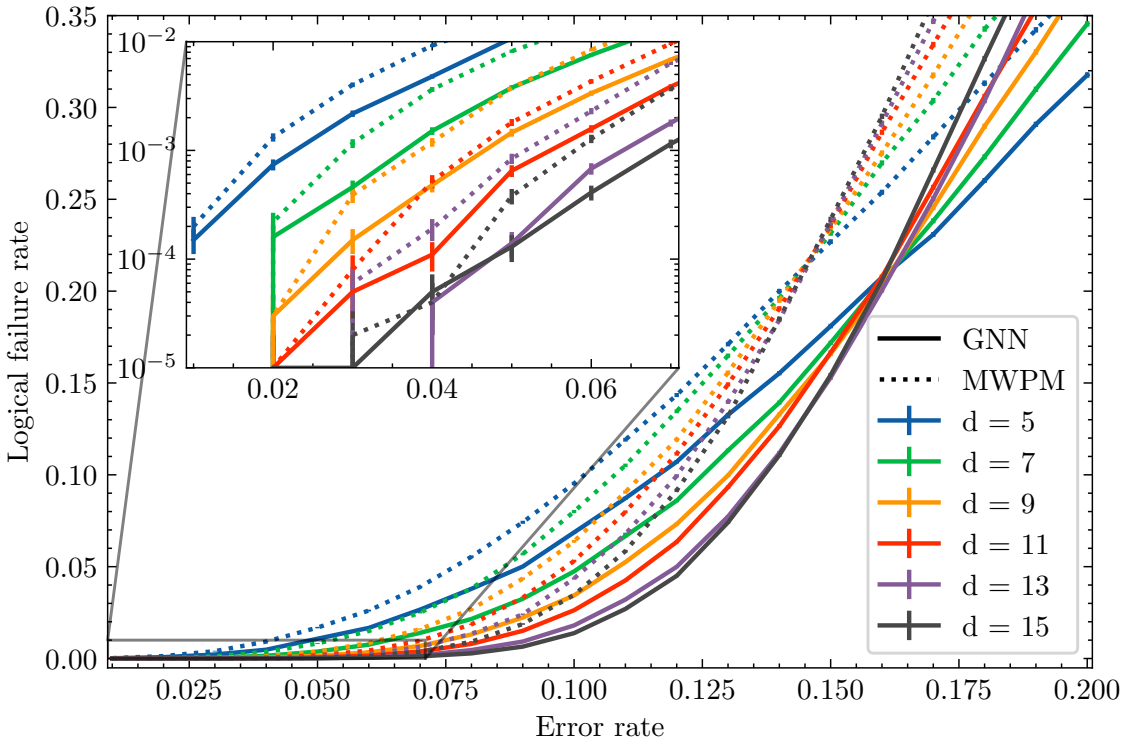


Figure 4.3: Comparison of decoding accuracy of the GNN decoder (solid lines) and MWPM (dotted lines). Logical failure rate as a function of the error rates of the data qubits. The inset shows a magnification of the range of low error rates on a logarithmic y-scale. Error bars indicate confidence intervals of one standard deviation.

ated from the same mix of error rates. Training is terminated whenever the network approaches convergence, i.e. when the validation accuracy does not improve significantly throughout several epochs. Given the fact that the number of nodes in the graphs grows with the code size given a certain error rate and because the networks have the same number of trainable parameters across all code sizes, the number of epochs needed for convergence grows with the code size. This behavior is evident from fig. 4.2, where the networks trained at code sizes five and seven seem to have reached convergence at around 300 epochs, whereas larger code sizes still seem to improve, i.e. need more training. The network used to decode syndromes from a surface code of size 15 for instance was trained for 10^3 epochs, meaning that the total number of unique samples was at the order of billions.

