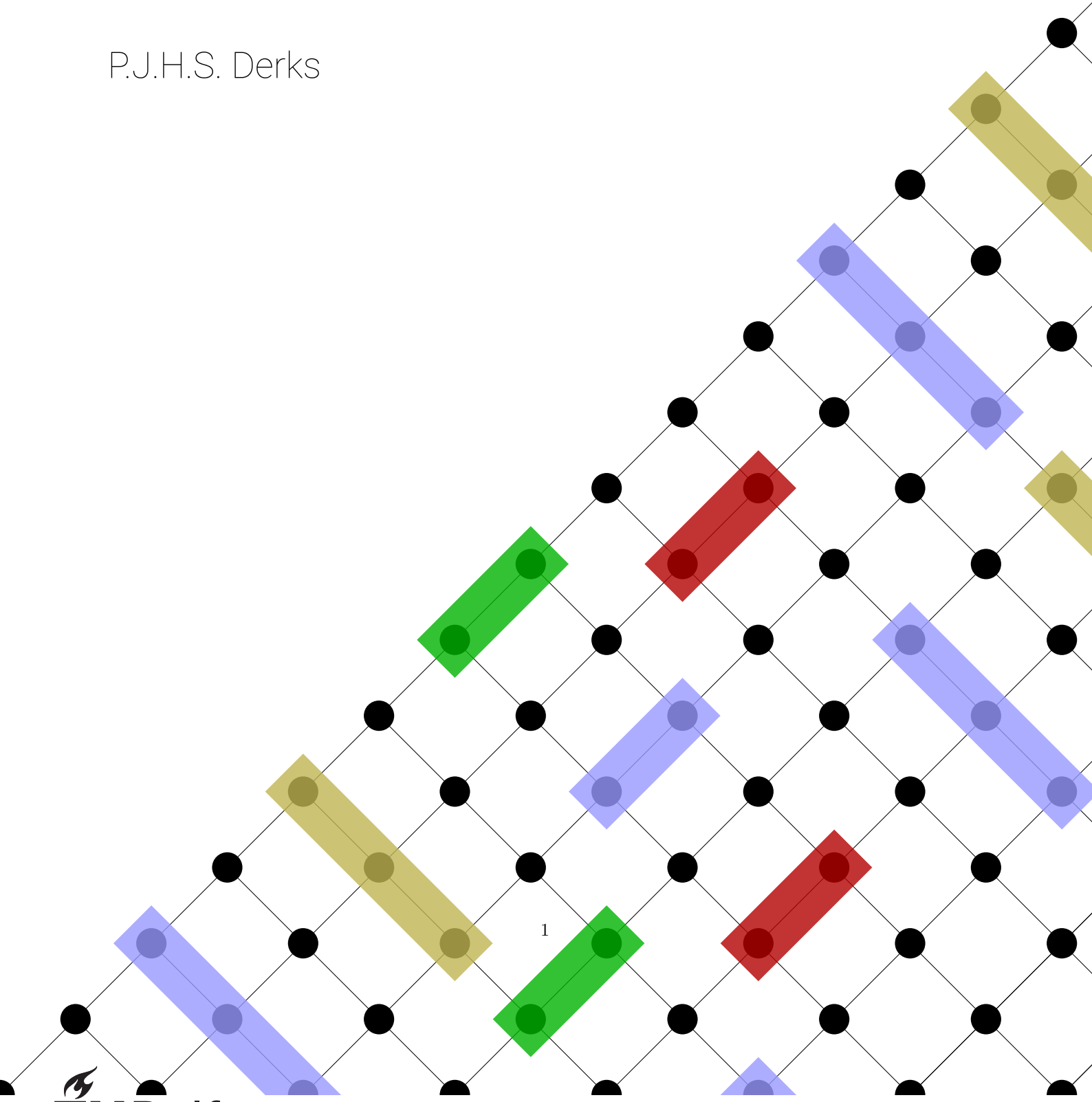
Improving the Pseudo-threshold of the Flag-Bridge Based Steane Code

P.J.H.S. Derks



TU Delft

Improving the Pseudo-threshold of the Flag-Bridge Based Steane Code

by

P.J.H.S. Derks

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday June 17, 2020 at 1:00 PM.

Student number:	4494997	
Project duration:	October 1, 2019 – June 17, 2020	
Thesis committee:	Dr. C. G. Almudever,	TU Delft, supervisor
	Prof. dr. L. DiCarlo,	TU Delft
	Dr. D. Elkouss	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

Fault-tolerant (FT) quantum computation is key for developing large-scale universal quantum computers capable of solving problems that are intractable nowadays. The approach to achieve fault tolerance is based on quantum error correction (QEC), which enables protection of quantum information and FT protocols. An important near-term milestone towards a FT quantum computer is the implementation of distance-3 QEC codes on current Noisy Intermediate-Scale Quantum (NISQ) devices. It has recently been shown that a fault-tolerant version of the Steane code can be mapped to the topologies of QuTech's Surface-17 and IBM's Tokyo devices, using flag-bridge qubits [1]. The goal of this work is to improve the performance (i.e. pseudo-threshold) of the flag-bridge based Steane code by using a different error decoding strategy and optimizing the error syndrome extraction flag-bridge circuits.

More precisely, we propose a decoding strategy that uses both a neural network and look-up table (LUT). The look-up table outputs corrections on the physical level and the neural network outputs corrections for the logical state. We show that with this decoder design, an increase of up 20% of the pseudo-threshold could be achieved when compared to a simple LUT decoder.

We also optimize the error syndrome extraction flag-bridge circuits of the Steane code by decreasing the number of timesteps required to perform the circuits. We show that when extracting different error syndromes in parallel, the resulting circuit remains FT. The highest pseudo-threshold we find for the flag-bridge based Steane code circuit is 1.7×10^{-4} for the circuit level noise model. This is lower than the 4.44×10^{-4} pseudo-threshold of the distance-3 rotated surface code, but flag-bridge circuits remain interesting as they use less qubits.

Acknowledgements

First, I would like to thank my supervisor **Dr. Carmen G. Almudever**. Your contagious enthusiasm during lectures and discussions got me excited and motivated. Thank you for all your valuable advice on research, writing and presenting. You encouraged me to discuss my work with others and to go to talks. Now and then you asked me how things were going non-researchwise and you were always available to answer questions. I would recommend any masters student to join your group.

Secondly, I would like to thank three people that directly helped me with my research. I would like to thank **Dr. Lingling Lao** for helping me during the first months of my project. I am very grateful for your guidance in learning about quantum error correction. I would like to thank **Ramon** for his advice on neural networks and error decoders. Thanks to **Erik** for the assistance with the use of the servers.

Special thanks to **Prof. Leonardo DiCarlo** and **Dr. David Elkouss** for accepting to be part of my thesis committee. Thanks to **Prof. Barbara Terhal** for letting me join the software and theory journal club.

Thanks to everyone in the group. You all made me feel at home and we had a lot of fun together. **Abid**, you motivated me to work hard, but also taught me that I sometimes need to relax. I hope we will have many more beers together. **Nikiforos**, the advice you gave me on how to handle situations during which I was stressed was very valuable. **Matthew**, I enjoyed discussing quantum error correction with you, I wish we had more time to do this. **Diogo**, you are an awesome programmer and it was nice to have someone to talk to who was in the same position. **Medina**, you seem to always be organized and well prepared, I am trying to copy you. **Hans**, your questions and comments were valuable and made me view my work from a different angle.

I would like to thank my friends from Delft Aerospace Rocket Engineering, TAPbeheer and from "het m8ig 8ste" for making my student life in Delft unforgettable. Special thanks to my roommates, **Thijs** and **Julian**, for putting my mind away from work when I was home. **Vincent** and **Berend**, I enjoyed studying together and discussing our thesis projects. Thanks to IM Efficiency for letting me study in their office during the pandemic. Lastly, I would like to thank my family for their support and my longtime friends from Heerlen and Amsterdam for always being there for me.

QuTech was a special place to be. It was a great experience to work together with people from different fields, countries and backgrounds, who are all extremely excited about science. It's unfortunate that during the last months I have not been able to experience the environment at QuTech. To everyone I have met during my time in Delft, I hope that we will cross paths again in the future.

Peter-Jan
June, 2020
Delft, the Netherlands

Contents

1 Introduction

In 1981, at the 1st conference on Physics and Computation, Feynman gave a talk named "Simulating Physics with Computers" [2]. Feynman set three requirements that a computer must satisfy to be able to simulate a physical system. The first requirement was that the simulations must be exact, dismissing numerical algorithms; “the computer will do exactly the same as nature". Feynman’s second requirement was that the size of the computer does not scale exponentially with the space-time volume of a physical system. The third requirement was that the elements of the computer are locally interconnected.

The second requirement rules out the option of using a classical computer to keep track of the probability amplitudes for all possible classical configurations of a quantum system, because the number of probability amplitudes grows exponentially. The natural question that followed is if a probabilistic classical computer should in theory be able to give the same probabilities as the quantum system does.

The answer to this question is no, due to quantum nonlocality. The measurement statistics of a quantum system do not admit an interpretation in terms of a local realistic theory, and therefore Feynman’s third requirement can not be met. The conclusion Feynman reached was that to simulate physics a computer itself must be built of quantum mechanical elements which obey the laws of quantum mechanics, a quantum computer;

> “Nature isn’t classical, dammit, and if you want to make a simulation of nature, you’d better make it quantum mechanical, and by golly it’s a wonderful problem, because it doesn’t look so easy.”

1.1. Progress in quantum computing

In 1985, David Deutsch put Feynman’s idea forward by formulating a quantum version of the Turing machine [3]. Deutsch constructed a simple example, Deutsch’s algorithm, suggesting that it is possible for a quantum computer to solve computational problems which have no efficient solution on a classical computer. Peter Shor and Lev Grover made further promising discoveries giving a strong indication that quantum computers are more powerful than classical computers. Shor showed that the prime factorization problem and the discrete logarithm problem could be solved efficiently on a quantum computer [4]. Grover designed a quantum algorithm that can perform a search through some unstructured search space faster than any known classical algorithm [5]. These problems are still believed to have no efficient solution on a classical computer [6]. Recently, a quantum computer has performed a calculation in 200 seconds, which would take a state-of-the-art classical supercomputer approximately 10,000 year [7]. This dramatic increase in speed compared to all known classical algorithms is an experimental realization of quantum supremacy [8]. This achievement further pushes the belief that a quantum computer can solve problems which have no efficient solution on a classical computer, but does not prove it. The superconducting quantum chip in which the quantum supremacy algorithm was performed is a Noisy Intermediate Scale Quantum (NISQ) device with 53 qubits [7].

Several NISQ devices have successfully been built, but scaling these up to devices that are large enough to be able to do useful computations presents formidable challenges. Current challenges are that quantum information needs to be protected from interactions with the environment and the inaccuracy of quantum operations. A quantum computer that can deal with both issues is said to be fault-tolerant (FT). Theoretically, a step towards solving the first challenge was taken in 1995, when Shor introduced the first Quantum

Error Correction (QEC) code, showing that quantum information can be encoded to protect from errors [9]. A year later Shor further showed that QEC can be performed with inaccurate gates, showing that the second challenge can be solved [10]. FT quantum algorithms are designed such that information is protected and inaccuracies in operations do not lead to unreliable computations. Although large-scale FT quantum computers are still far away, NISQ devices can be used to demonstrate FT principles by the use of small QEC codes.

1.2. Thesis motivation and objective

Theorist have come up with several QEC codes, the two most popular classes being surface codes [11] and color codes [12]. In recent years a color code has been implemented using trapped ions [13] and a simplified version of the surface code has been implemented using superconducting qubits [14]. The circuit implementing the surface code is FT, whereas this is not the case for general circuits implementing color codes. The smallest color code is the Steane code and its circuit can be made fault-tolerant by adding a few extra qubits, called flag qubits [15]. It has been shown that this circuit with flag qubits can be modified such that different fault-tolerant verions can be generated and mapped to existing NISQ devices using a flag-bridge approach. [1]. To perform QEC, a decoder processes the resulting stream of parity measurement results to infer which correction to make. For the surface code, many decoders have been designed using heuristic algorithms or neural networks. The heuristic approach is difficult to apply to flag-bridge error syndrome measurement circuits, as each version requires a different decoding strategy. That is why neural network decoders are preferred in this scheme, as they just need to be properly trained based on the error syndromes observed. Note that neural network based decoders have proven to be both accurate and fast [16–20].

The aim of this work is to improve the performance (in terms of pseudo-threshold) of the Steane code based on flag-bridge circuits [1]. We do this by exploring different neural network decoder approaches and new FT flag-bridge error syndrome extraction circuits. The code used for this thesis can be found on GitHub: FlagBridgeQEC.

1.3. Thesis organization

Chapter 2 starts with a short summary of the concepts behind quantum computing. Then we give an introduction to quantum error correction using the stabilizer formalism. The chapter ends with a section on flag based QEC, in which the flag -bridge syndrome extraction circuits implementing the Steane code are shown.

In Chapter 3 we focus on neural network based decoders. We explain the error decoding process, the motivation for using neural networks and high level and low level neural network based decoder.

In Chapter 4 we first undertake a study of different neural network decoder designs for the flag-bridge circuits. Then, we train a neural network as part of a high level decoder for one of the flag-bridge circuits from [1]. Additionally, in this chapter new FT flag-bridge error syndrome extraction circuits for NISQ devices that take less timesteps are introduced. We compare their pseudo-threshold to the distance 3 rotated surface code.

Finally, in Chapter 5, we summarize the results in this thesis, provide conclusions about our research and give an outlook for future work.

2

Quantum Computing and Quantum Error Correction

In the first section of this chapter the basics of quantum computing are introduced using four basic postulates of quantum mechanics. The second section is on the stabilizer formalism, which is key for understanding quantum error correction (QEC). In the third section the main concepts of QEC are explained, using the Steane code as an example. In the last section flag based fault-tolerant QEC is introduced.

2.1. Quantum computing

2.1.1. Qubits, gates and measurement

In quantum computing, the counterpart of the classical bit is the quantum bit or qubit, a two-level physical system. The first postulate of quantum mechanics states that a qubit lives in a two-dimensional Hilbert space, a complex vector space with inner product. A qubit is completely described by its state vector, which is a unit vector in the Hilbert space. Using two orthonormal vectors, for example $|0\rangle$ and $|1\rangle$, an arbitrary qubit state can be written as

$$|\psi\rangle = a|0\rangle + b|1\rangle\,. \tag{2.1}$$

The second postulate states that the evolution of a qubit is described by a unitary transformation. That is, the state of a qubit at time t_1 is related to the state of the qubit at time t_2 by a unitary operator which only depends on t_1 and t_2:

$$|\psi_{t_2}\rangle = U|\psi_{t_1}\rangle\,.$$

Examples of single qubit operators are the Pauli matrices X,Y and Z and the identity matrix I:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The Pauli matrices together with the identity matrix form a basis for the complex vector space of all 2×2 Hermitian matrices. Two Pauli matrices either commute following $[\sigma_a, \sigma_b] = 2i\varepsilon_{abc}\sigma_c$ or anti-commute following $\{\sigma_a, \sigma_b\} = 2\delta_{ab}I$. In these equations σ can be one of the three Pauli matrices X, Y or Z, ε_{abc} is the Levi-Civita symbol, δ_{ab} is the Kronecker delta and I is the identity matrix. Any quantum system in which one can define X and Z operators that satisfy these relations can be used as a qubit [11].

The third postulate states that measurements of a qubit are described by a set of measurement operators: $\{M_m\}$, where m refers to the measurement outcome. The probability of outcome m when measuring a qubit in state $|\psi\rangle$ is given by

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle\,. \tag{2.2}$$

The measurement operators must satisfy the completeness equation because the probabilities $p(m)$ must sum to one:

$$\sum_m M_m^\dagger M_m = I. \tag{2.3}$$

The qubit state after the measurement is

$$|\psi\rangle = \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{2.4}$$

A common set of measurement operators, describing measurements in the so-called computational basis is given by $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$. When measuring the state in Eq. 2.1 the probabilities of the measurement outcomes are $p(0) = a^2$ and $p(1) = b^2$ with post measurement states $|0\rangle$ and $|1\rangle$, respectively.

Computations are done with multiple qubits and the fourth postulate explains how to describe multiple qubit systems. The Hilbert space associated with multiple qubits is the tensor product of the Hilbert spaces of the individual qubits. If we prepare n qubits individually, the resulting state is given by

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes ... \otimes |\psi_n\rangle .$$

Not every state can be written as the tensor product of single qubit states. When this is not possible the qubits are said to be entangled. A gate acting on n qubits is described by a $2^n \times 2^n$ unitary matrix. Examples of two qubit gates are CNOT and CZ:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

2.1.2. Quantum circuits

Operators applied to qubits are depicted using quantum circuits. Figure 2.1 shows the circuit for teleporting a qubit. Circuits are read from left to right and qubits are depicted by horizontal lines. The states on the left side of the circuit are the input states and on the right side are the output states. The first two gates are CNOTs. After the qubits are measured they are drawn as a classical control wire with a double line.

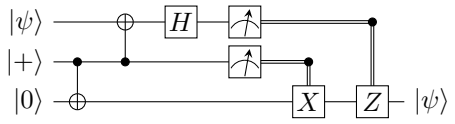


Figure 2.1: Quantum teleportation circuit.

Calculating the output of a circuit where a 2-qubit gate is applied to two qubits generally requires multiplying a 4×1 vector with a 4×4 matrix, which requires 64 multiplications. A slightly larger circuit where a 3-qubit gate is applied to three qubits requires multiplying 512 numbers. Thus, the time required to simulate quantum circuits grows exponentially. In the next section a formalism will be shown with which certain quantum circuits can be simulated efficiently on a classical computer.

2.2. Stabilizer Formalism

2.2.1. Schrödinger and Heisenberg picture in quantum mechanics

In the Schrödinger picture, to calculate the time-dependence of the expectation value of a physical observable

$$\langle\psi(t)| \hat{A} |\psi(t)\rangle \ , \tag{2.5}$$

the Schrödinger equation can be used to find the time-dependent wave function:

$$i\hbar \frac{\partial |\psi(t)\rangle}{\partial t} = \hat{H} |\psi(t)\rangle . \tag{2.6}$$

In the Heisenberg picture, to calculate the time-dependence of the expectation value, time-dependence is assigned to the physical observable and the wave function is time-independent

$$\langle A(t)\rangle = \langle\psi| \hat{A}_H(t) |\psi\rangle . \tag{2.7}$$

The Heisenberg equation of motion can be used to find the time-dependent operator. Using the Schrödinger equation, this equation can be derived from Eq.2.5. [21]:

$$\frac{\partial\langle A\rangle}{\partial t} = \left\langle \frac{\partial\psi}{\partial t}|\hat{A}|\psi \right\rangle + \left\langle \psi|\hat{A}|\frac{\partial\psi}{\partial t} \right\rangle \tag{2.8}$$

$$\frac{\partial\langle A\rangle}{\partial t} = \frac{i}{\hbar}\langle\psi|[\hat{H},\hat{A}]|\psi\rangle \tag{2.9}$$

$$\frac{\partial\hat{A}_H}{\partial t} = \frac{i}{\hbar}\left[\hat{H},\hat{A}_H\right] \tag{2.10}$$

This shows that the pictures are equivalent and we thus have the luxury of choice between the two pictures.

2.2.2. Heisenberg picture of quantum computing

Gottesmann introduced the main idea of the Heisenberg picture, assigning time dependence to operators, to quantum computing [22]. This lead to the stabilizer formalism, which is a useful tool for understanding quantum error correction codes. Instead of working with a quantum state itself, the stabilizer formalism works with a group of operators that describe states. Before applying the stabilizer formalism several definitions need to be introduced[1]. The n-qubit Pauli group $\mathscr{P}_n$ is defined as:

$$\mathscr{P}_n \equiv \{\pm I, \pm iI\} \times \{I, X, Y, Z\}^{\otimes n}.$$

The Pauli group for one qubit is therefore:

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

An n-qubit stabilizer group $\mathscr{S}$ is defined as a subgroup of the Pauli group which does not contain $-I$ and in which all elements commute:

$$\mathscr{S} \equiv \{S_i\} \text{ s.t. } -I \notin \mathscr{S} \text{ and } \forall S_i, S_j \in \mathscr{S}, \left[S_i, S_j\right] = 0.$$

Elements of a stabilizer group are called stabilizer operators. Stabilizer states are defined as +1 eigenstates of all elements of a stabilizer group. To discover the useful properties of the stabilizer groups, consider the following example:

$$\mathscr{S}_{\text{Bell}} = \{II, XX, ZZ, -YY\}$$

This group has one stabilizer state, the Bell state:

$$|\psi_{Bell}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.11}$$

Notice that we can define our stabilizer group with fewer operators by using the generators of the group:

$$\mathscr{G}_{\text{Bell}} = \langle XX, ZZ\rangle$$

Every element in $\mathscr{S}_{\text{Bell}}$ can be written as a product of elements in $\mathscr{G}_{\text{Bell}}$. The order of group $\mathscr{S}$ is defined as the number of elements in it and denoted with $|\mathscr{S}|$. In general, a stabilizer group with order $|\mathscr{S}|$ has a generating group with $\log(|\mathscr{S}|)$ generators. If a state is an eigenstate of all stabilizer generators, it is an eigenstate of all stabilizer operators. Notice that the choice of generators is not unique, in our example $\langle XX, -YY\rangle$ also generates $\mathscr{S}_{Bell}$.

