Delft University of Technology

Neural Network based Decoders for the Surface Code

Varsamopoulos, Savvas

DOI
10.4233/uuid:dc73e1ff-0496-459a-986f-de37f7f250c9

Publication date
2019

Document Version
Final published version

Citation (APA)
Varsamopoulos, S. (2019). *Neural Network based Decoders for the Surface Code*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:dc73e1ff-0496-459a-986f-de37f7f250c9

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

NEURAL NETWORK BASED DECODERS FOR THE SURFACE CODE

NEURAL NETWORK BASED DECODERS FOR THE SURFACE CODE

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Tuesday 4 June 2019 at 10:00 o'clock

by

Savvas VARSAMOPOULOS

Master of Science in Electronic Physics,
Aristotle University of Thessaloniki, Greece
born in Thessaloniki, Greece.

This dissertation has been approved by the promotors:

Prof. Dr. ir. K. L. M. Bertels (promotor)
Dr. C. Garcia Almudever (copromotor)

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. Dr. ir. K. L. M. Bertels,	Delft University of Technology, promotor
Dr. C. Garcia Almudever,	Delft University of Technology, copromotor
Independent members:	
Prof. Dr. Ir. S. Hamdioui,	Delft University of Technology
Prof. Dr. D. DiVincenzo,	RWTH Aachen, Germany
Prof. Dr. C. W. J. Beenakker,	Leiden University, The Netherlands
Dr. O. E. Scharenborg,	Delft University of Technology
Other Members:	
Dr. Ir. Z. Al-Ars,	Delft University of Technology

Keywords:	Quantum Error Correction, Quantum Error Detection, Neural Network based Decoders
Printed by:	Proefschriftmaken.nl
Front & Back:	Beautiful cover art created by Despoina Pouniou.

Copyright © 2019 by S. Varsamopoulos

ISBN 000-00-0000-000-0

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

ACKNOWLEDGEMENTS

I would like to show my appreciation to all the people that stood by me during my Ph.D. journey. The last four and a half years, constitute a beautiful chapter in my life that I will never forget.

First of all, I have to show my immense gratitude to Prof. Dr. Koen Bertels, who is the reason that I had the chance to come to Delft. **Koen** believed in me from the first Skype interview that we did and happily invited me to his newly formed team. Although that the first couple of years included a lot of learning and patience, Koen kept believing in me. He supported my ideas and gave me valuable advice and suggestions.

Next, I would like to thank Dr. Carmen Garcia Almudever for all the invaluable things that she has done. **Carmina**, you have been the most involved person in my research and you are an intricate part of writing this thesis. Thank you for teaching me and learning with me, thank you for the time spend reviewing my presentations, reports, posters and papers. Thank you for all the conversations and thank you for trusting my instincts.

I am also incredibly thankful to Dr. Daniel B. Criger. **Ben**, you played the role of the mentor and teacher when I needed it the most. You were there for me at every step of the road ever since our collaboration started. Your critical thinking and your undying passion to advance science is mesmerizing. You always guided me and gave me confidence to develop innovative work.

Dr. **Xiang** Fu, you are one of the most hard-working and persistent people that I know. You were always making detailed arguments and looked beneath the surface. Thank you for being there with me when our Ph.D. journey started. **Ling Ling**, you are very smart and very sweet. You are a great researcher and I enjoyed our conversations a lot. You are the mastermind of our group and everybody listens carefully to what you suggest. **Leon**, you are very charismatic. From science till music and from serious conversations till partying until the early morning, you can do it all. I am deeply thankful for our collaboration that lead to a publication. Your office will always be reminded as the office that everyone had a great time. **Nader** and **Imran**, you are both smart and humble. You are very productive and have proven to work efficiently through very stressful periods. Your suggestions aided in the development of my work and improved my writing skills. **Hans**, it was a pleasure interacting with you. You always asked the right questions and were eager to learn even more. Moreover, I would like to thank all the recent colleagues in the Quantum and Computer engineering lab, **Aritra, Abid, Miguel, Amitabh, Daniel, Alejandro, Anneriet**, for the interesting conversations and the fun that we had together. Also, I would like to thank **Andreas** and **Nikolas**, the Greeks that were part of the Quantum Computer Architecture lab. Even though that your stay was short, you really made a difference to my everyday life. Talking to you was always relaxing and inspiring. Special thanks also to the secretaries of Quantum and Computer engineering lab, **Lidwina** and **Joyce** for their help in administration issues.