After training, the logical accuracy of the networks is tested as a function of the error rate (fig. 4.3). The performance of the GNNs is benchmarked against the MWPM algorithm. Networks were trained up to code size 21. However, for code sizes larger than 15, the logical accuracies of the GNNs do not improve with the code size and are lower than MWPM. This is because convergence gets increasingly difficult and with the size of the networks kept constant. For this reason, only networks up to code size 15 are displayed in fig. 4.3. It is difficult to define a clear threshold (the error rate at which the logical failure rates intersect, i.e. the error rate where it pays off to increase the code size). This is because the level of convergence

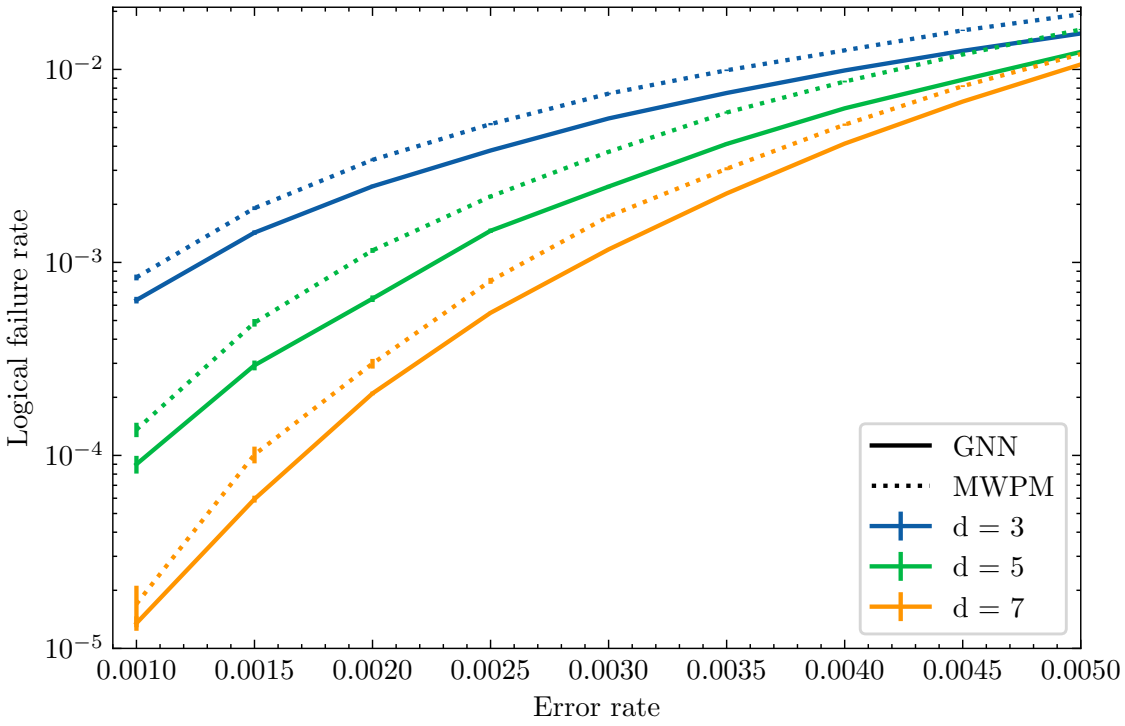


Figure 4.4: Decoding accuracy under circuit level noise. GNN decoder (solid lines) and MWPM (dotted lines). Logical failure rate as a function of the error rates of the data qubits on a logarithmic y-scale. Error bars indicate confidence intervals of one standard deviation.

of the networks influences the decoding accuracy. As discussed above, convergence is harder to achieve the higher the code size, implying that the real potential of the GNN might not be reached yet for larger code sizes.

4.3 Circuit-level noise

Next, networks with the very same architecture (up to an additional temporal node feature as described in section 3.1.2 at the input layer) are trained on syndromes generated from surface code simulations under circuit level noise with the `stim` library. Here, two different scenarios are considered: Firstly, the surface code measurements are repeated $d_t = d$ times, yielding detection events from $d + 1$ time steps as explained in the last paragraph of section 2.2.3. Similar to the previous section, one network per code size is trained on a dataset containing 5 million samples from a range of different error rates $p = 0.001, 0.002, 0.003, 0.004$ and 0.005. Again, 25% of the samples are replaced after each training epoch. The results show that the networks reach higher accuracies than the minimum weight perfect matching algorithm up to code size $d = 7$ (fig. 4.4). For larger code sizes and considering d cycles, graphs grow cubically with d, thus limiting the accuracy achievable with our architecture.