A linear combination of linearly independent stabilizer states is also a stabilizer state and therefore stabilizer states form a subspace. Two qubit states span a four dimensional Hilbert spaces with basis vectors

$$|00\rangle, |01\rangle, |10\rangle, |11\rangle.$$

The stabilizer generator ZZ divides the four dimensional Hilbert space into two orthogonal subspaces; the subspace spanned by the +1 eigenvectors ($|00\rangle, |11\rangle$) and the subspace spanned by the -1 eigenvectors ($|01\rangle, |10\rangle$).

[1] Appendix 2 of the textbook “Quantum Computation and Quantum Information" [6] contains the relevant group theory background needed to understand this section. The textbooks “Quantum Error Correction" [23] and “Quantum Computation with Topological Codes: from qubit to topological fault-tolerance" [24] provide a good introduction to the stabilizer formalism .

A different set of basis vectors for two qubit states is

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}, \frac{|00\rangle - |11\rangle}{\sqrt{2}}, \frac{|01\rangle + |10\rangle}{\sqrt{2}}, \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{2.12}$$

The stabilizer generator XX divides the four dimensional Hilbert space into two orthogonal subspaces; the subspace spanned the +1 eigenvectors $\left(\frac{|00\rangle+|11\rangle}{\sqrt{2}}, \frac{|01\rangle+|10\rangle}{\sqrt{2}}\right)$ and the subspace spanned by the -1 eigenvectors $\left(\frac{|00\rangle-|11\rangle}{\sqrt{2}}, \frac{|01\rangle-|10\rangle}{\sqrt{2}}\right)$. Together the two stabilizer generators divide the four-dimensional space into the orthogonal one-dimensional spaces in 2.12 each associated with different eigenvalues (+1+1, +1-1, -1-1, -1+1). The Bell state Eq in 2.11 is the +1+1 eigenvector.

In general, any n-qubit stabilizer generator divides a 2^n dimensional Hilbert space into two orthogonal subspaces. Because stabilizer operators commute with each other, k n-qubit stabilizer generators divide a 2^n dimensional space into 2^k orthogonal subspaces. As we have seen for $n = 2$ with $k = 1$ and $k = 2$, the dimension of the space spannend by stabilizer states is 2^{n-k}.

If we remove generator XX from $\mathcal{G}_{\text{Bell}}$, we are left with $\langle ZZ\rangle$ and a subspace V_s spanned by stabilizer states $|00\rangle$ and $|11\rangle$. An arbitrary vector is this subspace is written as

$$|\psi\rangle = a|00\rangle + b|11\rangle = a|0_L\rangle + b|1_L\rangle,$$

where the L stands for logical. We can address the degrees of freedom in a stabilizer subspace using so-called logical operators that commute with all stabilizer generators. Here a convenient choice of logical operators is $L_X = XX$ and $L_Z = ZI$. These logical operators satisfy the relations introduced in section 2.1.1 and therefore a two qubit system which is in V_s can be used as a single qubit. To see this consider how L_X and L_Z change the state of $|\psi_L\rangle$:

$$L_X|\psi_L\rangle = a|1_L\rangle + b|0_L\rangle = X|\psi_L\rangle$$
$$L_Z|\psi_L\rangle = a|0_L\rangle - b|1_L\rangle = Z|\psi_L\rangle$$

In general for the case with $k < n$, the degrees of freedom in the stabilizer subspace can be addressed with logical operators. These logical operators need to be chosen such that they commute with all stabilizer generators and together with the stabilizer generators form a linearly independent set.

2.2.3. Unitary gates on stabilizer operators

In the stabilizer formalism the action of a quantum gate is represented as a transformation of stabilizer generators. Suppose that the input state $|\psi_i\rangle$ of a circuit represented by U is a stabilizer state of S, then

$$\begin{aligned} |\psi_f\rangle &= U|\psi_i\rangle \\ &= US|\psi_i\rangle \\ &= USU^\dagger U|\psi_i\rangle \\ &= USU^\dagger|\psi_f\rangle. \end{aligned}$$

Thus the final state $|\psi_f\rangle$ is an eigenstate of $USU^\dagger$. Figure 2.2 shows an example of how a circuit transforms a stabilizer. The input state is stabilized by IZ and the output state is stabilized by XZ.

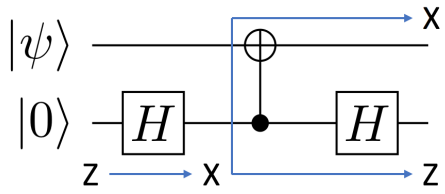


Figure 2.2: Example of how a quantum circuit transforms a stabilizer.

Not all unitary operators can can be described with the stabilizer formalism. The properties of a stabilizer group depend on the fact that its elements are in the Pauli group. Therefore, the unitary operators N that can be described must transform a Pauli operation into another Pauli operation, i.e.

$$NPN^\dagger \in \mathscr{P}_n \text{ for all } P \in \mathscr{P}_n.$$

The group of unitary operators that satisfy this requirement is called the Clifford group. Conveniently, all gates we need to consider for QEC are in the Clifford group.

2.2.4. Pauli basis measurements on stabilizer states

The stabilizer formalism gives an elegant description of measurements of observables in the Pauli group and can be used to efficiently simulate measurement outcomes on a classical computer. If a measured operator P commutes with all generators of the stabilizer group, P or $-P$ is an element of the stabilizer group.

For simplicity, assume the measured operator P is a single qubit Pauli operator and anticommutes with generator g_1 and commutes with all others. P can be written as $P_+ - P_-$ and similarly $I = P_+ + P_-$, where $P_\pm$ is the projector onto the eigenspace of P with eigenvalue $\pm$. We can thus write these projectors as $P_\pm = (I \pm P)/2$. The measurement probabilities are then

$$\begin{aligned} p(+1) &= \mathrm{tr}\left(\tfrac{I+P}{2}|\psi\rangle\langle\psi|\right) \\ p(-1) &= \mathrm{tr}\left(\tfrac{I-P}{2}|\psi\rangle\langle\psi|\right) \end{aligned}$$

Using the facts that $g_1|\psi\rangle = |\psi\rangle$ and $Pg_1 = -g_1P$ gives

$$\begin{aligned} p(+1) &= \mathrm{tr}\left(\frac{I+P}{2}g_1|\psi\rangle\langle\psi|\right) \\ &= \mathrm{tr}\left(g_1\frac{I-P}{2}|\psi\rangle\langle\psi|\right) \end{aligned}$$

Thus $p(+1) = p(-1) = 1/2$. After measurement the new state is $\left|\psi^\pm\right\rangle \equiv (I \pm g)|\psi\rangle/\sqrt{2}$ with stabilizers $\langle \pm P, g_2, \ldots, g_l\rangle$.

In general, the rules for updating the stabilizers after a measurement (and correction if the result is -1) are as follows:

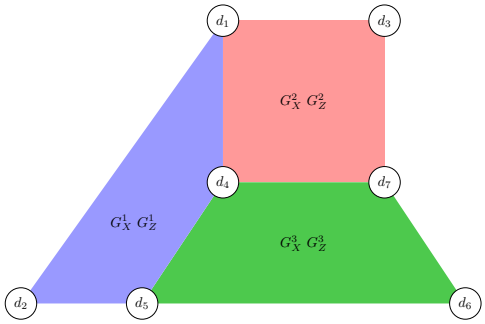
1. Identify and remove $g \in G$ satisfying $\{g, P\} = 0$.
2. Add P to the stabilizer.
3. For each $g \in G$ leave g alone if $[g, P] = 0$ and replace g with gP if $\{g, p\} = 0$.

2.3. Quantum error correction

On noisy devices qubits and operations are imperfect, causing errors during computations. This section explains how to protect quantum information from these errors. QEC is inspired in classical error correction in which information is encoded and errors are detected and corrected. However, due to three fundamental properties of quantum mechanics, how we protect quantum information is different than in classical error correction. These properties are that we can not clone quantum states (information cannot be copied), measurement collapses quantum states and that quantum noise is continuous. In the early beginning of quantum information science these properties lead to scepticism whether quantum computers would ever be able to do calculations in practice [25].

This scepticism was overcome by three findings. The most important one being that we can use entanglement to fight entanglement. Entanglement with the environment can damage the information in a qubit, but by entangling multiple qubits the information can be protected. The entangled qubits are referred to as data qubits and together form a so-called logical qubit[2]. The second finding is that when we entangle information in multiple qubits, we do not destroy this information if we measure a subset of the qubits. The last finding is that we can digitalize errors that occur during a computation. If an arbitrary error transforms the state of a qubit that was initially in the state $|\psi\rangle = a|0\rangle + b|1\rangle$, the resulting state can be written as a linear combination of the following four states:

[2]This is not the case for all QEC codes. In bosonic QEC codes qubits are encoded in different energy levels of a physical system. Here and in the remainder of this thesis we restrict our focus to stabilizer codes.



Name	Operator
G_X^1	$X_1X_2X_4X_5$
G_X^2	$X_1X_3X_4X_7$
G_X^3	$X_4X_5X_6X_7$
G_Z^1	$Z_1Z_2Z_4Z_5$
G_Z^2	$Z_1Z_3Z_4Z_7$
G_Z^3	$Z_4Z_5Z_6Z_7$

Figure 2.3: (Left) The layout of the Steane code. Data qubits are on the vertices and each plaquette represents an X and Z stabilizer. (Right) the six stabilizer generators.

1. no error occurs: $I|\psi\rangle = a|0\rangle + b|1\rangle$.
2. The qubit flips: $X|\psi\rangle = b|0\rangle + a|1\rangle$.
3. The relative phase flips: $Z|\psi\rangle = a|0\rangle - b|1\rangle$.
4. The qubit and relative phase flip: $XZ|\psi\rangle = a|1\rangle - b|0\rangle$.

By using measurements to project a qubit state to one of the four states and to inform us which of the four states the qubit is in, we can detect and correct arbitrary errors.

To see how QEC works it is helpful to study a particular code. While there are numerous codes that allow for error detection and correction, in this thesis we will focus on the Steane code. The Steane code has distance three, meaning that it can correct any single qubit error on data qubits. It is the smallest version of the 4.8.8 color code and encodes one logical qubit with seven data qubits. The 4.8.8 color code is a promising code because it allows for a transversal implementation of the entire Clifford group [26]. To understand why the Steane code works, it will be helpful to first understand how the classical $[7,4,3]$ Hamming code works.

The $[7,4,3]$ Hamming code is a linear code for which error-correction is most easily understood by defining the code in terms of the parity check matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

The codewords are defined as 7-element vectors x for which

$$Hx = 0 (\mathrm{mod}\, 2)$$

There are 16 codewords in total, so the code encodes 4 bits of information in 7 bits [27].

We define a vector v as an unknown codeword x on which an unknown bit flip e_i has occured, where e_i is a 7-element vector with a 1 in the ith place and zeros elsewhere. To correct this bit flip we apply H to find $H(x+e_i) = H(e_i)$, which is the ith column of H. Since all of the columns of H are distinct we know which bit has flipped and can correct it.

The Steane code exploits the classical error-correcting properties of the $[7,4,3]$ code. Using stabilizers a small subspace of the Hilbert space is designated as the code subspace. All of the errors that we want to correct move the code space to mutually orthogonal error subspaces. This means that each single qubit X and Z operator anti-commutes with a unique subset of the stabilizer group. From the rows of the parity matrix we choose six stabilizers that define our code space, shown in Figure 2.3.

In the next sections it is shown how to encode a logical qubit in the code space, how to detect errors and how to do error decoding.

2.3.1. Circuits for encoding, error detection and correction

The circuit in Figure 2.4 shows an overview of the different sub-circuits needed to perform QEC with the Steane code. The first step is to encode the data qubits. This is followed by measuring the six stabilizers, using ancilla qubits to detect errors. The last step is to decode the measurement results and apply corrections to the data qubits.

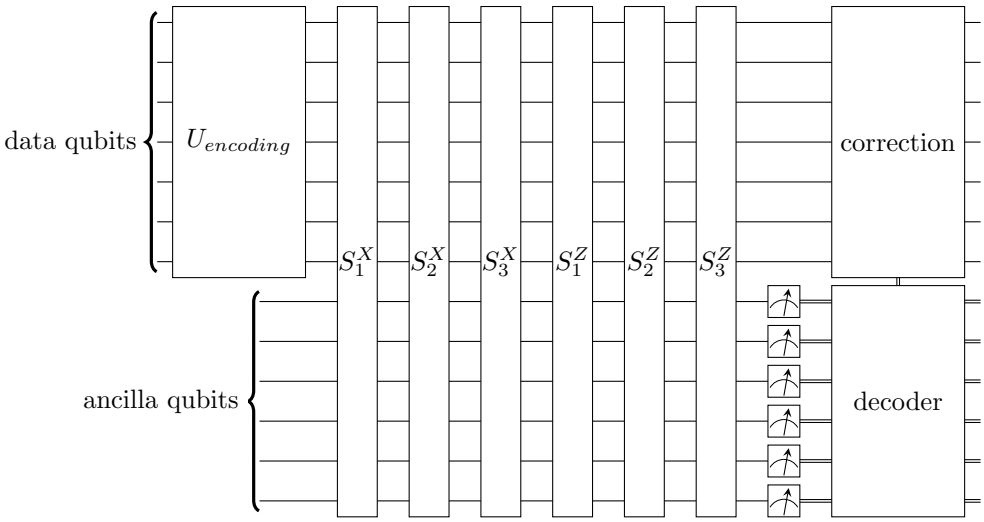
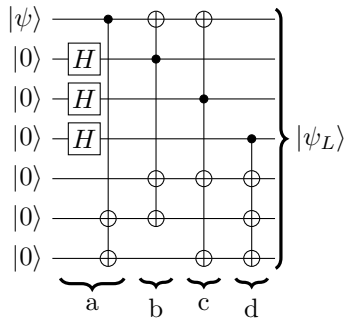


Figure 2.4: Overview of the different steps needed to perform QEC with the Steane code.

The circuit in Figure 2.5 encodes an arbitrary input state, into a logical qubit using the Steane code. That the circuit encodes a qubit is shown by tracking the stabilizer generators of the input state. The stabilizer generators of the output state are not the exact same as in Figure 2.3, but they are just a difference set of generators of the same stabilizer group and thus have the same stabilizer states. In other words, the stabilizer generators in Figure 2.5 have the same error correcting properties because they are just a relabeling of the qubits.



S	a	b	c	d
Z_2	X_2	$X_1X_2X_5X_6$	$X_1X_2X_5X_6$	$X_1X_2X_5X_6$
Z_3	X_3	X_3	$X_1X_3X_5X_7$	$X_1X_3X_5X_7$
Z_4	X_4	X_4	X_4	$X_4X_5X_6X_7$
Z_5	Z_5	Z_2Z_5	$Z_2Z_3Z_5$	$Z_2Z_3Z_4Z_5$
Z_6	Z_1Z_6	$Z_1Z_2Z_6$	$Z_1Z_2Z_6$	$Z_1Z_2Z_3Z_6$
Z_7	Z_1Z_7	Z_1Z_7	$Z_1Z_3Z_7$	$Z_1Z_3Z_4Z_7$

Figure 2.5: A circuit for encoding a logical qubit with the Steane code and corresponding generators.

The circuit shown in Figure 2.6 is called a syndrome extraction circuit and it used for measuring the stabilizers of a QEC code. The input to the circuit can be written in terms of the eigenstates, s_{+1} and s_{-1}, of the stabilizer S being measured:

$$|\psi_{in}\rangle = \lambda_{-1}|s_{-1}\rangle + \lambda_{+1}|s_{+1}\rangle$$

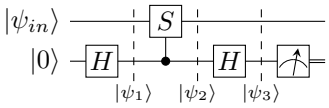
Since $S|s_{\pm1}\rangle = \pm|s_{\pm1}\rangle$ and using $|\psi_3\rangle$ in Figure 2.6 it is clear that the output state of the circuit will be $|\psi_{out}\rangle = |s_{\pm1}\rangle$ with probability $|\lambda_{\pm1}|^2$. Thus the output state of the circuit is stabilized by S.

In Figure 2.7 the error syndrome circuit is shown with a Pauli error on the data qubit. The state of the qubits after the second Hadamard gate is

$$\frac{1}{2}(P+SP)|\psi_{in}\rangle|0\rangle + \frac{1}{2}(P-SP)|\psi_{in}\rangle|1\rangle.$$

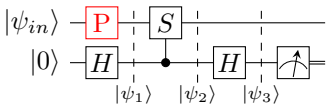
Thus if the error P anti-commutes with the stabilizer, the ancilla qubit will trigger, i.e. the measurement results will be -1.

In Figure 2.8 the syndrome extraction circuits of all stabilizers of the Steane code are shown. Because every single qubit X and Z error anti-commutes with a unique set of stabilizers, they cause a unique set of measurement results. We call these measurement results syndromes.



$$|\psi_0\rangle = |\psi_{in}\rangle|0\rangle$$
$$|\psi_1\rangle = |\psi_{in}\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$|\psi_2\rangle = \frac{1}{\sqrt{2}}|\psi_{in}\rangle|0\rangle + \frac{1}{\sqrt{2}}S|\psi_{in}\rangle|1\rangle$$
$$|\psi_3\rangle = \frac{1}{2}(I+S)|\psi_{in}\rangle|0\rangle + \frac{1}{2}(I-S)|\psi_{in}\rangle|1\rangle$$

Figure 2.6: Syndrome extraction circuit and the resulting quantum state.



$$|\psi_0\rangle = |\psi_{in}\rangle|0\rangle$$
$$|\psi_1\rangle = P|\psi_{in}\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$|\psi_2\rangle = \frac{1}{\sqrt{2}}P|\psi_{in}\rangle|0\rangle + \frac{1}{\sqrt{2}}PS|\psi_{in}\rangle|1\rangle$$
$$|\psi_3\rangle = \frac{1}{2}(P+SP)|\psi_{in}\rangle|0\rangle + \frac{1}{2}(P-SP)|\psi_{in}\rangle|1\rangle$$

Figure 2.7: Syndrome extraction circuit on which a Pauli error has occurred and the resulting quantum state.