A big thank you goes out to **Prof. Dr. Leonardo Di Carlo** and all the members of his group, for the frequent interaction that we had and the stimulating discussions. The DiCarlo group is one of the best groups in their field and the people working there are excellent researchers. Thank you for all the Monday meetings and the suggestions that you always gave me.

Finally, I would like to thank all my close friends, my girlfriend and my parents for supporting me throughout this period of my life. Completing a Ph.D. is not an easy task and there are a lot of ups and downs. However, having these people behind me, gave me the strength to accomplish my goal. Thank you!

CONTENTS

Acknowledgements **v**

List of Tables **ix**

List of Figures **xi**

Table of Acronyms **xvii**

1 Introduction **1**
- 1.1 Building a fault tolerant quantum computer 1
- 1.2 Quantum error correction 3
- 1.3 Research challenges 5
- 1.4 Dissertation contributions and organization 6

2 Background **9**
- 2.1 The power of quantum computing 9
- 2.2 Fault tolerant computation and quantum error correction 14
  - 2.2.1 Quantum error correcting codes 15
  - 2.2.2 Surface code 16
  - 2.2.3 Decoding the surface code 19
  - 2.2.4 Quantum error decoders 22
- 2.3 Quantum errors. 24
  - 2.3.1 Error propagation and transformation 24
  - 2.3.2 Simulated error models 25
- 2.4 Neural networks 26

3 Pauli Frames for Quantum Computer Architectures **31**
- 3.1 Introduction 31
- 3.2 Background 32
  - 3.2.1 Quantum error correction 33
- 3.3 Pauli frames 35
- 3.4 A quantum computer architecture with Pauli frame 35
  - 3.4.1 Benefits 36
  - 3.4.2 Implementation. 37
- 3.5 Simulation setup 38
  - 3.5.1 Logical error rate calculation. 40
- 3.6 Results 40
- 3.7 Conclusions. 42
- 3.8 Acknowledgments 43

4 Decoding Small Surface Codes with Feedforward Neural Networks 45
4.1 Introduction 45
4.2 Need for fast decoding 46
4.3 Related work 46
4.4 Neural network decoder 47
4.5 Results 49
4.6 Discussion and conclusion 55
4.7 Acknowledgments 57

5 Designing neural network based decoders for surface codes 59
5.1 Introduction 59
5.2 Designing neural network based decoders 62
5.2.1 Inputs/Outputs. 63
5.2.2 Sampling and training process. 63
5.2.3 Evaluating performance 63
5.3 Implementation parameters 65
5.3.1 Error model 66
5.3.2 Choosing the best dataset 66
5.3.3 Structure of the neural network 67
5.3.4 Training process 70
5.4 Results 72
5.4.1 Depolarizing error model 73
5.4.2 Circuit noise model. 74
5.5 Conclusions. 76

6 Decoding surface code with a distributed neural network based decoder 77
6.1 Introduction 77
6.2 RG decoding 79
6.3 Distributed decoding with overlapping tiles 81
6.4 Results 83
6.4.1 Optimizing for the size of training dataset 86
6.5 Conclusions. 88

7 Conclusions and future outlook 91
7.1 Conclusions. 91
7.2 Future research direction. 93

Bibliography 94
References 94

Summary 103

Samenvatting 105

Curriculum Vitæ 107

List of Publications 109

LIST OF TABLES

2.1 Comparison between decoders 23

3.1 Variables used in the execution schedules. 34
3.2 Execution steps for different operations when using a Pauli frame. . 36
3.3 The ESM circuit used in our test setup. 39