As a second scenario, we vary the number of surface code cycles at a fixed code size, training one network per number of cycles and code size. Each network is trained on a dataset with 5 million samples from a range of different error rates

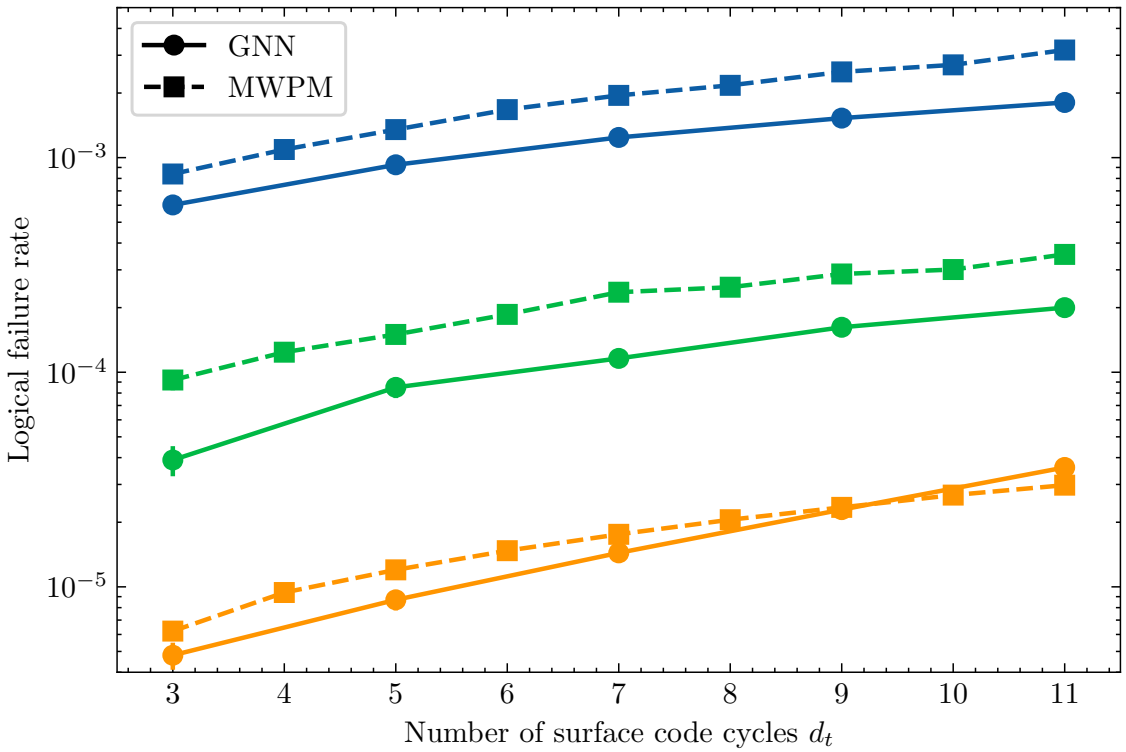


Figure 4.5: Decoding accuracy under circuit level noise ($p = 10^{-3}$): Logical failure rate as a function of the number of stabilizer measurement cycles. GNN decoder (solid lines) and MWPM (dotted lines). Code sizes $d = 3$ (blue), $d = 5$ (green) and $d = 7$ (orange). Error bars indicate confidence intervals of one standard deviation.

$p = 0.001, 0.002, 0.003, 0.004$ and 0.005, replacing 25% of the data each epoch. The networks are then tested on samples generated with an error rate of 10^{-3}. Assuming a constant error probability per surface code cycle ϵ, the failure rate after d_t surface code cycles grows exponentially with the number of cycles. Again, since the graphs do not only grow with the code size but also with the number of cycles d_t, the convergence of the networks gets increasingly difficult with increasing d and d_t. Nevertheless, the networks can beat MWPM in terms of decoding accuracy up to code size 7 with up to 11 cycles.