These six syndrome extraction circuits can also be used to prepare a set of data qubits in the logical states $|0_L\rangle$ and $|1_L\rangle$. This can be seen from the fact that the output of the circuit is a stabilizer state of the Steane code. For the purpose of building a quantum computer, being able to prepare the data qubits in a logical state is essential. [23]

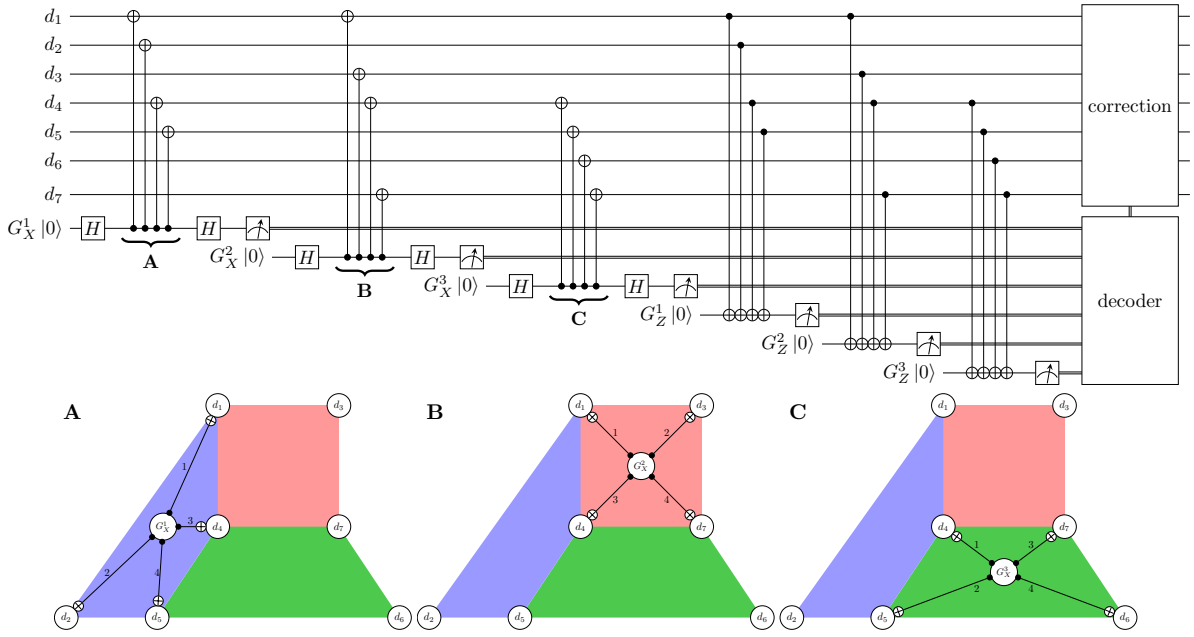


Figure 2.8: The top of the figure shows the quantum circuit for measuring the generators of the Steane code, to give the error syndrome. The bottom of the figure shows a circuit layout illustration with the scheduling of the CNOT gates

The task of the error decoder consists of determining the most likely error corresponding to an observed syndrome. For the circuit in Figure 2.8 this task is simple, but for the error correction circuits shown in the coming sections the task becomes more difficult. Even if the output of the decoder is the most likely error and restores the state to the code space, the logical state can undergo a logical error. For the circuit in Figure 2.8 this is the case when two errors of the same type occur on the data qubits. For example if an X error occurs on data qubit 1 and 2, this will change the measurement result of the ancilla measuring S_Z^2. The correction

applied, which is the most likely error for this syndrome, is X_3. $X_1X_2X_3$ is a logical operator and we are thus left with a logical error.

2.3.2. Syndrome extraction circuit is not fault-tolerant

A disadvantage of color codes compared to surface codes is that the syndrome extraction circuits are prone to errors on a single gate that cannot be corrected. Some errors on the second and third CNOT gate in the syndrome extraction circuit of the stabilizer S_Z^3 will be uncorrectable. In Figure 2.9 an example of an uncorrectable error on the second and third CNOT is given. In the circuit on the left a IZ error occurs on the second CNOT gate. This error will propagate to a weight two error Z_4, Z_5 on the data qubits. In the circuit on the right a ZZ occurs after the third CNOT gate, which will propagate to the same error, Z_4, Z_5. Z_4, Z_5 anti-commutes with $S_X^2 = X_1X_3X_4X_7$ and will thus fire the second X ancilla. The error Z_3 also only anti-commutes with this syndrome. In general, the weight two errors XX, ZZ, XY and ZY do not anti-commute with a unique set of stabilizers and can therefore not be corrected. This means that although only an error has occured on a single gate we can not find out the state of our logical qubit.

If a Z error occurs after the first CNOT or the last CNOT, this will not cause an uncorrectable error. An IZ error on the first qubit leads to the error $Z_4Z_5Z_6$, but this is correctable because $Z_4Z_5Z_6 \times S_Z^3 = Z_7$. Any error on the last CNOT will not propagate to an error on multiple data qubits.

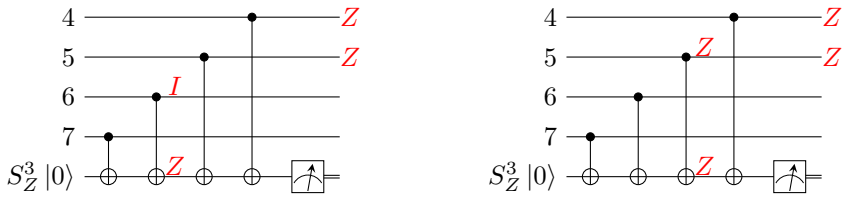


Figure 2.9: Examples of possible uncorrectable errors in a syndrome extraction circuit.

In the next section we will introduce a protocol for the Steane code with which each single error can be corrected. Such a protocol is said to be fault-tolerant.

2.4. Flag based QEC

In the previous section it was shown that the Steane code with 6 ancilla qubits does not make a quantum computer resistant to errors. The circuit needs to be adapted such that it safely performs error correction when the gates used are themselves noisy. A protocol for which this holds is said to be fault-tolerant. In the next two sections we first introduce some definitions and a formal requirement for arbitrary distance fault tolerant QEC. Afterwards the smallest fault tolerant protocol for the Steane code is introduced.

2.4.1. Requirements for fault-tolerant distance-3 quantum error correction

The different actions used by a quantum computer are preparation, single- and two-qubit quantum gates and measurements. Each instantiation of one of these actions is called a location. A qubit which is not used at a timestep in a circuit is idling, which is also counted as a location. A fault is any of these locations not performing the operation it should. A quantum computer is fault-tolerant if it can perform all operations on a state encoded in a quantum code without losing the code's protection against errors and if it can perform error correction when the gates used are themselves noisy. In the next section, we give the requirements for fault-tolerant distance three QEC. The requirements for other fault-tolerant operations and an insightful graphical notation are given in [22].

For fault-tolerant distance-3 QEC two requirements need to be met. Firstly, if the input state of the error correction circuit is a logical state $|\psi_L\rangle$ with a single qubit error and no faults occur during the circuit, the output state should be $|\psi_L\rangle$. The second requirement is that if the input state is $|\psi_L\rangle$ and a single fault happens on any location during the circuit, the output state is $|\psi_L\rangle$. A graphical representation of the requirements is given in Figure 2.10.

2.4.2. Fault-tolerant Steane Code

For the Steane code, the first requirement is satisfied by the circuit shown in Figure 2.8. As explained earlier, the second requirement is not satisfied, for example due to possible hook errors. To solve this issue, multiple

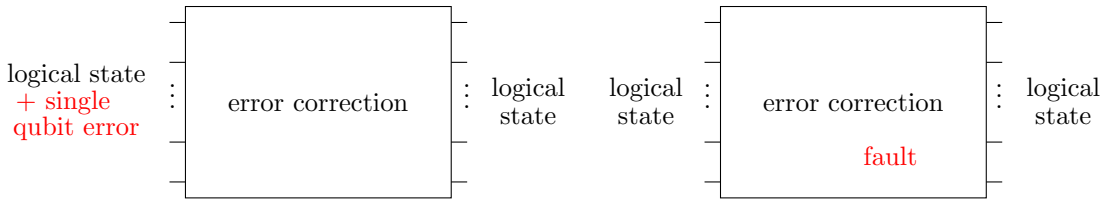


Figure 2.10: Graphical representation of the requirements for fault-tolerant distance-3 QEC.

syndrome extraction circuits with more ancilla qubits have been designed. However as qubits are a scarce resource on NISQ devices, we are interested in syndrome extraction circuits with low qubit overhead for near term implementation.

Shor was the first to propose a solution by entangling ancilla qubits in a Cat state. This requires 4 ancilla qubits for the Cat state and 1 ancilla qubit to verify the Cat state, shown in Figure 2.11a [28]. Steane and Knill proposed methods where the ancilla qubit is encoded as a logical qubit, but they require a high amount of ancilla qubits and are therefore not applicable to current NISQ devices [29] [30]. Improving upon Steane's method, DiVincenzo and Aliferis use a decoding trick on the Cat state to reduce the number of ancilla qubits to 4, shown in Figure 2.11b [31].

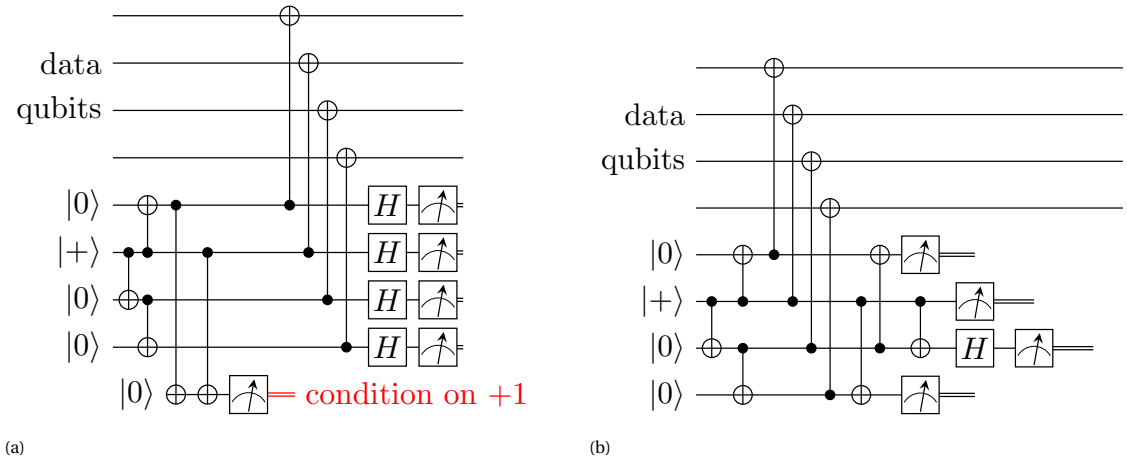


Figure 2.11: Two fault-tolerant syndrome extraction circuits: (a) Shor's circuit with 5 ancilla qubits and (b) a modified version using the DiVincenzo-Aliferis decoding trick with 4 ancilla qubits.

More recently, Yoder and Kim showed that for the Steane code, two ancilla qubits are sufficient [32]. The second ancilla qubit, next to the one needed to measure the syndrome, is called a flag qubit and circuits that use a flag qubit are referred to as flag circuits. Flag circuits are designed such that if an error occurs on the ancilla qubits that will spread to data qubits, this will cause the flag qubit to fire. The circuits will be shown and explained in the next section. The method used by Yoder and Kim can be generalized for any distance-3 code and thus for any code only two ancilla qubits are required [15]. It has been shown that under depolarizing noise, the flag qubit protocol outperforms Shor- and Steane-style error correction for the Steane code [15].

2.4.3. Syndrome extraction circuits with flag and bridge qubits

In Section 2.3.2 it was explained that certain errors on the second and third CNOT in the syndrome extraction circuit in Figure 2.9 will lead to uncorrectable errors on the data qubits. By entangling a flag qubit with the ancilla qubit, it can be detected when this occurs. The flag syndrome extraction circuit for measuring S_Z^3 is shown in 2.12a. If a Z error occurs after the second or third CNOT gate, it will propagate to the flag qubit, be translated by the Hadamard gate to an X error, and will cause the measurement result to be -1.

Chao and Reichardt [15, 33] have shown that it is possible to measure multiple stabilizers in parallel for the Steane code. The flag syndrome extraction circuits for measuring two stabilizers and three stabilizers are shown in Figures 2.13a and 2.14a. The flag syndrome extraction circuits are designed such that when no error occurs they behave the same as the circuits with a single ancilla qubit and such that when an error occurs that

can spread to an uncorrectable error this is detected by the flag qubit. In section 2.4.4 a formal requirement for the design of flag circuits is given.

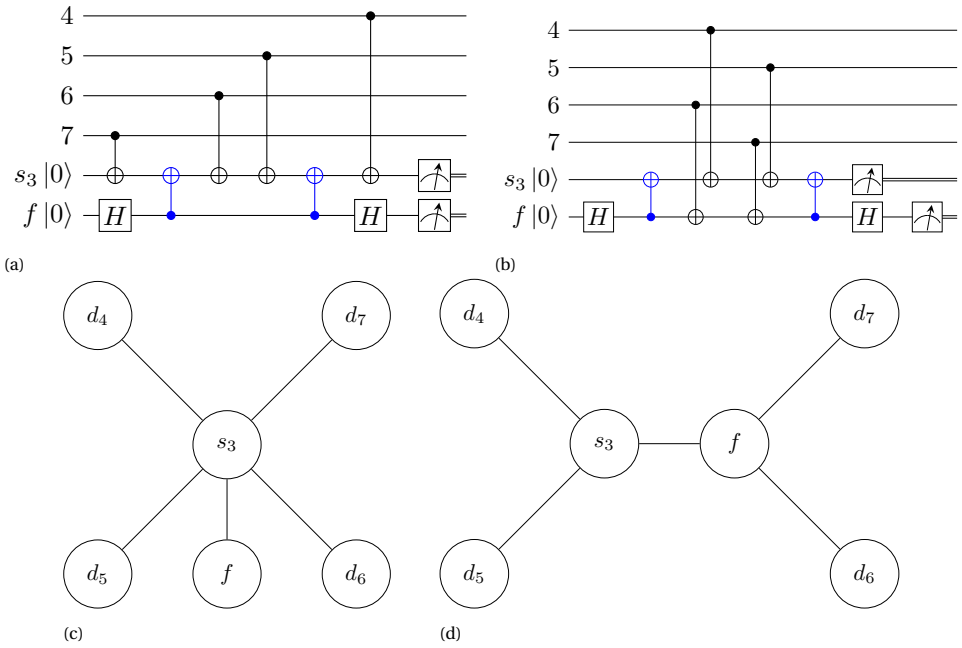


Figure 2.12: (a) Flag syndrome extraction circuit for measuring one stabilizer and (b) the corresponding flag-bridge syndrome extraction circuit. (c) Shows the connectivity graph of the flag syndrome extraction circuit and (d) shows the connectivity graph of the flag-bridge syndrome extraction circuit.

The circuits designed by Chao and Reichardt require a high connectivity between qubits and therefore cannot be directly mapped onto NISQ processors such as the IBM Q Tokyo processor (Tokyo) [34] and the Surface-17 transmon processor (Surface-17) [35]. Lao and Almudever have extended the set of available flag circuits for the Steane code with circuits that require a lower degree of connectivity between qubits. They introduce the concept of flag-bridge qubits; flag qubits are used as bridges to cope with qubit connectivity constraints [1]. How this is done can be explained with the stabilizer formalism.

In Figure 2.12a, the first blue CNOT gate entangles qubits s_3 and f and the stabilizer operator X_f propagates through the gate to $S_f = X_s X_f$. Because $S_f X_S = X_f$ each CNOT gates with the data qubits can be performed with any ancilla qubit. This freedom, together with the the fact that the first and last CNOT can also be placed between the blue CNOT's, is exploited by Lao and Almudever to modify the flag syndrome extraction circuits such that they can be mapped onto the Tokyo and Surface-17 processors.

In Figure 2.12b the flag-bridge syndrome extraction circuit measuring one stabilizer is shown. In Figures 2.12c and 2.12d we compare the connectivity graphs of the flag and flag-bridge syndrome extraction circuits. The flag-bridge syndrome extraction circuits for measuring two stabilizers and three stabilizers in parallel are shown in Figures 2.13b and 2.14b. Different combinations of circuits can be used to together measure all stabilizer generators of the Steane code. In Figure 2.15 the mappings on to the Surface-17 transmon device [36] IBM Q Tokyo device [34] are shown. In the mapping figures, the qubits in the coloured blocks are the ancillas in each flag-bridge syndrome extraction circuit. The colours correspond to the colours in the circuits in Appendix A.

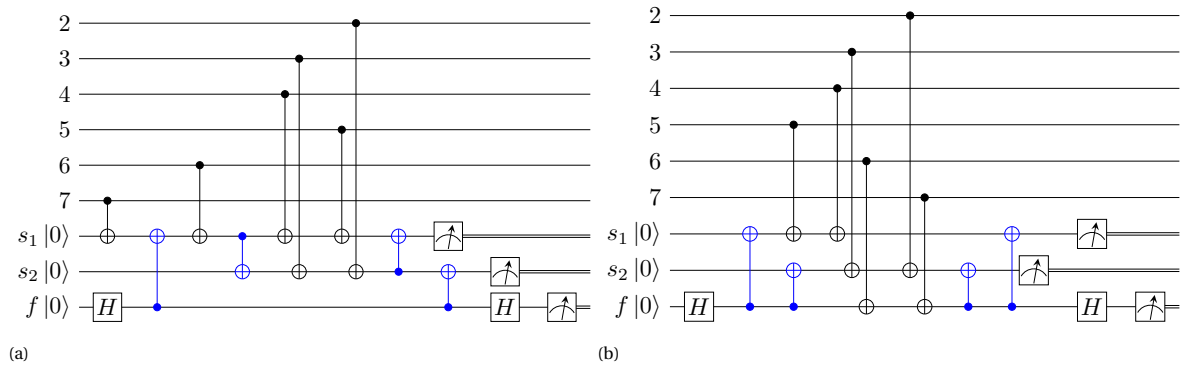


Figure 2.13: (a) Flag syndrome extraction circuit for measuring two stabilizers in parallel and (b) the corresponding flag-bridge syndrome extraction circuit.

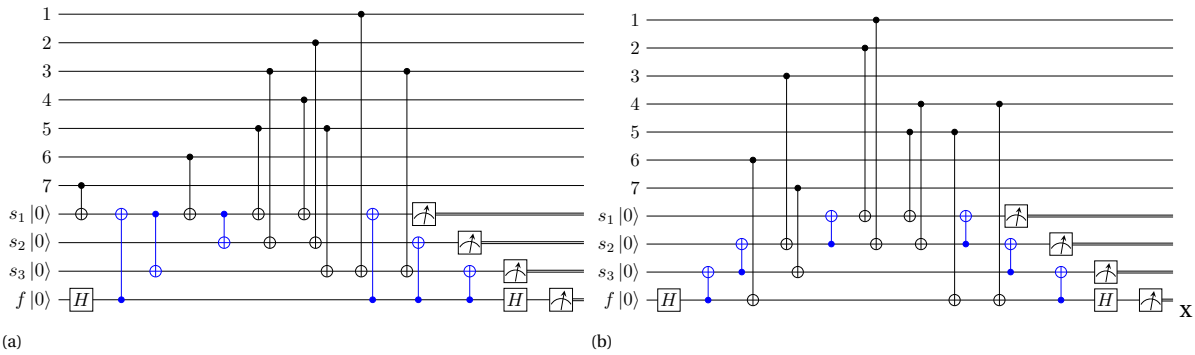


Figure 2.14: (a) Flag syndrome extraction circuit for measuring two stabilizers in parallel and (b) the corresponding flag-bridge syndrome extraction circuit.

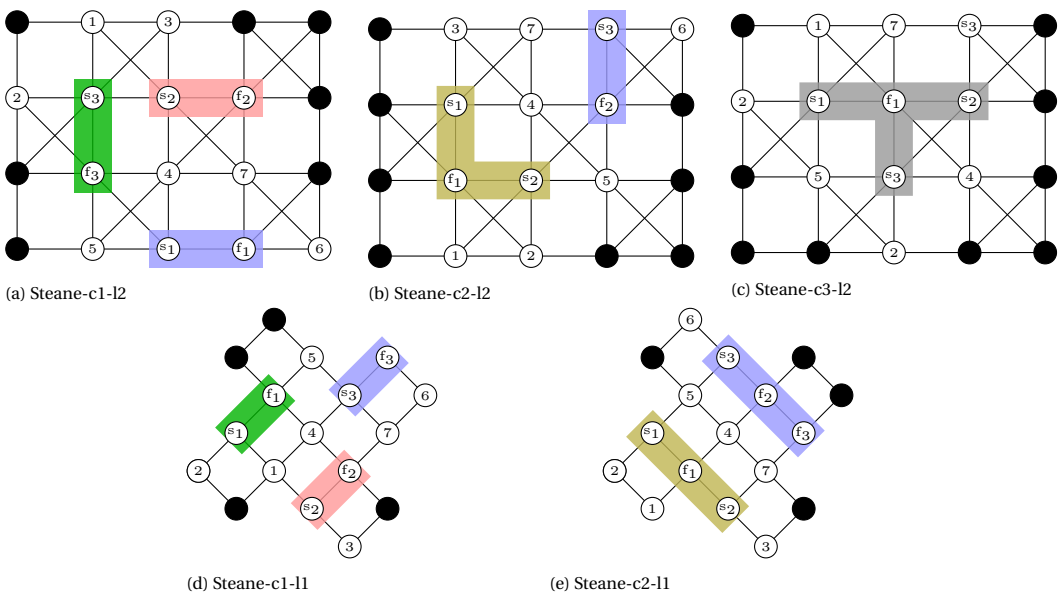


Figure 2.15: Mapping of the Steane code onto the (a,b,c) Tokyo topology and the (d,e) Surface-17 topology using the different flag-bridge syndrome extraction circuits in Figures 2.12b, 2.13b and 2.14a. We refer to the different circuits by the name given under each mapping. The colours correspond with the colours of the CNOT's in the circuits shown in Appendix A.