4.1 Layer sizes for the neural networks used throughout this work. The number of input nodes is determined by the number of syndromes in the quantum error correction scenario, using only X (or Z) syndrome bits for independent X/Z errors, and all syndrome bits for depolarizing errors. For fault tolerance error models, d rounds of measurement are followed by readout of the data qubits, and calculated stabilizer eigenvalues are included in the input. The output layer is restricted to two nodes for independent X/Z errors, since logical X/Z errors are also independent. In all other scenarios, four nodes are used to discriminate between $\mathbb{I}$, $\bar{X}$, $\bar{Y}$, and $\bar{Z}$. The number of nodes in the hidden layer is determined by analyzing the performance of the resulting decoder empirically. 49

5.1 Pseudo-threshold values for the tested decoders (d=3) under depolarizing error model 69
5.2 Average time for surface code cycle under depolarizing error model 72
5.3 Pseudo-threshold values for the depolarizing error model 73
5.4 Pseudo-threshold values for the circuit noise model 74

6.1 Size of training datasets 80
6.2 Reduction in required inputs of the neural network 84

LIST OF FIGURES

1.1 Overview of the full system stack required for building a quantum computer . . . 3
1.2 Representation of encoding and decoding process . . . 4

2.1 Visualization of a single qubit state in the Bloch sphere . . . 10
2.2 Symbolic representation of the Pauli matrices. . . . 12
2.3 Symbolic representation of the Hadamard, T and Phase gate. . . . 12
2.4 Quantum circuit representation of the CNOT, the CZ and the Toffoli gate. . . . 14
2.5 Quantum circuit describing the application of a single-qubit gate U_s to qubit q_0. . . . 14
2.6 Quantum circuit describing the application of a multi-qubit gate U_m to m qubits. . . . 14
2.7 Popular quantum error correcting codes and their respective families 16
2.8 Surface code structure. Data qubits are placed in the corners, X-type ancilla qubits are placed inside the white squares and Z-type ancilla are placed inside the grey squares. . . . 17
2.9 Error syndrome measurement circuit for the distance-3 rotated surface code [1–3]. Left: Measurement circuit for individual Z tiles (top) and X tiles (bottom), including an ancilla qubit to be placed at the center of each tile as seen at the right side. Ancilla qubits are prepared in the $+1$-eigenstate of the appropriate basis, four CNOT gates are executed, and the ancilla qubits are measured in the appropriate basis. Right: Interleaving of separate parity check measurements, including late preparation and early measurement for weight-two parity checks. . . . 18
2.10 Rotated surface code with code distance 3. Data qubits are placed at the corners of the tiles and are enumerated from 0 to 8 (D0-D8). X-type ancilla are placed in the center of the white tiles and Z-type ancilla are placed in the center of grey tiles. . . . 18
2.11 Rotated surface code with code distance 5. Errors are shown with X or Z on top of the data qubits and detection events that correspond to these errors are shown with red dots. . . . 20
2.12 Rotated surface code with code distance 3 at consecutive time steps. The alternating pattern on the measurement value of the same parity-check, indicates the presence of a measurement error. . . . 20
2.13 Decoding performance indicating the threshold of the surface code and the pseudo-thresholds of each code distance. . . . 21

2.14 Representation of an artificial neural network with 4 layers 27
2.15 A conceptual visualization of the recurrent nature of an RNN. 28
2.16 Structure of the LSTM cell and equations that describe the gates of an LSTM cell. 28

3.1 Schematic of a SC17 logical qubit. 34
3.2 Execution schedule of a SC17. 34
3.3 Execution schedule with Pauli frame. 37
3.4 Simplified architecture of a QCA targeting a SC17 logical qubit. . . . 37
3.5 High level architectural view of the PFU. 38
3.6 ESM results used for successive decoding windows. 39
3.7 The QPDO control stack used for the simulations. 40
3.8 Calculated LER for X_L errors for d=3 rotated surface code under circuit noise error model. The points denoted with circles are assuming the use of a Pauli frame and the points with squares do not. Each curve depicted with a different color, assumes a different t_{d} as described in the legend. 41
3.9 Relative reduction in t_{cycle} and LER by using a Pauli frame for different t_{d}. 42