4.4 Experimental repetition code data

For code sizes 3, 5, and 7, a network was trained on experimental repetition code data from [4]. Training was conducted over 100 epochs for the datasets of size $1.15 \cdot 10^7$, $1.05 \cdot 10^7$ and $9.5 \cdot 10^6$, respectively, with 1% of the datasets reserved for testing (fig. 4.6a). The logical accuracy of the networks tested on the test data reached 70%, 84% and 90% for code sizes 3, 5 and 7, respectively. From the logical error rate $P_{\text{logical}}(d_t)$ after $d_t = 50$ stabilizer measurement cycles, one can determine the logical error rate per cycle ϵ from an expression satisfying the properties $P_{\text{logical}} \to 0.5$ for $d_t \to \infty$ and $P_{\text{logical}} = \epsilon$ for $d_t = 1$ [46]. Furthermore, the probability of a logical

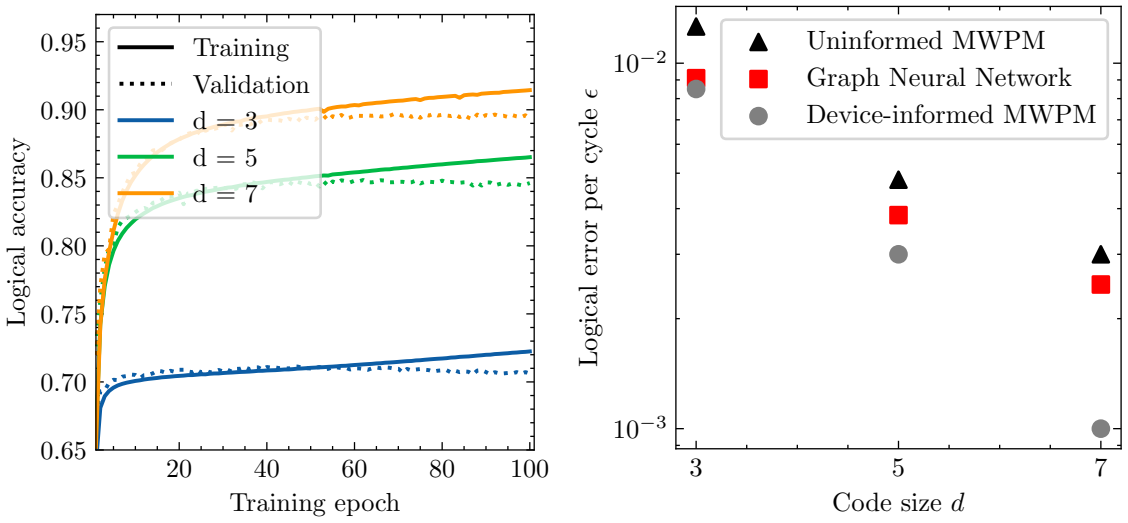


Figure 4.6: Training on experimental repetition code data [4]. **(a)** Training (solid lines) and validation (dotted lines) accuracy as a function of the training epoch. **(b)** Logical error per cycle calculated from eq. (4.2) against the code size: Uninformed minimum weight perfect matching with unit edge weights, graph neural network and device-informed MWPM with edge weights according to the error rates of the individual qubits.

error after $d_t + 1$ cycles is the sum of two channels describing a logical failure after d_t cycles but a success at cycle $d_t + 1$ or a logical success after d_t cycles but a failure at cycle $d_t + 1$:

$$P_{\text{logical}}(d_t + 1) = (1 - \epsilon)\, P_{\text{logical}}(d_t) + \epsilon\,(1 - P_{\text{logical}}(d_t))\,. \tag{4.1}$$

It follows that the logical error rate is exponential in the number of cycles [4]:

$$P_{\text{logical}} = \frac{1}{2}\left[1 - (1 - 2\epsilon)^{d_t}\right]. \tag{4.2}$$

The logical error per cycle as a function of the code size is displayed in fig. 4.6b. Here, we compare the performance of the GNN decoder with two MWPM decoders: The "uninformed" MWPM has no information about the underlying error rates of the physical qubits, i.e. its edge weights are set to one. The "device-informed" MWPM incorporates information about the error rates of data and stabilizer qubits and was specifically tailored to the device of the Google Quantum AI team. The numerical values of its logical error per cycle are taken from fig. 4.b in [4].