2.4.4. Fault-tolerant QEC with flag-bridge circuits

The error syndrome extraction circuits from the previous section can be combined to form a round of QEC. Here we first introduce the condition that the error syndrome extraction circuits must meet to be FT. Then we explain the procedure to perform FT QEC.

Fault-tolerant QEC condition

Let $c(g_1,...,g_n)$ denote a flag-bridge circuit that measures a subset of generators of the Steane code, where $g_i \in \{G^j_{X(Z)}\}$, with $i \in \{1,2,..,6\}$, $j \in \{1,2,3\}$. A set of flag bridge circuits that together measure all six stabilizers of the Steane code form a round of QEC and are denoted as $\mathscr{C} = \{c_1(),..,c_n()\}$. A set of flag-bridge circuits $\mathscr{C}$ is FT if all pairs of elements E, E' that can be caused by a single fault satisfy $sf(E) \neq sf(E')$ or $E \sim E'$. Here $sf(E)$ is the measurement result of syndrome and flag qubits caused by E. $E \sim E'$ is defined to be true if two errors are stabilizer equivalent.

Fault-tolerant QEC procedure

A full cycle of FT QEC using the circuits in Figures 2.12, 2.13 and 2.14 is performed as follows:

Algorithm 1 Procedure for Steane code flag-bridge QEC with a round composed of n syndrome extraction circuits: $\mathscr{C} \in \{c_1,...,c_n\}$. The syndrome measurement results of the first and second round are written as $\mathscr{S}^1 = \bigcup_i \mathscr{S}^1_i$ and $\mathscr{S}^2 = \bigcup_i \mathscr{S}^2_i$ and the flag measurement results as $\mathscr{F}^1 = U_i \mathscr{F}^1_i$ and $\mathscr{F}^2 = U_i \mathscr{F}^2_i$

```
i = 1
while i ≤ n do
    run circuit c_i
    if all measurement results are trivial then
        i++ (perform the next syndrome extraction circuit)
    else if the measurement result of a flag is nontrivial then
        Perform all circuits in 𝒞 sequentially and decode using at least ℱ¹, 𝒮² and ℱ²
        break
    else if the measurement result of a syndrome is nontrivial then
        Perform all circuits in 𝒞 sequentially and decode using at least 𝒮² and ℱ²
        break
    end if
end while
```

To show how the algorithm works, we explain how the ZZ error that was previously shown in Figure 2.9 can be corrected. Without flag qubits, Z_3 gives the same syndrome as Z_4Z_5 and is therefore uncorrectable. An example of a set of flag-bridge circuits is given in Figure 2.16. In this example $\mathscr{C} = \{c_1(G^1_Z), c_2(G^2_Z), c_3(G^3_Z), c_4(G^1_X), c_5(G^2_X), c_6(G^3_X)\}$. First, $c_1(G^1_Z)$ and $c_2(G^2_Z)$ are performed and both circuits have trivial measurement results. The next circuit $c_3(G^3_X)$, has a non-trivial flag. Thus, all 6 circuits in $\mathscr{C}$ are run. The resulting non-trivial measurement results are $\mathscr{F}^2_3$ and $\mathscr{S}^2_2$. The input to the decoder can be written as a bit string: $\mathscr{F}^1 = 001$, $\mathscr{S}^2 = 000010$ and $\mathscr{F}^2 = 000000$. The extra information from the flag qubits is used and because these measurement results are not caused by any other fault the ZZ error can now be corrected.

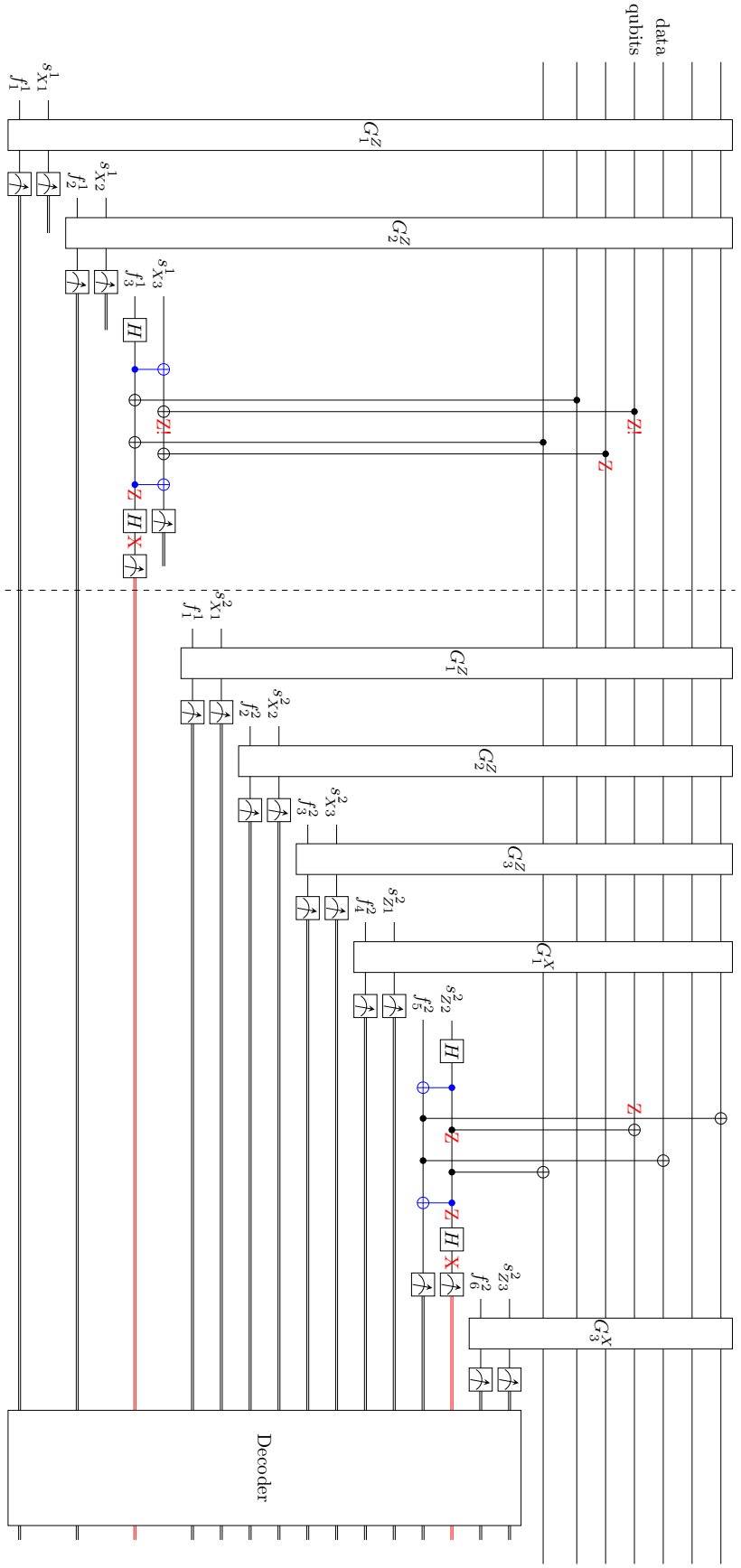


Figure 2.16: Example of a full cycle of flag bridge QEC. During $c_3(G_X^3)$ a ZZ error, shown with exclamation marks, occurs on the second CNOT. This error causes a non-trivial flag, thus following it all 6 circuits in $\mathscr{C}$ are run. The resulting non-trivial measurement results are $\mathscr{F}_3^2$ and $\mathscr{S}_2^2$. The input to the decoder can be written as a bit string: $\mathscr{F}^1 = 001$, $\mathscr{S}^2 = 000010$ and $\mathscr{F}^2 = 000000$.

3

Neural Network Based Decoders

The previous chapter discussed QEC with flag-bridge circuits. This chapter discusses the accompanying decoding process and different types of decoders, with a focus on neural network based decoders. In this chapter, we use syndromes to refer to measurement results of all ancilla qubits (including flags).

3.1. Error decoding process

The decoding process of stabilizer codes consists of processing the measurement outcomes to find corrections. Errors that lead to different logical operations I, X, Z or Y can have the same syndrome. The most likely logical operation for some measurement result is the operation that has the highest probability of occurring. The aim of a decoder is to output corrections that will transform the data qubits back to the most likely logical state.

Simulations are used to design and test decoders. Ideally, these simulations use complex error models that describe experimental reality, such that decoders are optimized for quantum hardware. Density matrix simulations are currently the best tool we have for simulating reality. However, because simulating density matrices is computationally expensive, error models that inject Pauli gates are often used.

The three most common types of error models are the code capacity, phenomenological and circuit level model, shown in figure 3.1. In these models, errors are all possible Pauli operations without phases. There are three single qubit errors, X, Y and Z and 15 possible two qubit errors $IX, IZ, IY, XI, XX, XZXY, ZI, ZX, ZZ, XY, YI, YX, YZ, YY$. The simplest model is the code capacity model. In this model, errors on the data qubits occur before the syndrome extraction circuit with probability p_1. Operations on ancilla qubits are error free and no idling errors occur. The phenomenological model adds measurement errors. With probability p_s an X gate occurs before the measurement. The circuit level model is the most complex of the three and includes idling errors, state preperation and measurement (SPAM) errors and two qubit gate errors. It is important to use the circuit level model for simulating fault-tolerant QEC.

A common metric for measuring the performance of a QEC code at a certain distance is the pseudo-threshold. The pseudo-threshold is defined as the highest physical error rate that the quantum device can operate with in order for error correction to be beneficial for a certain distance. Note that the pseudo-threshold depends on the error model and the accuracy of the decoder.

Designing a decoder with a high performance is in principle not difficult. The challenge is caused by the backlog problem, which limits the decoding time budget [37]. First we will discuss a situation where the back-

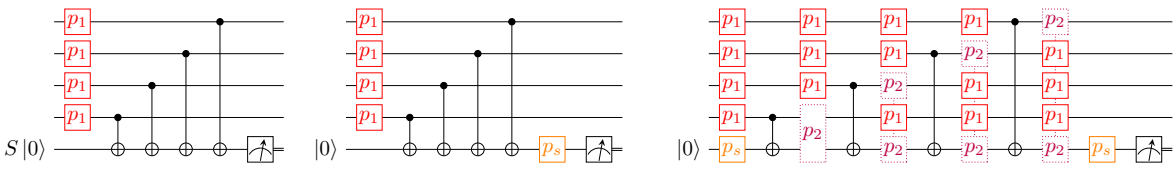


Figure 3.1: Three different types of errors models applied to a weight-4 syndrome extraction circuit; (a) the code capacity model, (b) phenomenological model and (c) the circuit level model. Single qubit errors are shown in red, SPAM errors are shown in orange and two qubit errors are shown in purple.

log problem does not occur, namely when a quantum circuit exists of only Clifford gates. For these quantum circuits, the time the decoder takes does not influence the performance of the circuit. This is motivated by two facts.

Firstly, it is never necessary to physically apply Pauli corrections to data qubits. We simply need to keep track of which eigenspace of the generators of a code we are in. There is no need to stay in the +1 eigenspace of all generators. Secondly because the algorithm consists of only Clifford gates, the classical information of the logical Pauli frame does not need to be available during the execution of the circuit. As explained earlier, the Pauli frame can be efficiently commuted through the circuit. Therefore, the Pauli frame given by the decoder can be applied after the final measurements in the circuit.

A universal gate set needs to contain at least one non-Clifford gate. A popular proposal for universal fault-tolerant quantum computing uses magic states and quantum teleportation to implement the T gate [24], as shown in figure 3.2.

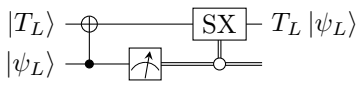


Figure 3.2: Circuit for performing the T gate using a one-bit teleportation.

If a logical error flips the measurement result on the bottom qubit, SX will incorrectly be applied. SX is a Clifford error and not a Pauli error and therefore can not be commuted through the circuit. If it is not corrected it may spread and become a more complicated multi-qubit error. Therefore it must be corrected immediatly after the T gate. If the decoder takes more time to process a round of QEC than it takes to perform a round of QEC, this will cause an exponential slow down during the computation, by the following arguments.

Let r be the ratio between the time it takes to run a round of QEC ($t_{generate}$) and the time it takes the decoder to process a round ($t_{process}$): $r = t_{process}/t_{generate}$. Assume $r > 1$ and we have just performed the T gate circuit and still need to process n_1 rounds. It takes the decoder $t_{process} \cdot n_1$ time to process these rounds, and while this is happening more rounds of QEC are run. The number of new rounds run is

$$n_2 = \frac{t_{process}}{t_{generate}} n_1 = r \cdot n_1 > n_1.$$

At the next T gate we need to have processed n_2. While the decoder is processing these rounds, again more rounds are run:

$$n_3 = \frac{t_{process}}{t_{generate}} n_2 = r \cdot n_2 = r^2 \cdot n_1 > r \cdot n_1 > n_1.$$

A pattern occurs; in order to execute the k^{th} T gate one has to wait $t_{process} \cdot r^k \cdot n_1$ time for the decoder to process syndromes. The conclusion is that the a decoder must have a processing time, $t_{processor}$ that is shorter than $t_{generate}$.

Furthermore the decoder does not only need to be fast, it also needs to be scalable. For the surface code, the number of measurement results that the decoder needs to process scales with $d^3 - d$.

3.1.1. Motivation for neural network decoders

Many different algorithms have been designed to decode QEC codes. In Figure 3.3 the speed and accuracy of different decoding algorithms is shown.

There are three main motivations for using neural network based decoders (NNbD). The first one can be seen in Figure 3.3, namely that the accuracy of neural network based decoders (NNbD) is comparable to the Maximum Likelihood Decoder (MLD) and decoders using the Markov Chain Monte Carlo algorithm and Blossom algorithm, but the execution time is shorter. Additionally, NNbD can be more easily implemented on hardware [1].

The second motivation for using NNbD's is that they can be adaptable to different error models and no information of the error model is needed. The decoder only uses input-output pairs without any knowledge of the QEC code and syndrome extraction circuits.

The last motivation is specific to flag-bridge circuits. Flag-bridge circuits are chosen based on the qubit topology, leading to different error-syndrome patterns and in turn requiring different decoding strategies. It

is difficult to design heuristic decoding algorithms that can be applied to various syndrome extraction circuits [1].

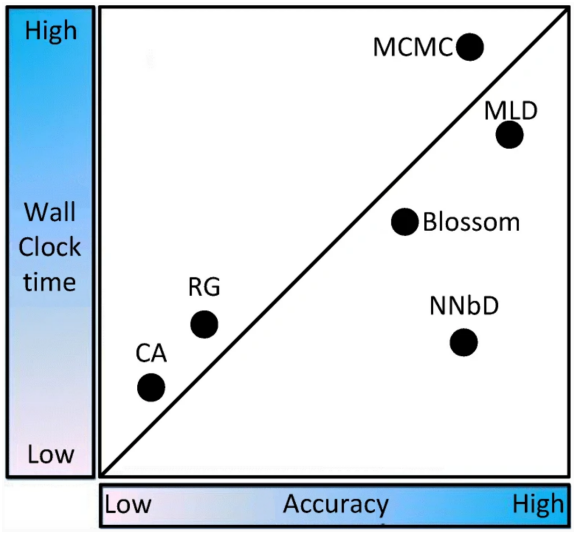


Figure 3.3: Abstract comparison between accuracy and wall clock time of various decoding algorithms. Figure taken from [18].

A remaining challenge for NNbD's is that neural networks have to be trained for every distance and in general machine learning methods become problematic when the input space becomes too large. Therefore NNbD's have a scalability issue.

3.2. Neural networks

The human brain has 86 billion neurons, and every neuron is connected to 7000 other neurons [38]. Neurons respond to impulses by generating an all-or-nothing electric pulse; if a neuron responds at all it responds completely. In accordance with this all-or-nothing principle, Frank Rosenblatt developed the perceptron, the first artificial neuron [39], inspired by earlier work by Warren McCulloch and Walter Pitts [40]. Currently, sigmoid neurons or neurons using the tanh or ReLU function are mostly used [41].

Artificial neurons can be combined to form a neural network. An example of the simplest type of neural network, a feed-forward neural network, is shown in figure 3.4. The universality theorem states that a feed-forward neural network can compute any function: $f(x_1, x_2, ..., x_n) = y_1, ..., y_n$ [41]. The last layer of a network is the output layer, and the layers between the input and output layers are called hidden layers. In a network we use w^l_{jk} to denote the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. b^l_j denotes the bias of the j^{th} neuron in the l^{th} layer. a^l_j is the activation of the j^{th} neuron in the l^{th} layer, given by

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) \tag{3.1}$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer.

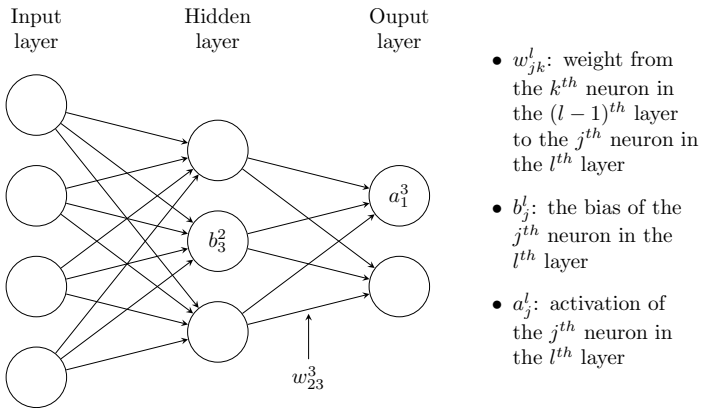


Figure 3.4: Schematic of a feed-forward neural network. The circles indicate neurons, except the circles in the input layer which are units whose activation is simply defined to output $x_1, x_2, ...x_n$

For a neural network to compute a function, it needs to learn weights and biases so that it gives the correct output. To give some intuition how learning works, suppose a change in one of the weights causes a small change in the output of the network. Now, we can tune this weight to get the output closer to the desired output. If we do this for all weights and biases we can figure out how to get the network to compute the function we want. Optimization algorithms can be devised that can automatically tune the weights and biases of a network. These optimization algorithms enable us to use neural networks in a way which is radically different to conventional circuits with logic gates. A neural networks can simply learn to solve problems, for which it would be extremely difficult to directly design a conventional circuit.

3.2.1. Backpropagation algorithm

There are different types of learning processes for neural networks, including: supervised learning, unsupervised learning, reinforcement learning and genetic learning. Here we restrict our discussion to supervised learning. For supervised learning, a dataset containing inputs and target outputs is used. To describe supervised learning we first need to define a metric for the accuracy of the output of a network. This metric is the cost and is calculated with the cost function, which describes how far the outputs a_j^L are from the target outputs y of an input x. An example of a frequently used cost function is the Mean-Squared-Error function (MSE) which describes the squared distance between the output and the target :

$$C_x = \frac{1}{2}\sum_n (a_j^L - y_n)^2$$

.

To calculate how changes in weights and biases influence the cost, an intermediate quantity $\delta_j^l = \frac{\partial C}{\partial t_z^L}$ is used. δ_j' is defined as the error in the j^{th} neuron in the l^{th} layer. Backpropagation will give us a procedure to compute the error δ_j^l, and then will relate δ_j^l to $\partial C_x/\partial w$ and $\partial C_x/\partial b$. To simplify the notation, equation 3.1 can be rewritten in vector form:

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

Here, a_l is a vector where the entries are the activation of each neuron in layer l, w_l is a matrix with entries w_{jk}^l and b_l is a vector with the bias of the neurons in layer l.

The backpropagation algorithm consits of five steps:

1. Input x : Set the corresponding activation a^l for the input layer.
2. Feedforward: For each layer in the network $l = 2,3,\ldots,L$, compute sequentially $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma\left(z^l\right)$.
3. Output error δ^L: Compute the vector $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$. Here $\odot$ is the Hadamard product and σ' is the first derivative of the transfer function: $\frac{d}{dz^l}\left(\sigma(z^l)\right)$.
4. Backpropagate the error: For each $l = 2,3,\ldots,L$ compute $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$

5. Output: The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

In the next section it is shown how the gradients of the cost function are used to train a neural network.