4.1 A surface code error E decomposed into three components; a stabilizer S, a fixed Pauli C which produces the same syndrome as E, and a logical operator L. 47
4.2 The graphical and functional descriptions of a feed-forward neural network. In the graphical description (left), inputs x_j are passed to neurons in a *hidden layer*, and each of these neurons outputs $\sigma\left(\vec{w}\cdot\vec{x}+b\right)$, where $\vec{w}$ and b are a local set of weights and a bias, and $\sigma(x)$ is a nonlinear *activation function* (we use $\sigma(x)=(1+\exp(-x))^{-1}$ for all neurons considered in this work). The final outputs y_k can be rounded to $\{0,1\}$, and interpreted as a class label. In the functional picture, the weights and biases are assembled into matrices and vectors, respectively, allowing the output vector to be expressed as a composition of functions acting on the input vector. 48
4.3 Code capacity error model without measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial lookup table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x=y$. All points of all three curves are lying on top of each other. 50
4.4 Code capacity error model without measurement errors for Surface Code with distance 5. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial lookup table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x=y$. 51

4.5 Code capacity error model without measurement errors for Surface Code with distance 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 51

4.6 Depolarizing error model without measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 52

4.7 Depolarizing error model without measurement errors for Surface Code with distance 5. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 52

4.8 Depolarizing error model without measurement errors for Surface Code with distance 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 53

4.9 Code capacity error model with measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 54

4.10 Depolarizing error model with measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 54

4.11 Circuit error model for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. 55

5.1 Overview of the quantum computer system stack 60

5.2 Overview of the quantum micro-architecture [4] 61

5.3 Description of the decoding process of the low level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) **Invalid** data qubits corrections and the corresponding error syndrome. (c) **Valid** data qubits corrections and the corresponding error syndrome. 64

5.4 Description of the decoding process of the high level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) Corrections proposed by the simple decoder for the observed error syndrome. (c) Additional corrections in the form of the $\bar{X}$ logical operator to cancel the logical error generated from the proposed corrections of the simple decoder. 65
5.5 Different configurations of layers and nodes for the d=5 rotated surface code for the ***depolarizing error model***. The nodes of the tested hidden layers are presented in the legend. Training stops at 500 training epochs for all configurations, since a good indication of the training accuracy achieved is evident by that point. Then, the one that reached the highest training accuracy continues training until the training accuracy cannot increase any more. 68
5.6 Left: Comparison of decoding performance between Blossom algorithm, low level decoder and high level decoder for the ***d=3*** rotated surface code for the ***depolarizing error model***. Right: Zoomed in at the region defined by the square. 69
5.7 Execution time for the high level decoder (hld) and the low level decoder (lld) for Feed-forward (FFNN) and Recurrent neural networks (RNN) for d=3 rotated surface code for the ***depolarizing error model***. 71
5.8 The design for the high level decoder that was used for the depolarizing and the circuit noise model. 72
5.9 Decoding performance comparison between the high level decoder trained on a single probability dataset, the high level decoder trained on multiple probabilities datasets and Blossom algorithm for the ***depolarizing error model*** with perfect error syndrome measurements. Each point has a confidence interval of 99.9%. 73
5.10 Decoding performance comparison between the high level decoder trained on a single probability dataset and the high level decoder trained on multiple probabilities datasets for the ***circuit noise model*** with imperfect error syndrome measurements. Each point has a confidence interval of 99.9%. 75

6.1 Abstract comparison between decoding performance and execution time of various decoding algorithms 78
6.2 Encoding levels of a concatenated code. At level 0 there are nine qubits, that are encoded in three qubits at level 1 and these qubits are encoded in one qubit at level 2. Arrows show the information flow. 80
6.3 Tile segmentation that represents the levels of concatenation in a concatenated code. The smallest level of concatenation is represented by the green tiles, the next level of concatenation is represented by the red tiles, the following level of concatenation is represented by the blue tiles, etc. 81

6.4 Description of the simple decoder operation for the rotated surface code with distance 5. Detection events are presented with the red dots. Red lines indicate which data qubits are going to be corrected. . 82
6.5 Segmentation of a d=5 rotated surface code into four overlapping tiles of d=3 rotated surface codes. 83
6.6 Comparison of decoding performance between the distributed decoder with ***four*** overlapping tiles of d=3 rotated surface codes inside a d=5 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green). 85
6.7 Comparison of decoding performance between the distributed decoder with ***nine*** overlapping tiles of d=3 rotated surface codes inside a d=7 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green). 85
6.8 Comparison of decoding performance between the distributed decoder with ***sixteen*** overlapping tiles of d=3 rotated surface codes inside a d=9 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green). 86
6.9 Description of the design flow of the optimized version of the distributed decoder. 87
6.10 Comparison between the optimized version of the distributed decoding (blue) to the unoptimized version (red), the unoptimized version of the Blossom algorithm (pink) and the neural network based decoder (green). 88