Strikingly, the GNN reaches higher logical error rates per cycle than the uninformed MWPM decoder. Even though samples were subsampled from a distance-25 experiment, i.e. the physical qubits and hence the error rates vary between samples, the GNN learns device-specific properties that are only incorporated in the "device-informed" MWPM. This promises a particular practical advantage of a GNN-based decoder: Instead of benchmarking all the qubits and building that information into a decoder, the GNN simply learns the error distribution from experiments without any prior assumptions about the noise model.

5
Conclusion

In this work, we explored the capabilities of a machine-learning-assisted decoder for topological stabilizer codes. A graph neural network was trained on billions of samples of surface code simulations under the assumption of both perfect stabilizer measurements and circuit-level noise and on experimental data from a repetition code experiment. This approach is model-free, i.e. no information about the underlying qubits is incorporated into the network and purely data-driven, i.e. fully leveraged (and limited) by training on large datasets. Decoding the stabilizer codes was brought down to a graph classification problem of finding the most likely logical coset of the underlying error chain. A set of violated stabilizer measurements was mapped to a graph labeled with node features including stabilizer type and position and edge weights capturing distances between two detection events.

First, the surface code with perfect stabilizer measurements was simulated by drawing random depolarizing noise and computing the corresponding syndrome. Graph classification of this syndrome measurements achieved accuracies higher than the minimum weight perfect matching decoder for code sizes up to d = 15. In principle, the graph neural networks of fixed architecture (size) are capable of decoding surface codes of any size. However, it became evident that decoding larger code sizes at a given error rate requires larger networks to process the larger graphs. This upscaling also influences the decoding time, as discussed below.

Second, surface code experiments with circuit-level noise were simulated with `stim`. Training was limited to code size 7 due to the limitations in the complexity of our networks. For those small code sizes, our networks reached logical accuracies higher than MWPM, both as a function of low error rates with $d_t = d$ surface code cycles, but also as a function of the number of surface code cycles at a fixed error rate of $p = 10^{-3}$. These results show that the machine-learning-based, data-driven approach is a possible option for practical quantum error correction under realistic noise scenarios.

Third, experimental repetition code data was used to train networks for code sizes 3, 5 and 7. Compared to an uninformed MWPM decoder, the GNN was able to achieve lower logical error rates per cycle when tested on the test data. Only a device-specific MWPM decoder optimized for the error rates of the qubits can reach higher accuracies. This marks an interesting strength of GNN-based decoders: Instead of investing time and resources to fine-tune the error model and the decoder, one simply trains with a sufficient amount of data and lets the network learn the underlying error model. This method, however, is limited by the amount of available data, limiting this work to small code sizes. It remains an interesting challenge to train a graph neural network on experimental surface code data.

Lastly, the analysis of the inference time shows that the decoding time per sample scales approximately quadratically with the code size, comparable to MWPM. A further overhead from the mapping from stabilizer measurements to a graph is added to the total inference time. However, the latter algorithm could be sped up significantly. Additionally, the GNN-based decoder could potentially be implemented in hardware to allow for fast inference times.

Apart from extending our work to larger code sizes, another future direction of research could be the decoding of different noise models such as biased or non-independent and non-identically distributed noise. Furthermore, one could explore decoding different stabilizer architectures as the $XZZX$ or the XYZ^2 code [47, 48].

Bibliography

[1] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] Richard P. Feynman. "Simulating physics with computers, 1981". In: *International Journal of Theoretical Physics* 21.6/7 (1981).