3.2.2. Optimization algorithms

Optimization algorithms are used to automatically tune the weights and biases of a network of artificial neurons. A simple optimization algorithm is gradient descent. Using the output of the backpropagation algorithm it updates weights and biases as follows:

$$\begin{aligned} w_k \rightarrow w'_k &= w_k - \eta \frac{\partial C}{\partial w_k} \\ b_l \rightarrow b'_l &= b_l - \eta \frac{\partial C}{\partial b_l}. \end{aligned} \tag{3.2}$$

where η is a positive parameter known as the learning rate. The aim of gradient descent is to find the global minimum of C_x by repeatedly using the backpropagation algorithm and applying the update rules.

In most settings, for a neural network to be useful, it needs to give the correct output for many inputs. To do this the difference between the output and target needs to be minimized for every training sample:

$$C = \frac{1}{n} \sum_x C_x = \frac{1}{2n} \sum_x \left\| y(x) - a^L(x) \right\|^2.$$

From now on, we will use w to denote weights and biases. To decrease the cost over all samples the partial derivatives with respect to all weights need to be calculated:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x = \frac{1}{n} \sum_x \left(\frac{\partial C_x}{\partial w_1}, ..., \frac{\partial C_x}{\partial w_n} \right)$$

When the number of training inputs is very large calculating the cost can take a long time, and learning thus occurs slowly. An idea called stochastic gradient descent (SGD can be used to speed up learning. The idea is to estimate the gradient ∇C by computing ∇C for a small sample of randomly chosen training inputs, labeled $X_1, X_2, ..., X_m$, that form a so-called batch. The update rule for SGD is

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

where m is the batch size.

The major challenge in using SGD is choosing the learning rate. The learning rate needs to be large enough such that the weights and biases do not get stuck in local minimum, but small enough to find a global minimum. In complex cases, the learning rate needs to be changed during the optimization, making the choice of learning rates even more complex as it has to be done at multiple timesteps.

To solve this, more advanced optimization algorithms incorporate not just information about the gradient, but also information about how the gradient is changing. Incorporating how the gradient is changing could be done by calculating second derivatives $\frac{\partial^2 C}{\partial w_j \partial w_k}$ to form the Hessian matrix, but the computational cost of doing this is too high for a network with many weights. Instead momentum-based gradient descent introduces a notion of velocity. The gradient acts to change the velocity, not directly the weights, comparable to how a physical force changes the velocity and only indirectly affects position. Secondly a hyper parameter μ, called momentum is introduced which tends to gradually reduce the velocity. The update rules for the momentum algorithm are

$$\begin{aligned} v \rightarrow v' &= \mu v - \eta \nabla C \\ w \rightarrow w' &= w + v' \end{aligned}$$

Where v is an array of velocity variables for each weight $v = v_1, v_2, \dots v_n$. The advantage of using the momentum algorithm is that the initial choice of learning rate is less important and the learning rate is automatically updated to speed up optimization. A disadvantage of the momentum algorithm is that it introduces an additional hyperparameter μ.

In Figure 3.5 a visual representation of how the momentum algorithm outperforms SGD is given. In this figure, the cost is more sensitive to changes in one weight than the other. Therefore it makes sense to use a separate learning rate for each parameter.

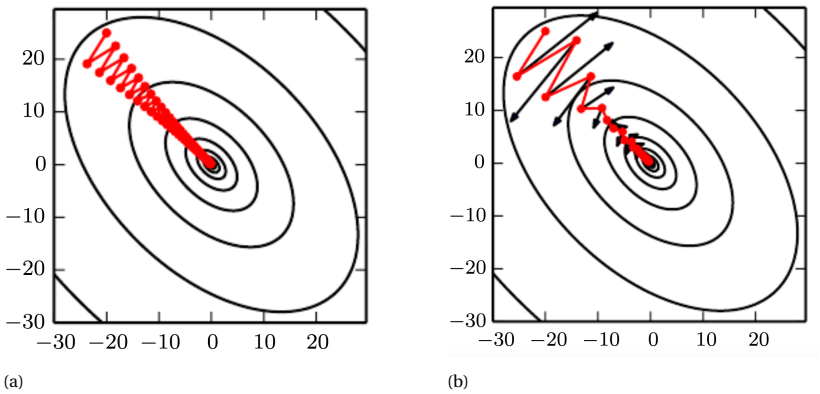


Figure 3.5: Illustration of the advantage of using the momentum algorithm compared to SGD. The contour lines depict the loss function for two weights. The red path indicates the path followed by the (a) stocastich gradient descent algorithm and the (b) momentum algorithm. The red dots indicate the value of the weights at each timestep and in (b) the black arrows indicate the velocity at each timestep. Both figures are taken from [43]

The Adam algorithm uses a separate learning rate for each weight and automatically updates the learning rates. The name Adam refers to "adaptive moments" and the algorithm estimates the first- and second-order moment of the gradient at each time step. The momentum hyperparameter μ we have previously seen is incorporated in the first-order moment estimate m_t. [42]. The Adam algorithm is widly used due to its high performance and because the initial value of the hyper-parameters typically require little tuning. For more information and the update rules of the Adam algorithm we refer the reader to [42].

3.3. Decoding with Neural Networks

Neural networks can be used for the task of error decoding in two ways; they can be trained to provide corrections on the physical level or they can be trained to provide corrections that restore the logical state [16], [17], [44]. The former is referred to as a low level decoder and the latter a high level decoder. The output of a low level decoder is a Pauli operation for each data qubit and the output of a high level decoder is a single Pauli operation.

To use a high level decoder a so-called simple decoder needs to be used that provides a pure error. A pure error is an operation that brings the state of data qubits back to the logical subspace. Any error can be expressed as a product of three operators,

$$E = S \cdot C \cdot L$$

a pure error C, a stabilizer S and a logical operation L.

For a high level decoder for the surface code, two neural networks need to be used, as shown in Figure 3.6. The task of the first neural network is to detect measurement errors. These measurement errors are filtered out of the input of the simple decoder. The task of the second neural network is to output L.

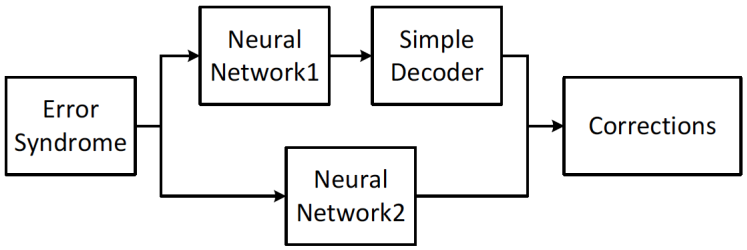


Figure 3.6: Design for high level decoder. Note that for a code capacity model network 1 is not used. Figure taken from [45]

In general, machine learning techniques and neural networks excel at classification problems. By using a simple decoder, the task of a neural network is reduced to a classification problem, as it needs to identify L to be I, X, Z or Y. The training data for a high level decoder consists of probability distributions over the four logical operations. A low level decoder does not perform a classification problem because corrections

on the physical level are not separate categories. Therefore the labels of training data for a low level decoder are not probility distributions. When using a low level decoder it is difficult to prevent the neural network from just memorizing the dataset, instead of learning the error model. Just memorizing the dataset will lead to problems when a neural network is used for a higher distance QEC code, as it is not possible for a dataset to contain all possible measurement results.

4

Results

In the first part of this chapter we focus on increasing the pseudo-threshold by exploring different neural network decoding approaches. The different circuits used here and their names are shown in Figure 2.15. First we introduce the look-up table decoder and neural network decoder used in [1].Then we find an approximate upper bound to the performance of neural network decoders and discuss our findings. In the last part we, present the simulation results of high level decoder after the neural network has been trained.

In the second part of this chapter, we focus on increasing the pseudo-threshold by decreasing the number of timesteps the flag-bridge syndrome extraction circuits take. We describe the steps taken to optimize the circuits and the procedure to show that our optimized circuits are FT. In the last section we present the pseudo-thresholds of the optimized circuits for the circuit level noise models and draw conclusions.

4.1. Implementation of the look-up-table and neural network decoder

In [1] a look-up table (LUT) and a neural network based decoder (NNbD) for flag-bridge circuits are presented. The input for the LUT and NNbD is slightly different. There are two cases in which decoding is required. The first case is if nontrivial syndromes (no flags) are measured in the first round. For this case the decoders will decode using only the measurement results in the second round. The second case is if nontrivial flags are measured. For this case the decoders will decode using these flags and the measurement results in the second round. In both cases for the measurement information in the second round, the LUT decoder only considers the results of syndrome qubits, while the NNbD additionally takes the flags of the second round into account. The look-up table is designed such that it can correct all errors caused by a single fault, while the NNbD can correct some errors caused by two faults.

The NNbD is trained "on the fly" meaning that the batches of data are generated during the training procedure, instead of creating a dataset before training. Each sample in the batch is found by running one cycle of QEC and consists of measurement results whose corresponding labels are errors on data qubits. During the training procedure the measurement results in the batch are propagated through the network. The NNbD's consist of three hidden layers. In the hidden layers the tanh or ReLU activation functions are used and for the output layer the sigmoid activation function is used. The difference between the output of the network and labels is calculated using the cross entropy cost function. At the end of each batch the parameters of the neural network are adjusted using either the Adam or Nadam optimizer. The learning rate used is 0.002 and the batch size is 50.

In [1] the NNbD outperforms the LUT decoder and therefore the NNbD is used in numerical simulations to compare the different circuits. The circuits are simulated under four different noise models; the circuit level noise model and three modifications with different idling error rates: $p_I = 0$, $p_I = 0.01p$, $p_I = 0.1p$. Figure 4.1 shows all errors that can occur during a flag-bridge syndrome extraction circuit of one stabilizer.

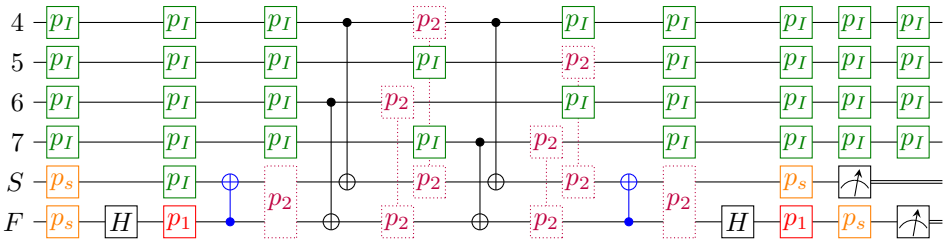


Figure 4.1: Error model used in [1]. All gates and idling locations are followed by a Pauli error with some probability. p_I and p_1 are the probabilities that an idling locations and single qubit gates are followed by an X, Y or Z error. p_s is the probability that an X occurs after qubit preparation or measurement. p_2 is the probability that a two qubit gate is followed by a two qubit Pauli error (excluding II).

4.2. Simulation setup

In this section we introduce our simulation setup to compare decoders. The metric we use to compare different decoders is the pseudo-threshold, previously introduced in Section 3.1. To find the pseudo-threshold, we use the stabilizer formalism to simulate the circuits. In our simulation the Pauli and Clifford groups are represented in binary symplectic form, such that stabilizer generators are represented as binary vectors and matrix operations as binary operations. Here we will give a short introduction to the binary symplectic representation, for more details we refer the reader to Section 2 of [46]. The set of single qubit phase free Pauli operators, denoted as $[\mathscr{P}^1]$, are mapped to $(\mathbb{Z}_2)^2$ as: $I \rightarrow 00, X \rightarrow 01, Y \rightarrow 11, Z \rightarrow 10$. This mapping is especially useful due to two reasons; it induces an isomorphism between $(\mathbb{Z}_2)^2$ and $[\mathscr{P}^1]$ because addition of vectors in $(\mathbb{Z}_2)^2$ is equal to multiplication of Pauli operators up to a global phase and the symplectic product can be used to calculate if two Pauli operators commute.

We use the Python package sparse_pauli as it conveniently implements the binary operations we need. We use Pythons multiprocessing package for the execution of multiple concurrent processes and schedule jobs using 20 to 50 CPU's using the Slurm resource manager. For the design and training of neural networks we use TensorFlow [47].

To compare the performance of different decoders we first simulate the approximate upper bound to the performance using a complete dataset. In the next section we explain in detail how we do this.

4.3. Neural network decoders: determining the best performance

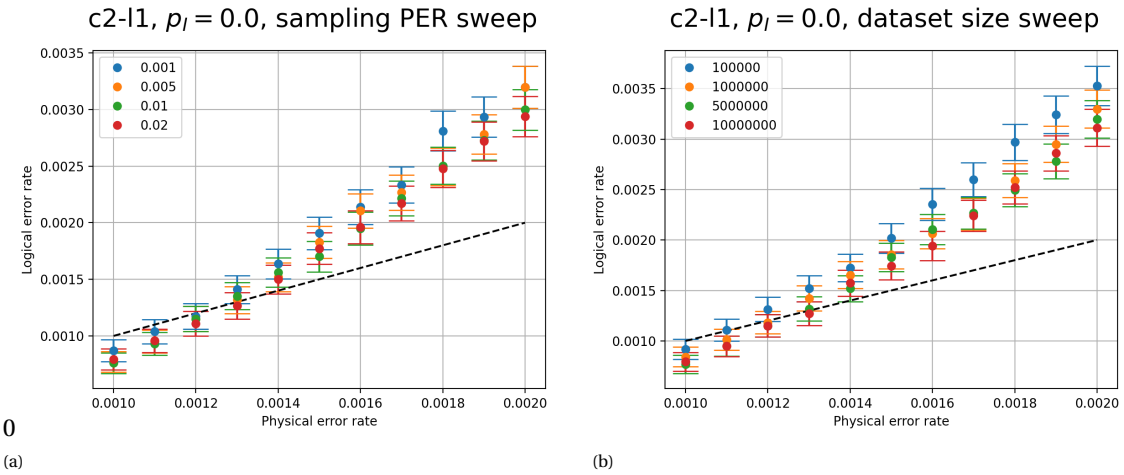
To calculate the best possible performance of a low level neural network decoder we start by creating a dataset containing measurement results and corresponding corrections to make. More precisely, the dataset consists of pairs of binary strings of measurement results and a binary string representing the most frequently occurring error on data qubits (in binary symplectic representation). We use all measurement results (flags and syndromes of both rounds), to have as much information as possible to improve the performance.

After generating the dataset we use the dataset itself as a decoder, by searching through it to find which corrections to make for some measurement result. Ideally, this dataset would be very large and the physical error rate (PER) we sample at would be equal to the pseudo-threshold. We perform two experiments with circuit c2-l1 with $p_I = 0.0$ to find out what the influence of these two variables are on decoder performance.

In Figure 4.2a we compare the performance of datasets created by running the QEC cycle 10^7 times using four different sampling PERs. For each point in the numerics, 10^6 full QEC cycles have been run. In the figure it can be seen that the dataset that was created by sampling with PER 0.005 has the lowest pseudo-threshold. Previous work from Varsamopoulos et al. [17] shows that at higher distance QEC codes the PER used to sample data does matter significantly, but here the PER we sample at does not make a difference larger than the confidence interval of 99.9%.

In Figure 4.2b we compare the performance of datasets sampled at PER 0.005 with different dataset sizes. The difference is again not larger than the confidence interval and we argue that a dataset created by running the circuit 10^7 times is large enough because the performance is nearly identical to a dataset created by running the circuit 5×10^6 times. In the rest of this chapter all datasets are created using 10^7 samples and for the error models with $p_I = 0, p_I = 0.01, p_I = 0.1$ we sample at PER 0.0005 and for $p_I = 1$ we sample at PER 0.001.

Next to the low level decoder, we have additionally designed a high level decoder. For the high level decoder the LUT decoder from [1] is used as a simple decoder. Each sample is created by running the circuit, applying the correction found in the LUT and then saving the resulting logical operation. The overall sampling procedure is similar to the one of the low level decoder. The resulting dataset consists of pairs of binary



0

(a) (b)

Figure 4.2: (a) Numerical simulation of circuit c2-l1 with $p_1 = 0$ using four different datasets created by sampling 10^7 times at PER's between 0.001 and 0.02. (b) Numerical simulation of circuit c2-l1 with $p_1 = 0$ using four different datasets created by running the circuit between 10^5 and 10^7 times with PER 0.005.

strings of measurement results and a probability distribution of logical operations.

4.3.1. High level and low level decoder results

We simulate the five flag-bridge error syndrome extraction circuits, shown in Figure 2.15, for the same range of PERs and idling error rates as in [1]. As discussed in the previous section, the datasets are used as decoders to find approximate upper bounds to the high level and low level neural network based decoders.

Table 4.1 shows a summary of the pseudo-thresholds found for $p_I = 0, p_I = 0.001p$ and $p_I = 0.1p$, where p_I is the probability of a Pauli error occurring after an idling location and p is the probability of all other errors occurring. The plots that show the pseudo-thresholds for c2-l1, c1-l2, c2-l1 and c2-l2 are in Appendix B and for c3-l2 in Figures 4.4 and 4.5. For $p_I = 0$ and $p_I = 0.01p$ the pseudo-threshold found for the low level decoder are on average 12% better than the pseudo-thresholds found for the high level decoder. The reason the high level decoder does not achieve the same level of performance is that the LUT does not always provide a pure error. As explained in Section 3.3, for the surface code a neural network takes out measurement errors before inputting the measurement results to the simple decoder. For the surface code, three rounds of measurement results are performed, and therefore measurement errors can be found. For the FT protocol for flag-bridge codes it is not possible to detect measurement errors. The reason is that there are only two rounds of measurement results and situations occur where the measurement results should be different. Note that using only a neural network outputting logical operations as a decoder would not be FT.

For $p_I = 0.1$ the pseudo-thresholds found for c2-l1, c2-l2 are the same for the high level decoder and low level decoder. For c3-l2 the pseudo-threshold of the low level decoder is 19% better. In Figure 4.3 the results of c1-l1, c1-l2, c2-l1 and c2-l2 are shown for the case $p_I = 1p$. For $p_I = 1p$ and circuits c1-l1, c2-l1 and c3-l2 the logical error rate of the high level decoder is 12%, 13% and 8% lower than that of the low level decoder. For $p_I = 1p$ and circuits c1-l2, and c2-l2 the low level decoder outperforms the high level decoder, but the difference is smaller than the confidence interval.

The reason the high level decoder is performing better for higher idling error rates than the low level decoder, while this was not the case for low idling error rates is because the importance of measurement errors has decreased. As can be seen in Figure 4.1, idling errors play a bigger role than measurement (SPAM) errors. In general, without measurement errors we expect the high level decoder to outperform the low level decoder, as is the case for the surface code [17, 18, 45].

For all idling error rates circuit c3-l2 has the highest pseudo-thresholds, because it uses the least amount of operations and the least amount of timesteps. In the next section we focus on training a high level neural network decoder for circuit c3-l2.