TABLE OF ACRONYMS

2D	Two-dimensional
3D	Three-dimensional
FPGA	Field Programmable Gate Array
FT	Fault Tolerant
GPU	Graphical Processing Unit
CPU	Gentral Processing Unit
HDL	Hardware Description Language
NISQ	Noisy Intermediate-Scale Quantum
NN	Neural Network
PF	Pauli Frame
QEC	Quantum Error Correction
QECC	Quantum Error Correction Code
QED	Quantum Error Detection
ESM	Error Syndrome Measurement
LER	Logical Error Rate
PER	Physical Error Rate
PEL	Physical Execution Layer
PFU	Pauli Frame Unit
QPDO	Quantum Platform Development Framework
QCU	Quantum Control Logic
QISA	Quantum Instruction Set Architecture
RBLUT	Rule Based Look Up Table
PLUT	Partial Look Up Table
ASIC	Application-Specific Integrated Circuit
RG	Renormalization Group
MWPM	Minimum Weight Perfect Matching
MLD	Maximum Likelihood decoder
MCMC	Markov Chain Monte Carlo
CA	Cellular Automaton
NNbD	Neural Network based Decoder
LSTM	Long Short-Term Memory
TPU	Tensor Processing Unit
FFNN	Feed-Forward Neural Network
RNN	Recurrent Neural Network

1

INTRODUCTION

1.1. BUILDING A FAULT TOLERANT QUANTUM COMPUTER

Quantum computers as a theoretical model and later as realistic implementation have been studied for almost 40 years now. The main reason is the immense computational power that quantum computers can potentially achieve through ***superposition*** and ***entanglement***, and the fact that they can produce solutions to NP-hard problems that are intractable or need an exponential amount of time to be solved classically.

The idea of building a quantum computer was introduced by Feynman in 1981 [5]. Feynman proposed to use a universal quantum computer that would be able to simulate in an exact way a quantum system, instead of a classical computer that will perform an approximate simulation. Following this theoretical idea of building a quantum computer, the same year Toffoli introduces the Toffoli gate which together with the NOT and XOR gates provides a set of reversible gates that can be used for reversible classical computation. In 1982, Benioff introduces the first theoretical framework of a quantum computer [6] and in 1985 Deutsch describes a universal quantum computer similar to the concept of the universal Turing machine [7]. From 1994 until 1997, there are significant advances in quantum computing in terms of quantum algorithms and first experimental implementations. Shor introduces Shor's algorithm [8], which is a factorization algorithm of prime numbers that can achieve almost exponential speedup compared to its classical counterpart [9]. Grover proposes the quantum database search algorithm, which has a quadratic speedup compared to classical algorithms and can be applied to a wide variety of problems [10]. Shor [11] and Steane [12–14] independently propose their schemes for quantum error correction and Monroe and Wineland implement the first experimental CNOT gate [15]. DiVincenzo proposes a list of criteria that should be satisfied for creating a quantum computer [16]. In 1998, a working two-qubit quantum computer solves Deutsch's problem [17, 18] and a three-qubit working computer based on Nuclear Magnetic Resonance (NMR) is created

[19, 20]. Grover's algorithm is executed and the proof that a subclass of quantum computations can be efficiently simulated in a classical computer (Gottesman–Knill theorem) is provided [21].

In the following decades, many more quantum algorithms have been proposed for various applications, such as quantum simulation [22, 23], search [24], machine learning [25], and solving graph and algebraic problems [26, 27], etc.