[3] Eric Dennis et al. "Topological quantum memory". In: *J. Math. Phys. 43* (2001). DOI: 10.1063/1.1499754.

[4] Google Quantum AI. "Suppressing quantum errors by scaling a surface code logical qubit". In: *Nature* 614.7949 (Feb. 2023), pp. 676–681. DOI: 10.1038/s41586-022-05434-1.

[5] Sebastian Krinner et al. "Realizing repeated quantum error correction in a distance-three surface code". In: *Nature* 605.7911 (May 2022), pp. 669–674. DOI: 10.1038/s41586-022-04566-8.

[6] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* (1965). DOI: 10.4153/CJM-1965-045-4.

[7] James R. Wootton and Daniel Loss. "High Threshold Error Correction for the Surface Code". In: *Physical Review Letters* 109.16 (Oct. 2012), p. 160503. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.109.160503.

[8] Adrian Hutter, James R. Wootton, and Daniel Loss. "Efficient Markov chain Monte Carlo algorithm for the surface code". In: *Physical Review A* 89.2 (Feb. 2014), p. 022326. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.89.022326.

[9] Sergey Bravyi, Martin Suchara, and Alexander Vargo. "Efficient algorithms for maximum likelihood decoding in the surface code". In: *Physical Review A* 90.3 (Sept. 2014), p. 032326. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.90.032326.

[10] Karl Hammar et al. "Error-rate-agnostic decoding of topological stabilizer codes". In: *Phys. Rev. A* 105 (4 Apr. 2022), p. 042616. DOI: 10.1103/PhysRevA.105.042616.

[11] Giacomo Torlai and Roger G. Melko. "Neural Decoder for Topological Codes". In: *Physical Review Letters* 119 (3 July 2017), p. 030501. DOI: 10.1103/PhysRevLett.119.030501.

[12] Stefan Krastanov and Liang Jiang. "Deep neural network probabilistic decoder for stabilizer codes". In: *Scientific Reports* 7.1 (2017), p. 11003. DOI: 10.1038/s41598-017-11266-1.

[13] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. "Decoding small surface codes with feedforward neural networks". In: *Quantum Science and Technology* 3.1 (2017), p. 015004. DOI: 10.1088/2058-9565/aa955a.

[14] Christopher Chamberland and Pooya Ronagh. "Deep neural decoders for near term fault-tolerant experiments". In: *Quantum Science and Technology* 3 (2018), p. 044002. DOI: 10.1088/2058-9565/aad1f7.

[15] Paul Baireuther et al. "Machine-learning-assisted correction of correlated qubit errors in a topological code". In: *Quantum* 2 (2018), p. 48. DOI: 10.22331/q-2018-01-29-48.

[16] Nikolas P Breuckmann and Xiaotong Ni. "Scalable Neural Network Decoders for Higher Dimensional Quantum Codes". In: *Quantum* 2 (2018), p. 68. DOI: 10.22331/q-2018-05-24-68.

[17] Philip Andreasson et al. "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.

[18] Ryan Sweke et al. "Reinforcement learning decoders for fault-tolerant quantum computation". In: *Machine Learning: Science and Technology* 2.2 (2020), p. 025005. DOI: 10.1088/2632-2153/abc609.

[19] Xiaotong Ni. "Neural Network Decoders for Large-Distance 2D Toric Codes". In: *Quantum* 4 (Aug. 2020), p. 310. ISSN: 2521-327X. DOI: 10.22331/q-2020-08-24-310.

[20] Nishad Maskara, Aleksander Kubica, and Tomas Jochym-O'Connor. "Advantages of versatile neural-network decoding for topological codes". In: *Physical Review A* 99.5 (2019), p. 052351. DOI: 10.1103/PhysRevA.99.052351.

[21] David Fitzek et al. "Deep Q-learning decoder for depolarizing noise on the toric code". In: *Physical Review Research* 2 (2 May 2020), p. 023230. DOI: 10.1103/PhysRevResearch.2.023230.