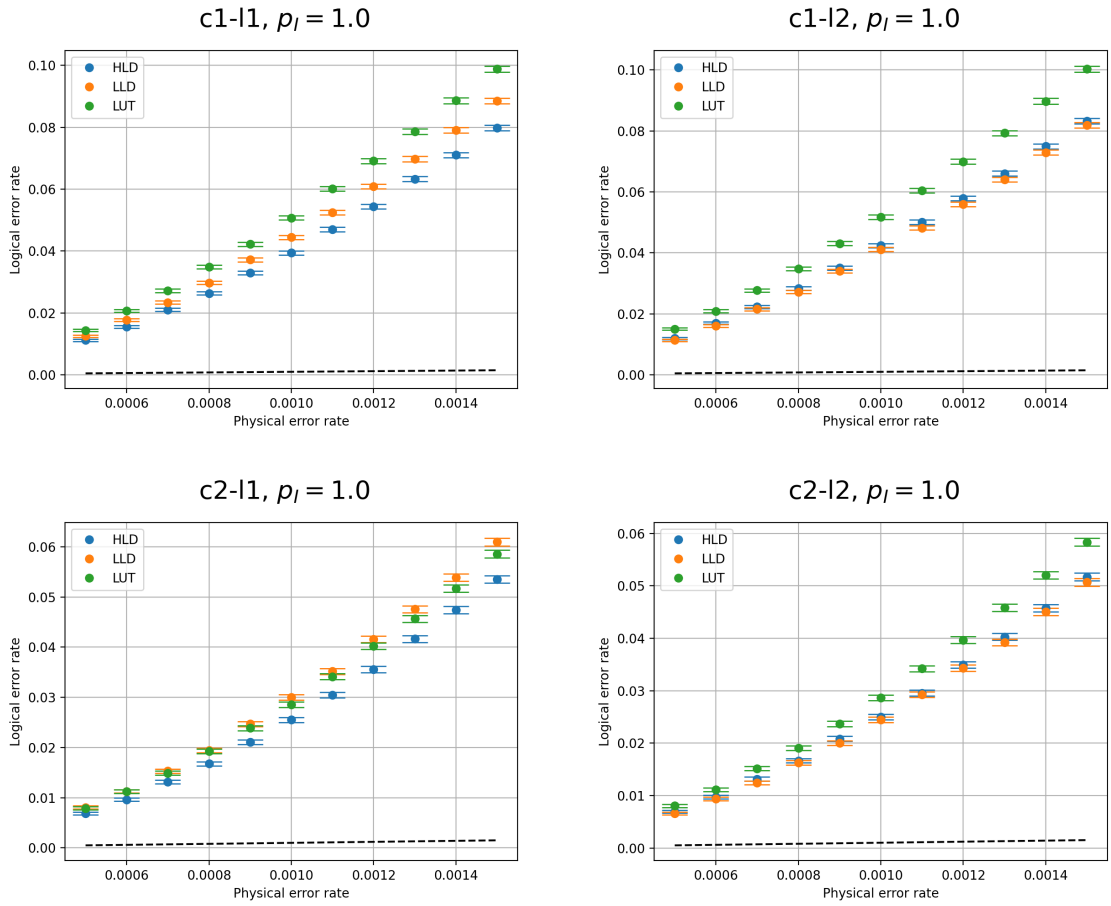


Figure 4.3: Numerical simulation of three different decoders on four different flag-bridge circuits using the circuit level noise model ($p_1 = p_2 = p_M = p_I$).

		pI=0	pI=0.01	pI=0.1
c1-L1	LLD	0.0011	0.0010	
	HLD	0.0010		
c2-L1	LLD	0.0013	0.0011	0.0005
	HLD	0.0011	0.0010	0.0005
c1-L2	LLD	0.0010	0.0010	
	HLD			
c2-L2	LLD	0.0015	0.0013	0.0006
	HLD	0.0012	0.0012	0.0006
c3-L2	LLD			0.0011
	HLD	0.0019	0.0018	0.0009

Table 4.1: Comparison of the pseudo-threshold found for the high level decoder and low level decoder. In the green cells the pseudo-threshold was not found because the physical error rate remained lower than the logical error rate for the range of physical error rate tested. In the red cells the pseudo-threshold was not found because the physical error rate remained higher than the logical error rate for the range of physical error rate tested.

4.4. Neural network decoder results

Compared to previous work we make two main changes in the implementation of the NNbD. Firstly, instead of generating batches of data during training we train using a dataset. The main advantage of creating a dataset before training is that by simulating the circuit many times, we can select the most likely error for a certain measurement result, and thus only train the neural network with the most likely errors. If data is

generated while training, the neural network will need to learn which error occurs most often for a certain measurement result, making the learning process more complex. A disadvantage of using a large dataset is that it consumes a lot of memory. In a hardware implementation, addressing a large dataset can take a considerable amount of time, due to the scheduling of reading a lot of memory addresses [48]. In this thesis we deal with distance 3 codes and the largest dataset only contains 7829 different samples. Therefore the advantages of using a dataset outweigh the disadvantages. For a small distance-3 QEC code it is possible to create a dataset that contains all syndromes that have a reasonable probability of occurring.

The second main change is that we use a the neural networks as part of a high level decoder, instead of using it is a low level decoder. The advantages of high level decoders have been discussed in Section 3.3 . Additionaly, in the previous section we have shown that for our setup the datasets generated for a high level decoder outperforms the low level decoder for the circuit level noise model.

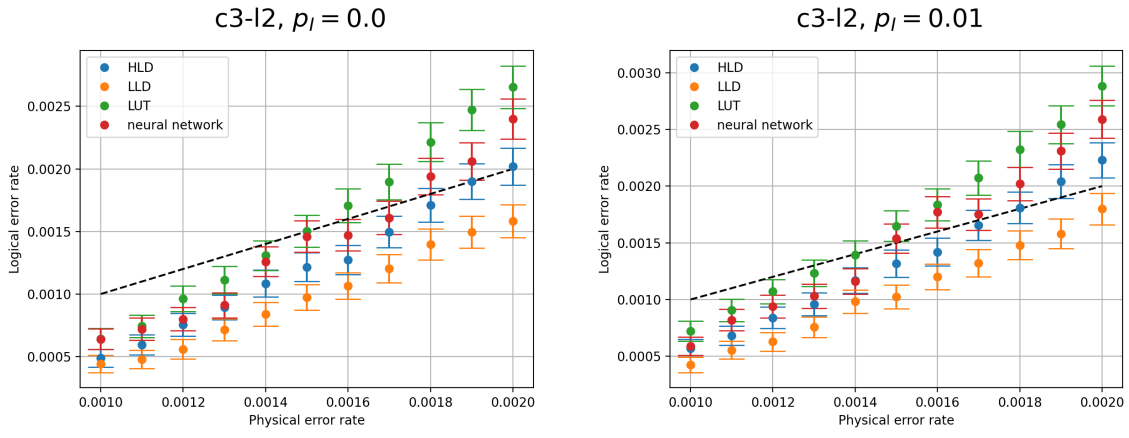
Due to these changes our training setup is different. From our dataset we only use the samples that have occurred at least 20 times, as we do not want to provide our neural network with incorrect labels during the training process. We use 80% of the samples for training and 20% for validation. At the start of each epoch the training samples are shuffled and split into batches of 100 samples. We continue training until the accuracy on the validation data does not increase over 20 epochs. Typically our training takes between 500 and 1000 epochs. After training the networks, we analyze the performance by finding it's pseudo-threshold. We start by running the circuit at a low PER and find the LER. Then the PER is increased until it is equal to the LER.

4.4.1. Results

For each idling error probability a separate neural network is trained to decode errors in the circuit c3-l2. All neural networks consist of three hidden layers, each with 64 nodes. The hyperbolic tangent function is used as the hidden activation function, the sigmoid function is used as the output activation function and mean squared error is used as the cost function. The Glorot normal initializer is used for the initial values of the nodes in the network. The optimizer used is Adam and the intial learning rate is 0.002.

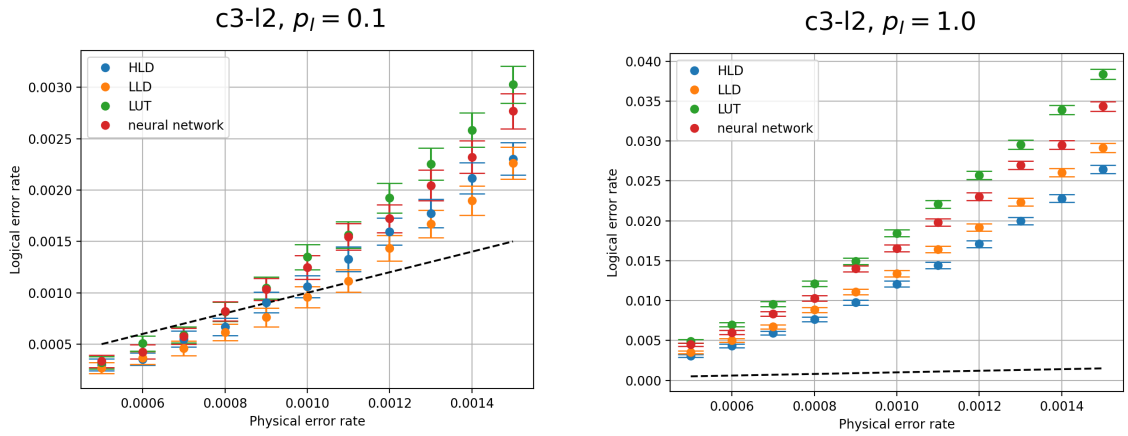
In Figures 4.4 and 4.5 the results of trained neural network decoders for the circuit c3-l2 is compared to the datasets, previous work and the LUT decoder. The neural network outperforms the LUT for all four error models, proving that the training process is successful. On average the logical error rate for the neural network decoder is 10% lower than for the LUT decoder. The biggest difference is found for circuit level noise model, where the logical error rate of the neural network decoder is 14% lower on average.

Because the performance of the neural network is not close to the upper bound, we foresee a possible improvement if the neural network is further optimized. We realize that the optimal size and sampling PER for generating a dataset that provides a good training data may not be the optimal size and sampling PER for a dataset that is itself used as a decoder. In general there is not a simple way to figure out if the sampling rate or dataset size is optimal, due to the many variables involved. For example, for a bigger training dataset, a neural network may need more nodes in each layer, more layers or you may need to train for more epochs to properly learn the dataset.



(a) (b)

Figure 4.4: Numerical simulation of circuit c3-l2 comparing the high level decoder, low level decoder, LUT decoder and neural network decoder. In the two plots the idling error probability 0 and $0.01p$ and all other errors occuring with equal probability ($p_1 = p_2 = p_M$).



(a)

Figure 4.5: Numerical simulation of circuit c3-l2 comparing the high level decoder, low level decoder, LUT decoder and neural network decoder. In the two plots the idling error probability is $0.1p$ and $1.0p$ and all other errors occuring with equal probability ($p_1 = p_2 = p_M$).

4.5. Optimized flag-bridge circuits

In the previous section we explored different decoder strategies. In this section we explore new FT flag-bridge error syndrome measurement circuits. We start by explaining the steps taken in designing the new circuits, followed by an explanation of how we show that the circuit are FT.

4.5.1. Steps taken to optimize the circuits

We have taken three main steps to reduce the number of timesteps of the circuits. The flag-bridge based Steane code circuits from [1] combine multiple syndrome extraction circuits that each measure a subset of stabilizer generators of the Steane code. As shown in Figure 4.6a, the different syndrome extraction circuits are performed sequentially. The first step we take to reduce the number of timesteps is that we interleave the different syndrome extraction circuits to run them in parallel. From here on we refer to the circuits in [1] as sequential circuits.

The second step taken is to prepare qubits as late as possible and measure qubits as soon as possible. The last step is that the Z syndrome extraction circuits start before ending the X syndrome extraction circuits. At the start and end of each syndrome extraction circuit Hadamard gates are applied on either the syndrome or flag qubits. To minimize the number of timesteps, we apply the Hadamard gates to different qubits for the X and Z extraction circuits. In the FT QEC procedure from [1] explained in Section 2.4.4, if during the first round an X or Z ancilla qubit fires, than the second round would start with measuring all X stabilizers followed by Z stabilizers. In our new protocol, if an X (Z) ancilla fires in the first round the second round starts with the Z (X) syndrome extraction circuits.

Although the optimized circuits perform the same unitary operations as the sequential circuits, it is not trivial whether they remain FT. The brute-force procedure for checking if the circuits are FT is similar to the procedure for creating the LUT decoder. One by one a different two-qubit Pauli error is inserted after a CNOT in the circuit of one QEC round. The resulting error on data qubits and the measurement results are saved. After all errors have been inserted, we check that the set of all resulting errors and measurement results satisfy the fault-tolerance criteria. First, we check if two errors with the same measurement results are equal. If this is not the case we check if the errors are stabilizer equivalent. This is checked by multiplying the errors and checking if the resulting operation commutes with the logical operations of the Steane code.

4.6. Optimized flag-bridge circuits results

In this section we first show the circuit statistics of the optimized circuits and compare them to the sequential circuits. Afterwards we discuss the numerical simulations performed to test the optimized circuits.

On the Surface-17 layout we reduce the number of timesteps of circuit c1-l1 from 50 to 17 steps (66% reduction) and circuit c2-l1 from 40 to 19 steps (52.5% reduction). The optimized version of c1-l1, which we call s17-222 is shown in Figure 4.6b and its mapping is shown in Figure 2.15d. The circuit and the mapping of s17-33, the optimized version of c2-l1, is shown in Appendix A.

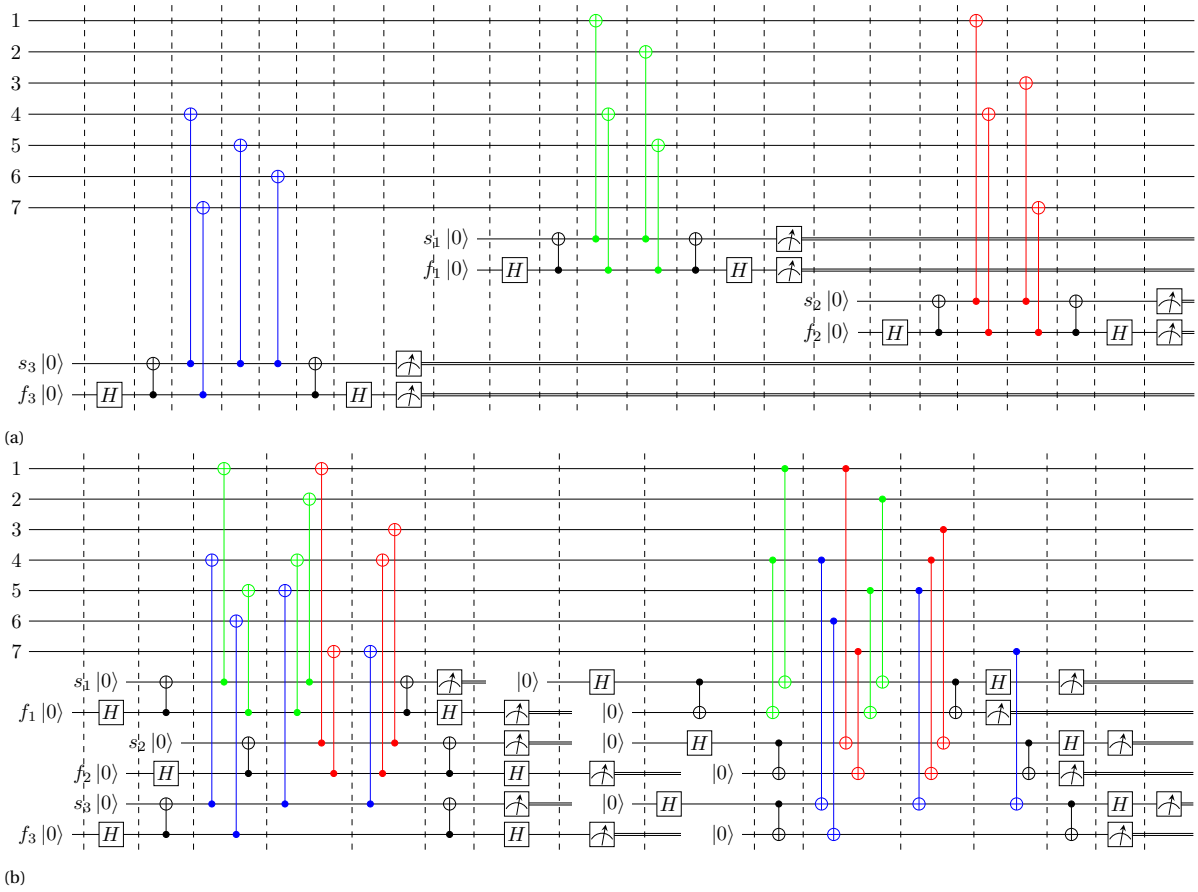


Figure 4.6: (a) The previous circuit c1-l1 with 3 flags and 3 syndrome qubits, (b) the optimized version s17-222.

On the Tokyo layout we reduce the number of timesteps of circuit c1-l2 from 48 to 17 steps (64.6% reduction), of circuit c2-l2 from 36 to 19 steps (47.2% reduction) and circuit c3-l2 from 26 to 25 (3.8% reduction). The optimized versions of the circuits and are shown in Appendix A. We name the circuits on the Tokyo processor according to the number of qubits they use; Tokyo-13, Tokyo-12 and Tokyo-11. We name the circuits on the Surface-17 processor; Surface-17-222 and Surface-17-33. Surface-17-222 consists of three Z-syndrome and three X-syndrome extraction circuits, all using two ancillas. Surface-17-33 consists of two Z-syndrome and two X-syndrome extraction circuits, all using three ancillas.

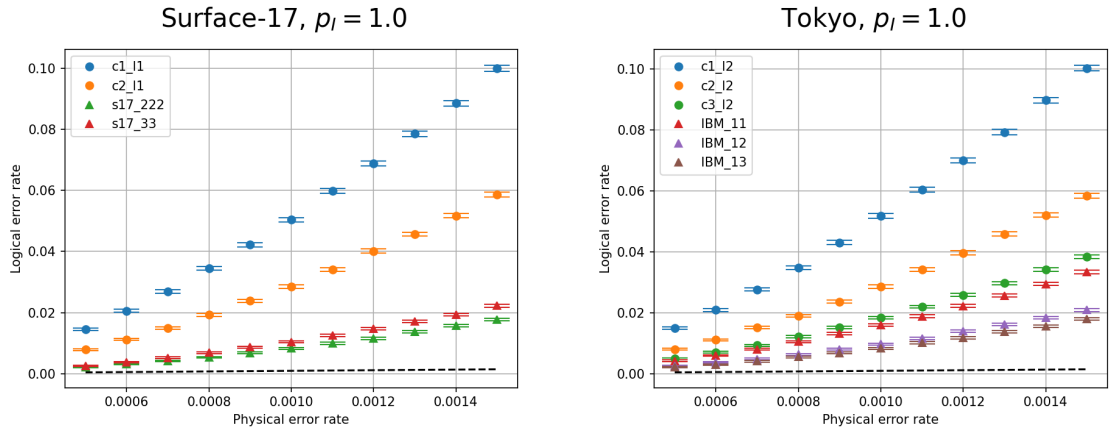
To compare our circuits with previously designed circuits we simulate all of them using a LUT decoder at four different idling error rates. Additionally, for the circuit level noise model, we generate high level decoder datasets for all optimized circuits. Table 4.2 shows the pseudo-thresholds found for the sequential circuits, the optimized circuits and the distance-3 rotated surface code. The pseudo-thresholds of the surface code for $p_I = 0, p_I = 0.01$ and $p_I = 0.1$ are taken from [1] and the pseudo-threshold for $p_I = 1$ is taken from [45].

As shown in the table, for low idling error rates $p_I = 0$ and $p_I = 0.01$ the optimized circuit have nearly identical pseudo-thresholds compared to the sequential circuits. It is interesting to note that for these low idling error rates the circuits Steane-c3-l2 and Tokyo-11 have the same pseudo-threshold as the surface code. For higher idling error rates $p_I = 0.1$ and $p_I = 1.0$ optimized circuits have a much better performance than the sequential circuits. The difference in performance between the sequential circuits and optimized circuits is the biggest for $p_I = 1.0$, as can be seen in Figure 4.7. On the Surface-17 topology, the logical error rate at PER 0.0015 of c1-l1 is reduced by 81% and of c2-l1 by 64%. On the Tokyo topology, the logical error rate at PER 0.0015 of c1-l2 is reduced by 81%, of c2-l2 by 65% and of c3-l2 by 5%.

The sequential circuits that consist of less independent syndrome extraction circuits have a lower logical error rate. For the optimized circuits, the opposite holds; the circuits S17-222 and Tokyo-13, which combine six syndrome extraction circuits, perform the best. These two circuits use more ancilla qubits and therefore there are more options to perform CNOT's in parallel. From the sequential circuits, the conclusion was drawn that the Tokyo chip can outperform the Surface-17 chip when running the Steane code [1]. With the

	# Qubits	# Operations	# Timesteps	$p_I = 0$	$p_I = 0.01$	$p_I = 0.1$	$p_I = 1.0$
Steane-c1-L1	13	72	50	0.007	0.007		
Surface-17-222	13	72	17	0.008	0.008	0.006	0.00017
Steane-c2-L1	13	72	40	0.009	0.009	0.005	
Surface-17-33	13	72	19	0.008	0.008	0.006	0.00014
SC d=3	17	48	8	0.015	0.014	0.01	0.000444
Steane-c1-L2	13	72	48	0.007	0.006		
Tokyo-13	13	75	17	0.015	0.008	0.006	0.00014
Steane-c2-L2	12	64	36	0.009	0.007		
Tokyo-12	12	62	19	0.01	0.009	0.008	0.00013
Steane-c3-L2	11	54	26	0.015	0.014	0.008	
Tokyo-11	11	54	25	0.015	0.014	0.008	0.00010

Table 4.2: Comparison of the quantum error correction circuits when different mappings are applied. In the four rightmost columns the pseudo-thresholds at different idling error rates are shown. If no pseudo-threshold is found the cell is red. For the idling error rates $p_I = 0, p_I = 0.01, p_I = 0.1$ a LUT decoder is used and for $pI = 1$ a dataset for a high level decoder is used. The results of the surface code for $p_I = 0, p_I = 0.01$ and $p_I = 0.1$ are taken from [1] and the result for $p_I = 1$ is taken from [45].