The fragility of the quantum bits and the inability of creating scalable devices are the main reasons that are keeping us from the realization of a large-scale quantum computer. There are many competing quantum technologies that are being explored, such as ion traps [28, 29], superconducting [30–32], semiconducting [33–35], nitrogen-vacancy centers [36, 37], nuclear magnetic resonance [19, 20], photon polarization modes [38], majoranas [39], that have their own advantages and disadvantages. Based on the construction and the connectivity of the qubits, quantum computation and storage might be easier with a given quantum technology. At the moment, the most promising quantum technologies are the ion traps and the superconducting, for which demonstration of the DiVincenzo criteria has been accomplished [29–31, 40]. We should also mention that semiconducting technology can potentially scale faster than the other two technologies as proposed in [41].

According to the DiVincenzo criteria [16], all these technologies have to obey the minimum requirements needed to perform quantum computing in a physical system. These requirements are namely i) a scalable physical system with well-characterized qubits, ii) the ability to initialize the state of the qubits to a simple fiducial state, such as $|000\cdots\rangle$, iii) long relevant coherence times, much longer than the gate operation time, iv) a "universal" set of quantum gates and v) a qubit-specific measurement capability.

Moreover, building a quantum computer involves more than developing good quantum technologies. Building a quantum computer implies that the quantum software and quantum hardware will work in tandem. It implies the development of an entire system stack, as shown in Figure 1.1, that connects both hardware and software and allows the execution of quantum algorithms on quantum processors. Quantum algorithms can be written in a high-level language and then translated through a compiler to executable instructions for the hardware called Quantum Instruction Set Architecture (QISA). These instructions are sent to the micro-architecture that among other tasks, it translates the instructions into low-level signals that are sent to operate on the qubits.

An important note that needs to be mentioned is that quantum hardware is still error-prone and control of the quantum hardware is imperfect. These factors will lead to poor quantum computation if no measure against noise is taken. Quantum error correction (QEC) is the suggested solution to that problem, since by employing more resources and continuously monitoring the system, fault tolerant computation can be achieved.

As can be seen from the third dimension of Figure 1.1, various quantum error correcting codes (QECCs) exist and they affect different software and hardware layers of the system stack. For all these reasons, QEC is deemed as necessary in order to have reliable quantum computation and storage with noisy components.

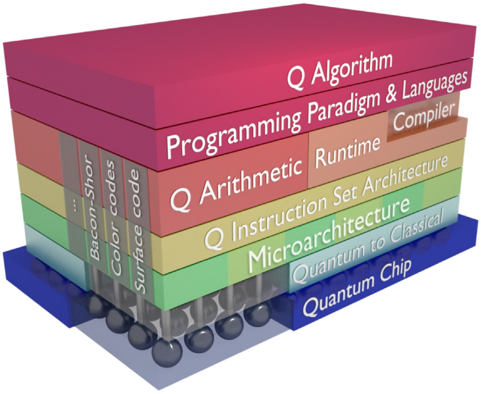


Figure 1.1: Overview of the full system stack required for building a quantum computer

1.2. QUANTUM ERROR CORRECTION

In order to overcome the obstacle of the fragile nature of quantum bits and imperfect quantum operations, an error correction process can be employed. Qubits suffer from decoherence, undesirable alteration of the quantum state, even when there is no action upon them due to unavoidable interaction with their environment. Short coherence times, time that the desired state is available, make the time budget for quantum computation limited. Furthermore, noise can be inserted in the quantum system due to the imperfect application of quantum operations through the classical control electronics.

To counteract the presence of noise, researchers came up with quantum error correction techniques. QEC is the framework that allows reliable performance of quantum computation and safe storage of the quantum information for a long period of time.

The origins of quantum error correction can be found in classical error correction. In classical error correction bits are encoded by making an odd number of copies of the initial bit. For instance a 0 is encoded as 000 and a 1 is encoded as 111. The 000 and 111 are the codewords of the encoded version of the code. Errors can be detected and identified by performing majority voting. This process of error identification is known as decoding.

It can be proven that if the error probability of a single bit is less than 1/2, then the probability of no error while using encoding - decoding techniques is much better [42]. This can be easily seen at the binary symmetric channel, where the probability of no bit-flip error is $(1 - p)$ and the probability of a bit-flip error is p. Therefore, for p < 1/2, error correction increases the fidelity of the outcome. Finally, this type of encoding is known as repetition. By repeating the bit 0 or 1 with itself, one can accomplish safer (less error prone) transmission of the bit value; the probability that two or more bits are flipped is $3p^2(1-p) + p^3$, so the probability of an error is $p_e = 3p^2 - 2p^3$ compared to the probability of an error of the unencoded bit that is $p_e = p$ [42].