[22] Hugo Théveniaut and Evert van Nieuwenburg. "A NEAT Quantum Error Decoder". In: *SciPost Physics* 11 (1 2021), p. 5. DOI: 10.21468/SciPostPhys.11.1.005.

[23] Spiro Gicev, Lloyd CL Hollenberg, and Muhammad Usman. *A scalable and fast artificial neural network syndrome decoder for surface codes*. 2021. arXiv: 2110.05854.

[24] Mengyu Zhang et al. *A Scalable, Fast and Programmable Neural Decoder for Fault-Tolerant Quantum Computation Using Surface Codes*. 2023. arXiv: 2305.15767 [quant-ph].

[25] Michael A. Nielsen et al. *Quantum computation and quantum information - 10. ed.* Cambridge University Press, 2010. ISBN: 9781107002173.

[26] A Yu Kitaev. "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6 (Dec. 1997), p. 1191. DOI: 10.1070/RM1997v052n06ABEH002155.

[27] John Clarke and Frank K Wilhelm. "Superconducting quantum bits". In: *Nature* 453.7198 (2008), pp. 1031–1042.

[28] J. I. Cirac and P. Zoller. "Quantum Computations with Cold Trapped Ions". In: *Phys. Rev. Lett.* 74 (May 1995). DOI: 10.1103/PhysRevLett.74.4091.

[29] Bruce E Kane. "A silicon-based nuclear spin quantum computer". In: *nature* 393.6681 (1998), pp. 133–137.

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

[31] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. 1997. arXiv: quant-ph/9705052 [quant-ph].

[32] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Phys. Rev. A* 86 (3 Sept. 2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324. URL: https://link.aps.org/doi/10.1103/PhysRevA.86.032324.

[33] Lingfei Wu et al. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022, p. 725.

[34] Xun Gao and Lu-Ming Duan. "Efficient representation of quantum many-body states with deep neural networks". In: *Nature Communications* 8.1 (Sept. 2017). DOI: 10.1038/s41467-017-00705-2.

[35] Patrick Reiser et al. "Graph neural networks for materials science and chemistry". In: *Communications Materials* 3.1 (Nov. 2022). DOI: 10.1038/s43246-022-00315-6.

[36] Gage DeZoort et al. "Graph neural networks at the Large Hadron Collider". In: *Nature Reviews Physics* 5.5 (Apr. 2023), pp. 281–303. DOI: 10.1038/s42254-023-00569-0.

[37] Bernhard Mehlig. *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021. DOI: 10.1017/9781108860604.

[38] Christian Forssén. *Learning from data*. URL: https://gitlab.com/cforssen/tif285-book.

[39] Christopher Morris et al. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks". In: *CoRR* (2018). URL: arxiv.org/abs/1810.02244.

[40] Craig Gidney. "Stim: a fast stabilizer circuit simulator". In: *Quantum* 5 (July 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497.

[41] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. DOI: 10.48550/arXiv.1412.6980.

[42] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[43] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG].

[44] Oscar Higgott and Craig Gidney. *PyMatching v2*. https://github.com/oscarhiggott/PyMatching. 2022.

[45] Ramon W. J. Overwater, Masoud Babaie, and Fabio Sebastiano. "Neural-Network Decoders for Quantum Error Correction Using Surface Codes: A Space Exploration of the Hardware Cost-Performance Tradeoffs". In: *IEEE Transactions on Quantum Engineering* 3 (Feb. 2022), pp. 1–19. DOI: 10.1109/tqe.2022.3174017.

[46] "Exponential suppression of bit or phase errors with cyclic error correction". In: *Nature* 595.7867 (2021), pp. 383–387.

[47] J Pablo Bonilla Ataides et al. “The XZZX surface code”. In: *Nature communications* 12.1 (2021), p. 2172.

[48] Basudha Srivastava, Anton Frisk Kockum, and Mats Granath. “The XYZ^2 hexagonal stabilizer code”. In: *Quantum* 6 (2022), p. 698.

DEPARTMENT OF PHYSICS
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden
www.gu.se

UNIVERSITY OF
GOTHENBURG