(a) (b)

Figure 4.7: Numerical simulation with a LUT of the sequential and optimized circuits with the circuit level noise model on the (a) Surface-17 and (b) Tokyo topologies.

optimized circuits we show that the opposite is true. The circuit s17-222 has the same amount of time steps as Tokyo-13 meaning that the higher connectivity of the Tokyo chip is not useful for the 13 qubit circuit.

5 Conclusion

5.1. Conclusion

Peforming FT distance-3 QEC on NISQ devices is an important near-term goal towards universal FT quantum computation. Currently, NISQ devices are being designed to implement the distance-3 rotated surface code. This is the leading candidate for QEC due to its high pseudo-threshold and its simple two-dimensional structure (i.e. 2D array of qubits with nearest neighbour interactions). It has recently been shown that different FT versions of the Steane code can be mapped to the topology of current NISQ devices such as IBM's Tokyo processor [34] and the Surface-17 transmon processor [36], using flag-bridge qubits [1]. These FT versions of the Steane code have a similar performance (i.e. pseudo-threshold) to the distance-3 rotated surface code for specific noise models.

This thesis extends on the work in [1]. The aim of this thesis is to improve the pseudo-threshold of the flag-bridge based Steane code. This is done by exploring different neural network decoder approaches and by optimizing the FT flag-bridge error syndrome extraction circuits (reducing the number of timesteps).

We introduce two neural network decoder designs for the circuits in this thesis, a high level decoder and low level decoder. We simulate the approximate upper bounds for these two decoders and find that the low level decoder outperforms the high level decoder for error models with low idling error rates. The pseudo-thresholds found for the low level decoder are on average 12% higher than the pseudo-thresholds found for the high level decoder. The reason that the high level decoder does not have the same performance is due to the fact that the LUT does not always provide valid pure errors.

For the circuit level noise model, where the probability of idling errors is equal to that of two qubit gates, single qubit gates and SPAM errors, measurement errors play a smaller role. For this error model the high level decoder outperforms the low level decoder for three of the five circuits considered in this thesis. For these three circuits, c1-l1, c2-l1 and c3-l2 the logical error rate of the high level decoder is on average 12%,13% and 8% lower than the logical error rate of the low level decoder, respectively. For the other two circuits, c1-l2 and c2-l2, the difference in logical error rates of the decoders is smaller than our confidence interval. Our findings are similar to that of simulations of the surface code with the circuit level noise model, as for this code and error model high level decoders outperform low level decoders [16, 17, 45].

For each of the four idling error rates considered, we train a high level neural network decoders for circuit c3-l2. We find that the neural network successfully corrects logical errors for all idling error rates and therefore the high level neural network decoder decreases the logical error compared to the LUT decoder. For the three lowest idling error rates, the high level neural network decoder has on average an 8% higher pseudo-threshold than the LUT decoder.

Next to exploring different decoder approaches, we have proposed new FT flag-bridge syndrome extraction circuits. We show that if different error syndrome extraction circuits, measuring subsets of stabilizer generators of the Steane circuits, are performed in parallel, the flag-bridge error syndrome extraction circuits remain FT. To c An automated brute-force check proved that the new circuits are FT. This was used to optimize the flag-bridge circuits on the topology of QuTech's Surface-17 processor and IBM's Tokyo processor. The upper bound pseudo-threshold of the optimized circuits was found using a high level decoder dataset and compared to the distance-3 rotated surface code pseudo-threshold. The highest pseudo-threshold found is 1.7×10^{-4} for a flag-bridge circuit using 13 qubits on the Surface-17 device. The pseudo-threshold is lower

than that of the surface code, which is 4.44×10^{-4}, but the circuit requires four less qubits. It is found that for the flag-bridge circuit with 13 qubits, the extra connectivity of the Tokyo topology does not offer an advantage over the Surface-17 topology. On the Surface-17 layout, only flag-bridge circuits with 13 qubits are possible, while on the Tokyo topology circuits with 11 and 12 qubits are possible. The psuedo-threshold of these circuits with 11 and 12 qubits are 1.3×10^{-4} and 1.0×10^{-4}.

5.2. Outlook

The optimized flag-bridge error syndrome extraction circuits for the Steane code open exciting prospects for future research. One possible direction could be to test the circuits with a more accurate simulator of quantum hardware, for example a full density matrix simulator such as Quantumsim [49]. In this thesis we have only considered the topology of two NISQ devices, while in reality the NISQ devices have more restrictions. These restrictions should be taken into account when doing a density matrix simulation.

For decoding the optimized circuits we did not train neural networks. It would be interesting to do this and to do a design space exploration to find how the performance of the neural network depends on its design. Parameters that could be looked at are the number of layers, layer size and transfer function.

Another promising research direction would be to extend the optimized circuit to higher distances. Using the 6.6.6 colour code the circuit can be extended in a straightforward manner to higher distances on a grid topology. When this is done the first step would be to test if the circuit is FT. Ideally, a proof of fault-tolerance would be found such as the one in [50] which extends to arbitrary distance. After this is done, it would be interesting to calculate the exREC pseudo-thresholds and compare them to the Surface code. The flag-bridge implementation of the 6.6.6 colour code with two qubits for each stabilizer could potentially have some advantages when performing logical operations due to the transversal properties of the Steane code.

Bibliography

[1] Lingling Lao and Carmen G. Almudever. Fault-tolerant quantum error correction on near-term quantum processors using flag and bridge qubits, 2019.

[2] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[3] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[4] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997. ISSN 1095-7111. doi: 10.1137/s0097539795293172. URL http://dx.doi.org/10.1137/S0097539795293172.

[5] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, Jul 1997. ISSN 1079-7114. doi: 10.1103/physrevlett.79.325. URL http://dx.doi.org/10.1103/PhysRevLett.79.325.

[6] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011. ISBN 1107002176, 9781107002173.

[7] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. doi: 10.1038/s41586-019-1666-5. URL https://doi.org/10.1038/s41586-019-1666-5.

[8] John Preskill. Quantum computing and the entanglement frontier, 2012.

[9] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52: R2493–R2496, Oct 1995. doi: 10.1103/PhysRevA.52.R2493. URL https://link.aps.org/doi/10.1103/PhysRevA.52.R2493.

[10] Peter W. Shor. Fault-tolerant quantum computation, 1996.

[11] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[12] Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice. Fault-tolerant quantum computing with color codes, 2011.

[13] D. Nigg, M. Muller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt. Quantum computations on a topologically encoded qubit. *Science*, 345(6194):302–305, jun 2014. doi: 10.1126/science.1253742. URL https://doi.org/10.1126%2Fscience.1253742.

[14] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I. C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and John M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519:66 EP –, 03 2015. URL https://doi.org/10.1038/nature14270.

[15] Rui Chao and Ben W Reichardt. Quantum error correction with only two extra qubits. *Physical review letters*, 121(5):050502, 2018.

[16] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, nov 2017. doi: 10.1088/2058-9565/aa955a. URL https://doi.org/10.10882%F2058-9565%2Faa955a.

[17] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network based decoder. *arXiv preprint arXiv:1901.10847*, 2019.

[18] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network–based decoder. *Quantum Machine Intelligence*, 2(1):1–12, 2020.

[19] Nikolas P. Breuckmann and Xiaotong Ni. Scalable Neural Network Decoders for Higher Dimensional Quantum Codes. *Quantum*, 2:68, May 2018. ISSN 2521-327X. doi: 10.22331/q-2018-05-24-68. URL https://doi.org/10.22331/q-2018-05-24-68.

[20] P. Baireuther, M. D. Caio, B. Criger, C. W. J. Beenakker, and T. E. O'Brien. Neural network decoder for topological color codes with circuit level noise. *New Journal of Physics*, 21(1):013003, January 2019. doi: 10.1088/1367-2630/aaf29e.

[21] Yuli V. Nazarov and Jeroen Danon. *Advanced Quantum Mechanics: A Practical Guide*. Cambridge University Press, 2013. doi: 10.1017/CBO9780511980428.

[22] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[23] Daniel A Lidar and Todd A Brun. *Quantum error correction*. Cambridge university press, 2013.

[24] Keisuke Fujii. *Quantum computation with topological codes: from qubit to topological fault-tolerance*, volume 8. Springer, 2015.

[25] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, Jan 1998. ISSN 1471-2946. doi: 10.1098/rspa.1998.0167. URL http://dx.doi.org/10.1098/rspa.1998.0167.

[26] Andrew J Landahl, Jonas T Anderson, and Patrick R Rice. Fault-tolerant quantum computing with color codes. *arXiv preprint arXiv:1108.5738*, 2011.

[27] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[28] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, pages 56–65. IEEE, 1996.

[29] Andrew M Steane. Active stabilization, quantum computation, and quantum state synthesis. *Physical Review Letters*, 78(11):2252, 1997.

[30] Emanuel Knill. Quantum computing with realistically noisy devices. *Nature*, 434(7029):39–44, 2005.

[31] David P DiVincenzo and Panos Aliferis. Effective fault-tolerant quantum computation with slow measurements. *Physical review letters*, 98(2):020501, 2007.

[32] Theodore J Yoder and Isaac H Kim. The surface code with a twist. *Quantum*, 1:2, 2017.

[33] Ben W. Reichardt. Fault-tolerant quantum error correction for steane's seven-qubit color code with few or no extra qubits, 2018.

[34] IBM. Quantum experience. URL https://www.research.ibm.com/ibm-q.

[35] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Phys. Rev. Applied*, 8:034021, Sep 2017. doi: 10.1103/PhysRevApplied.8.034021. URL https://link.aps.org/doi/10.1103/PhysRevApplied.8.034021.

[36] Christian Kraglund Andersen, Ants Remm, Stefania Lazar, Sebastian Krinner, Johannes Heinsoo, Jean-Claude Besse, Mihai Gabureac, Andreas Wallraff, and Christopher Eichler. Entanglement stabilization using ancilla-based parity detection and real-time feedback in superconducting circuits. *npj Quantum Information*, 5(1):69, 2019. doi: 10.1038/s41534-019-0185-4. URL https://doi.org/10.1038/s41534-019-0185-4.

[37] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2): 307, 2015.

[38] Suzana Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences*, 109(Supplement 1):10661–10668, 2012.

[39] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[40] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[41] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[44] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3(4):044002, 2018.

[45] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Designing neural network based decoders for surface codes. *arXiv preprint arXiv:1811.12456*, 2018.

[46] A Robert Calderbank, Eric M Rains, PM Shor, and Neil JA Sloane. Quantum error correction via codes over gf (4). *IEEE Transactions on Information Theory*, 44(4):1369–1387, 1998.

[47] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[48] Ramon Overwater. Cryogenic Hardware Considerations Of Neural Network Decoders For Quantum Error Correction Using Rotated Surface Codes. Master's thesis, Technical University of Delft, the Netherlands, 2019.

[49] TE O'Brien, B Tarasinski, and L DiCarlo. Density-matrix simulation of small surface codes under current and projected experimental noise. *npj Quantum Information*, 3(1):1–8, 2017.

[50] Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. Topological and subsystem codes on low-degree graphs with flag qubits. *Physical Review X*, 10(1): 011022, 2020.

A

Fault-tolerant flag-bridge circuits

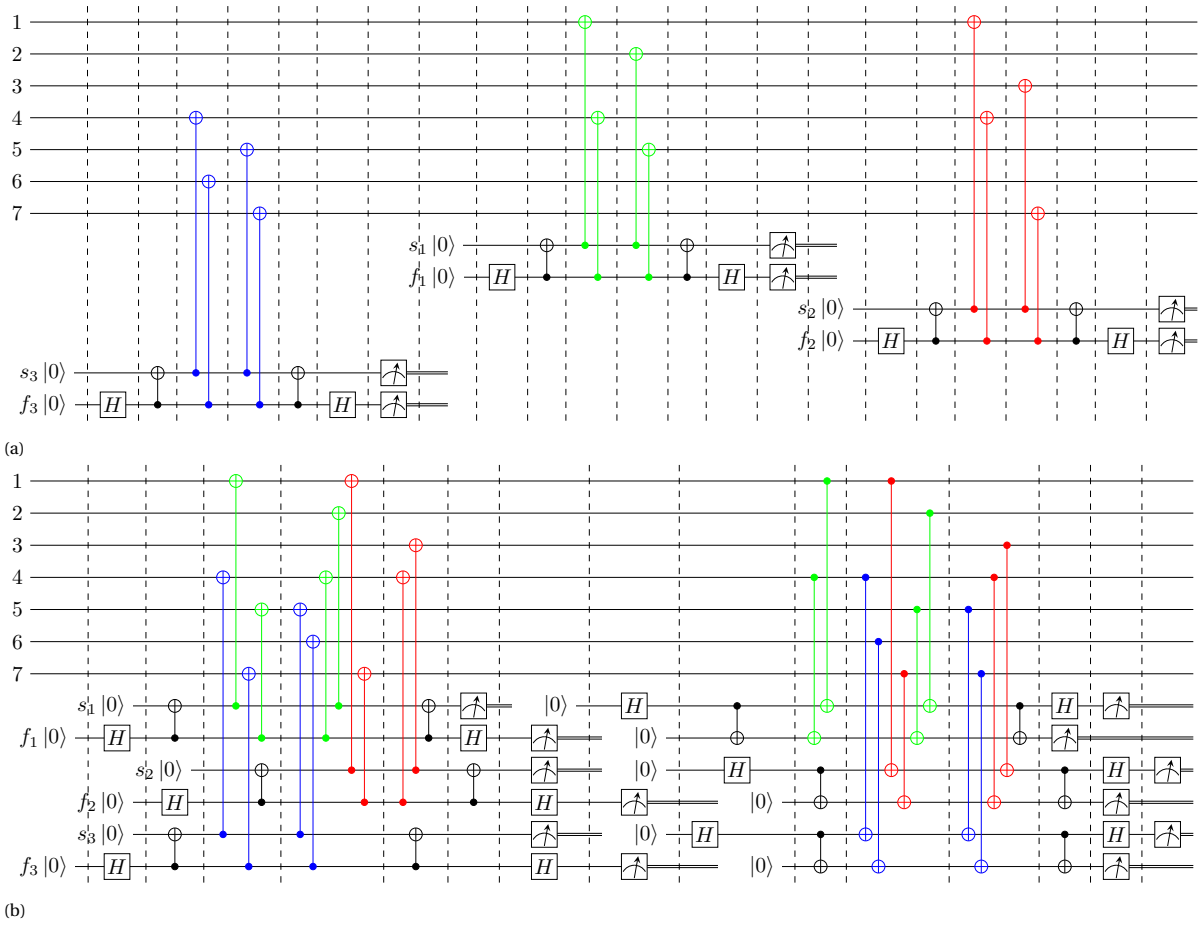


Figure A.1: (a) Circuit c1-l2 with three sequential Z-syndrome extraction circuits. (b) Circuit Tokyo-13 with X- and Z-syndrome extraction circuits in parallel.

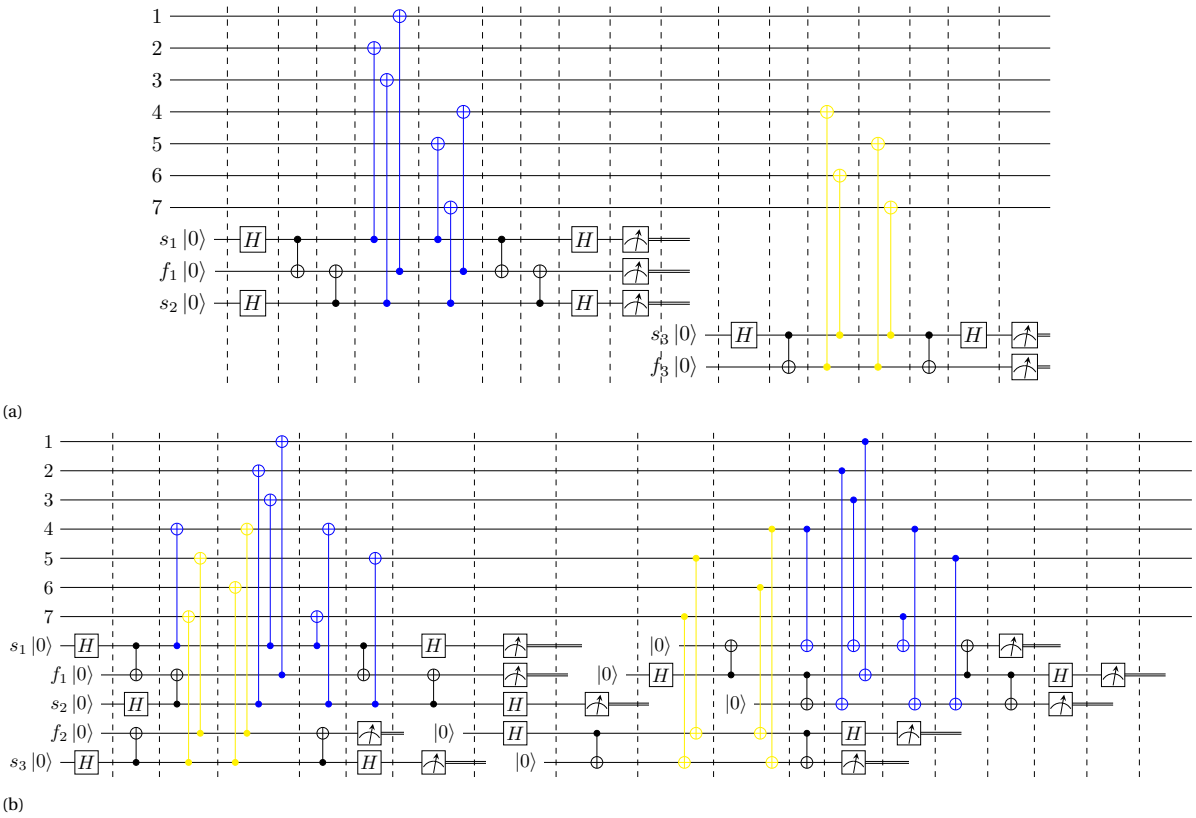


Figure A.2: (a) Circuit c2-l2 with two sequential Z-syndrome extraction circuits. (b) Circuit Tokyo-12 with X- and Z-syndrome extraction circuits in parallel.

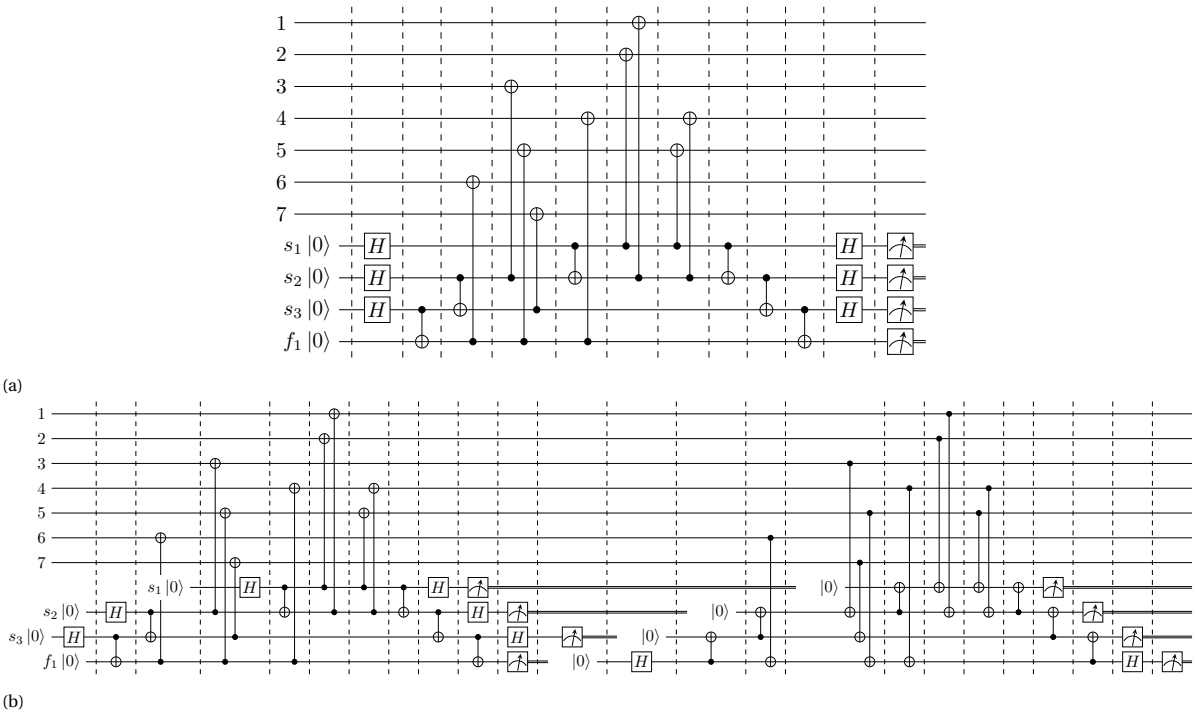


Figure A.3: (a) Circuit c3-l2 with one Z-syndrome extraction circuit. (b) Circuit Tokyo-11 with X- and Z-syndrome extraction circuits.