However, there are three fundamental differences between classical and quantum computing that make the principles of classical error correction not directly applicable to quantum error correction. These differences are the following:

- A quantum state cannot be copied

  There are already developed schemes that create a copy of a desired pure state with decent fidelity [43], however a copying mechanism of an unknown state is in general prohibited by quantum mechanics [42]. Therefore, we cannot copy a qubit state for encoding. A solution would be to use entanglement between the qubits.

- Quantum errors are continuous

  Quantum errors are continuous, which means that they cannot be completely discretized into Pauli errors with the finite resources that we provide. Such errors can be amplitude decreasing, rotations in a slightly different angle than desired, leakage to non-computational states, and many more [44]. Therefore, qubits do not suffer a complete bit- or phase-flip, but an angular shift. A solution would be to use a discretization process of continuous to Pauli errors.

- Quantum measurements are destructive

  As will be explained in Section 2.1, measuring a qubit that is in a superposition state will project the qubit into a post-measurement state of $|0\rangle$ or $|1\rangle$, destroying the initial state of superposition. Therefore, we cannot directly measure the qubit to detect errors [42, 45]. A solution would be to use extra qubits to perform indirect measurements that do not affect the quantum state, while simultaneously provide error information.

Similar to classical error correction, quantum error correction involves an encoding process of the quantum information into multiple qubits and a decoding process that identifies and counteracts the noise that is inserted in the quantum system, as shown in Figure 1.2.

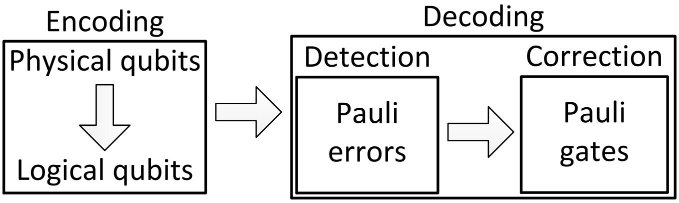


Figure 1.2: Representation of encoding and decoding process

Many unreliable ***physical qubits*** are encoded to one more reliable qubit, known as ***logical qubit***. The encoding process creates logical qubits that are more resilient to noise, however quantum errors can still occur at the physical qubits. Therefore, we need a decoding process that identifies the errors at the physical level.

As mentioned quantum error correction adds extra qubits to the quantum system. These qubits are called ancillary or ancilla qubits and are not used to store quantum information. They are entangled to the qubits that store quantum information, known as data qubits, and perform parity-checks between the state of the data qubits that they are entangled to. By measuring the ancilla qubits, we obtain information about changes in data qubit states (errors), without explicit measurement of the data qubits that will lead to loss of the quantum information. The state of the ancilla qubit is collapsed when it is measured, however this does not affect the state of the quantum system. The measurement result is a binary value that is used to identify the location and type of errors. The collection of all measurement results from the ancilla qubits are forwarded to the classical unit that contains the decoder, which is the topic of this thesis. The decoder proposes corrections based on the measurement results and the corrections can be applied in the quantum system to erase the errors that have accumulated. Since both the errors and the corrections are considered Pauli errors and corrections, they can be easily tracked and monitored in the classical system. A Pauli frame [46] can be used to keep track of errors and corrections, therefore there is no need to apply the corrections to the quantum system and potentially introduce extra errors due to the imperfect application of the corrective gates.

Since errors are generated randomly in the quantum system, constant decoding is required. In a fault tolerant quantum computation, an error correction cycle that includes the state preparation of the ancilla, the entangling process between the ancilla and the data qubits and the measurement of the ancilla, is required after each logical operation. The decoding process should take at most as much time as it is required to do an error correction cycle, so that the corrections out of the decoder are provided in time and no stalling of any operation is required.