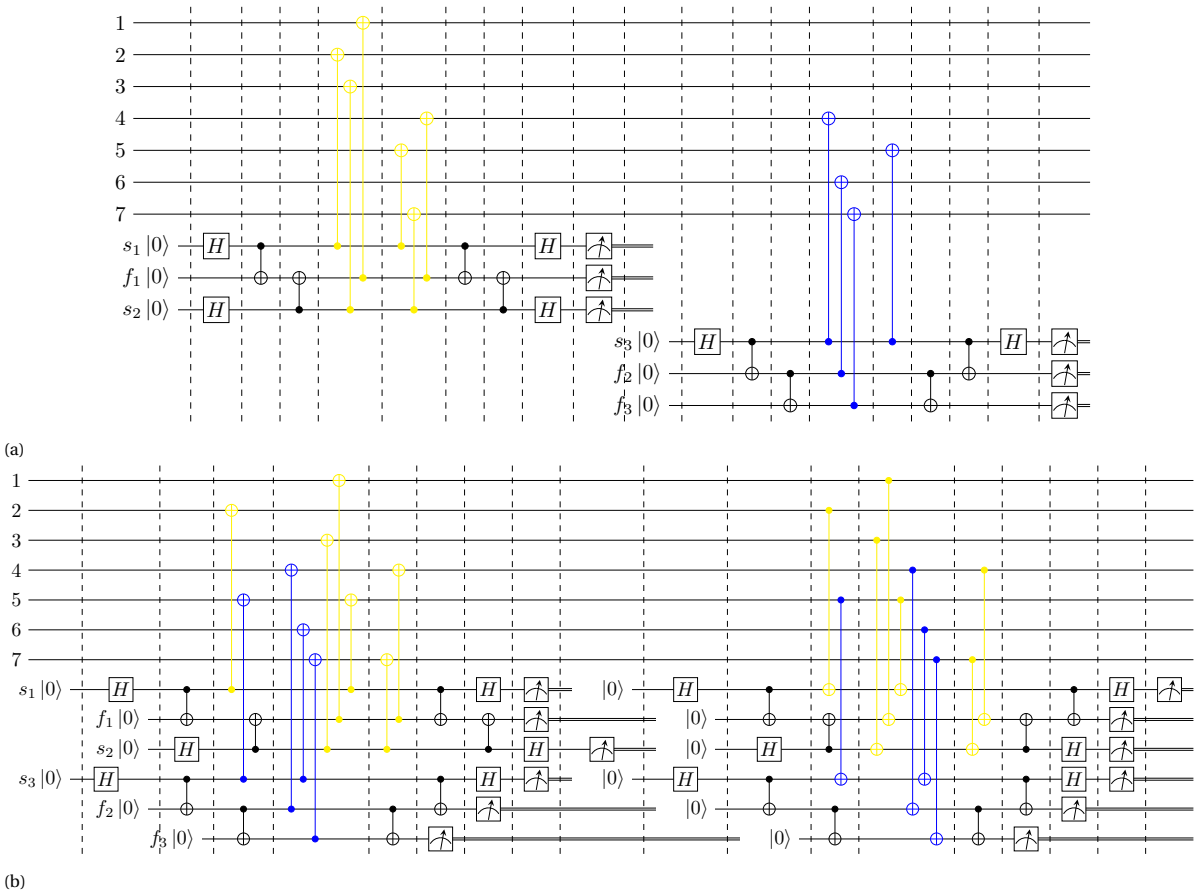
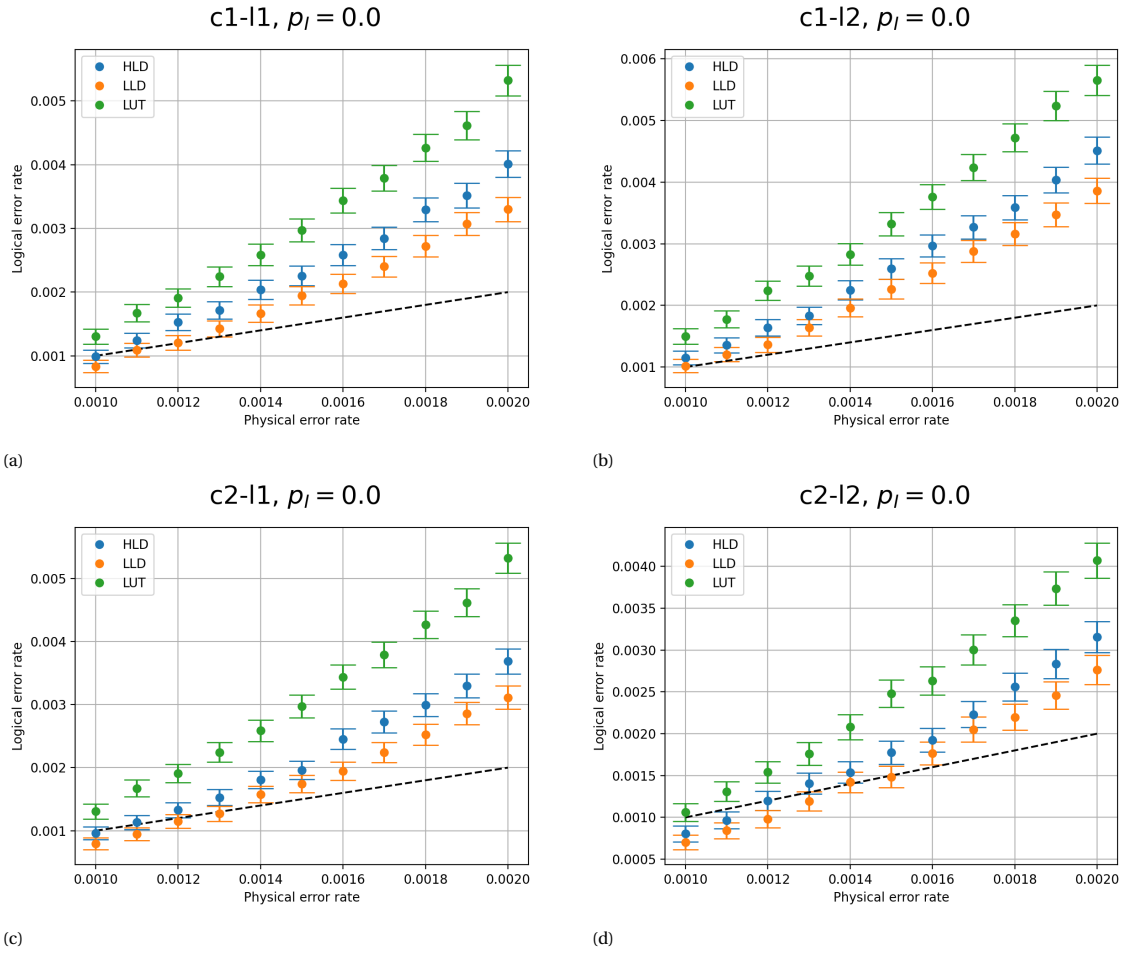


Figure A.4: (a) Circuit c2-l1 with two sequential Z-syndrome extraction circuits. (b) Circuit s17-33 with X- and Z-syndrome extraction circuits in parallel.

B

Pseudo-threshold plots



(a) (b) (c) (d)

Figure B.1: Numerical simulation of three different decoders on four different flag-bridge circuits. The noise model used is $p_1 = p_2 = p_M$ with $p_I = 0$.

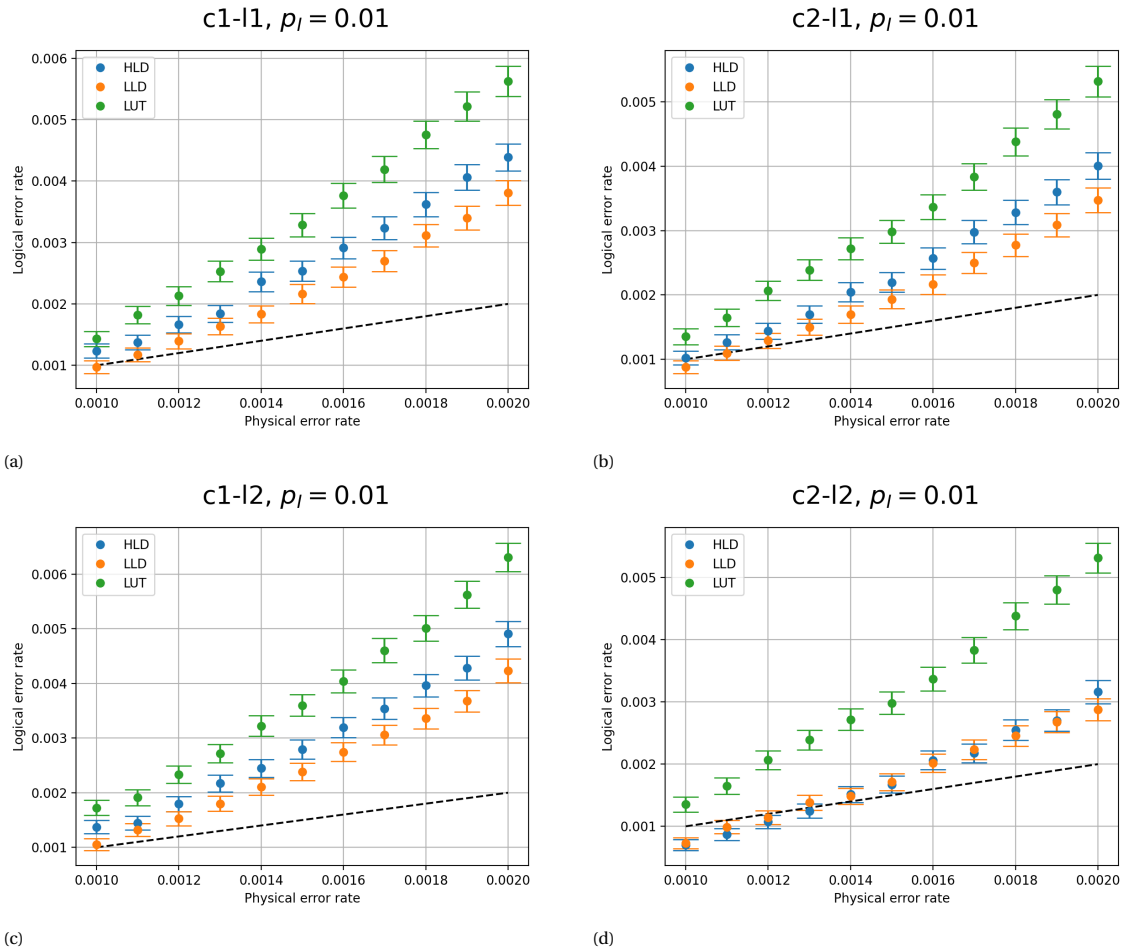
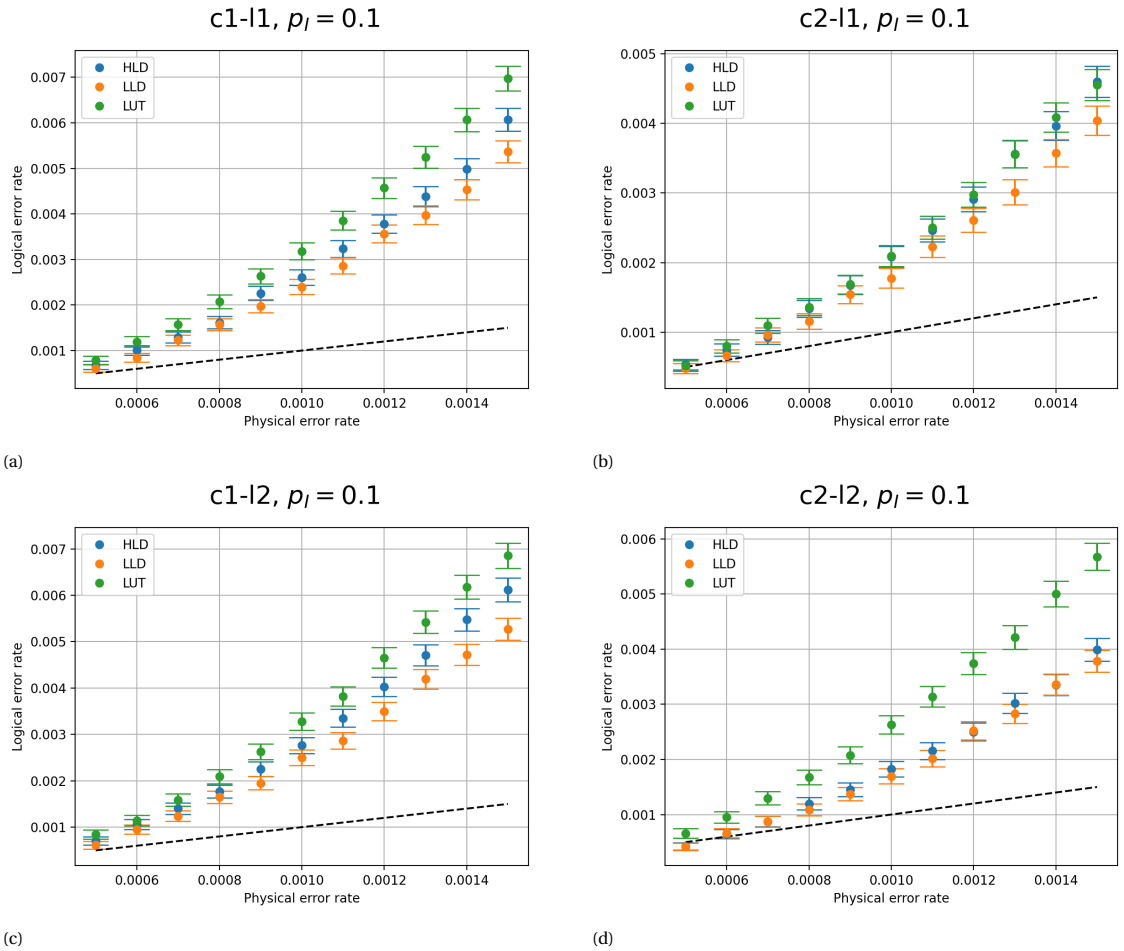
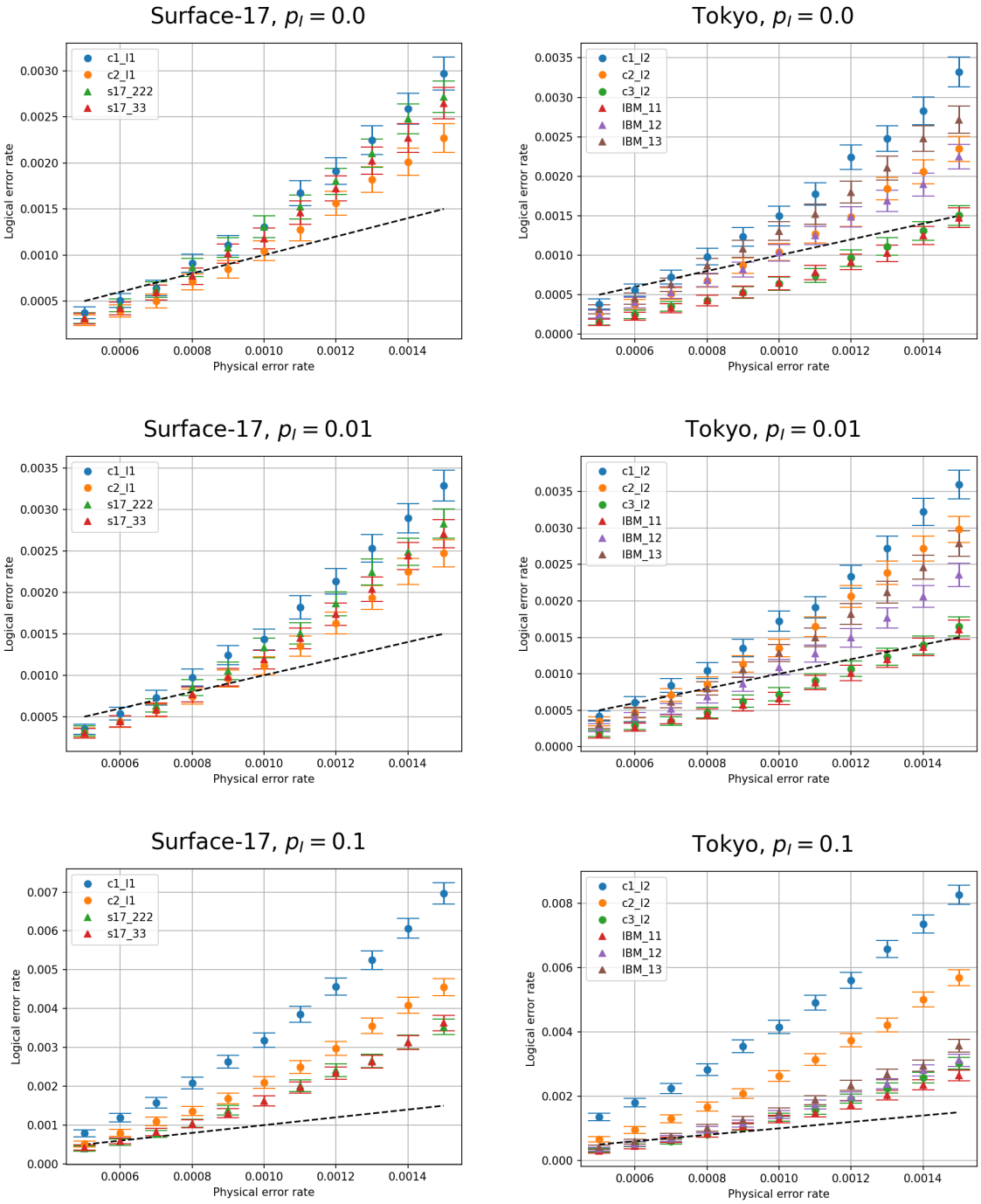


Figure B.2: Numerical simulation of three different decoders on four different flag-bridge circuits. The noise model used is $p_1 = p_2 = p_M$ with $p_I = 0.01$.



(a) (b) (c) (d)

Figure B.3: Numerical simulation of three different decoders on four different flag-bridge circuits. The noise model used is $p_1 = p_2 = p_M$ with $p_I = 0.1$.



(a)

Figure B.4: Numerical simulations of all flag-bridge circuits with the LUT decoder. The noise model used is $p_1 = p_2 = p_M$ with $p_I = 0$, 0.01 and 0.1.

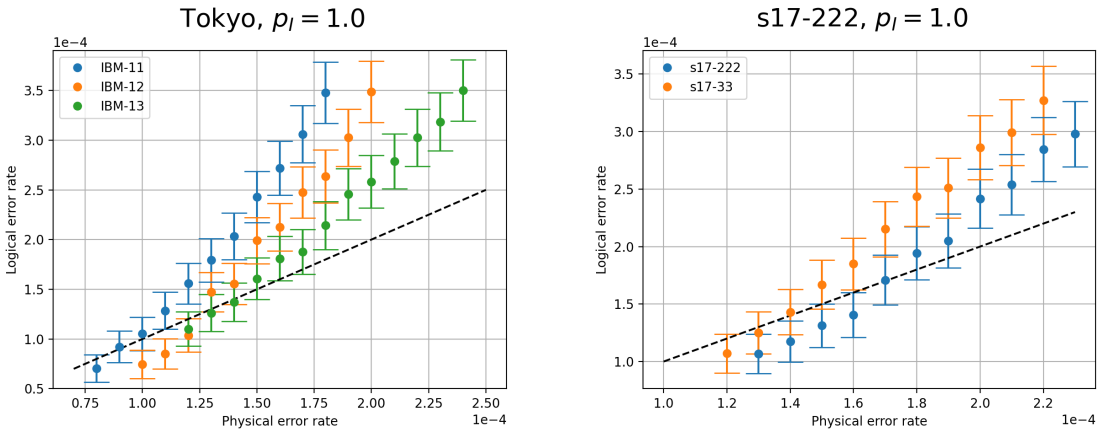


Figure B.5: Numerical simulation of the optimized circuits using the HLD decoder using the circuit level noise model.