Finally, we should mention that there are other forms of quantum error correction, which include techniques that minimize or even completely suppress noise [47], known as a passive quantum error correction or error mitigation [48]. Techniques such as decoherence free subspaces, composite pulse sequences, positive-operator valued measure (POVM) and others are used to avoid many of the error inducing mechanisms [44]. However, in the context of this thesis, we are focusing on the active form of quantum error correction as was described in this section.

1.3. RESEARCH CHALLENGES

In this thesis, we mainly focus on the development of high decoding performance and high speed decoders that can be applied to different quantum error correcting codes, with emphasis on the surface code. The main challenges of developing such decoders are:

- ***Execution time***: It is quantified as the time between the moment that all data have reached the input of the decoder and the moment that the decoder outputs the corrections. Due to the current limited budget for the execution time of the decoder, a high speed implementation is required. The decoder can work in parallel to the quantum error correction circuit, however, it cannot exceed the time budget of quantum error correction. If the execution time is larger than the available time, either a backlog of data is created or the quantum operations need to be stalled. In the former case, errors keep accumulating as they are not being corrected, which will lead to wrongful computation or storage. In the latter case, stalling of quantum operations in order to give more time to the decoder to identify the errors will lead to less operations being applied, since qubits decohere.

- ***Decoding performance***: It is described based on the ability of the decoder to correctly identify physical errors. The more errors that are properly identified and corrected by the decoder, the higher the decoding performance. Typically, the decoding performance is synonymous to the rate of logical errors. Naturally, good decoding performance is the main objective of a decoder. The decoding performance needs to be as high as possible for the given encoding scheme and error model. Furthermore, it needs to stay high (correct a specified minimum amount of errors) as the size of the quantum system increases.

- ***Scalability***: It is the ability of the decoder to successfully decode any quantum system regardless of its size. Successful decoding can be though in terms of the decoding performance and the execution time. In terms of decoding performance, the decoder needs to successfully correct the minimum weight of errors based on the encoding scheme and the quantum error correcting code. In terms of the execution time, the decoder should scale in such a way that no back-log of input data is created while also no quantum operations are stalled to perform the decoding.

More information about these metrics is given throughout the thesis.

1.4. DISSERTATION CONTRIBUTIONS AND ORGANIZATION

In Chapter 2, we provide the background information to topics like quantum error correction, surface code, error models, decoding algorithms and neural networks that are required for the rest of the thesis.

We start our investigation on decoding algorithms in Chapter 3 with a simple rule-based decoder that performs error correction only for the surface code with 17 qubits. This decoder is added in a Pauli frame unit that can perform the execution schedule of the quantum operations required for a surface code cycle. We investigate the case where the error syndrome measurements occur in parralel with the decoding process and present a more efficient execution schedule for a logical qubit that reduces the cycle time t_{cycle}. We found that this execution schedule also relaxes the timing constraints on both the error syndrome measurements and the decoder. Furthermore, we showed that as a result of the reduced t_{cycle}, the decoding performance of such a logical qubit, can be reduced up to 70% when the time required for error correction equals the decoding time $t_{ec} = t_d$.

However, this approach does not scale well since it was based on a rule-based approach. Moreover, this approach is similar to other classical decoding algorithms like the Blossom algorithm. Blossom algorithm can reach good decoding performance but the execution time scales polynomially with the size of the system. Furthermore, for a given quantum system, Blossom scales polynomially with the physical error rate at which errors are generated. This is due to the graph matching approach that it uses to solve the decoding problem. To avoid this non-constant execution time with the physical error rate and still observe high decoding performance and scalability with the number of qubits, we decided to investigate neural network based decoders in Chapter 4.

We explored decoders that include neural networks, because it has been proven that neural networks exhibit constant execution time after being trained. This is also known as inference time, which means that regardless of which input is selected, the output of the neural network will be generated after the same time. Therefore for the same quantum system, the inference time will be constant, which is really valuable for decoding. In Chapter 4, we present our initial design for a neural network based decoder, which consists of a classical (a non-neural network) decoding module and a neural network. This design was enough to act as a proof of concept that such kind of neural network based decoder can successfully decode small distance surface codes for different error models. Our neural network based decoder reached equivalent performance to the Blossom decoder and showed good levels of generalization. For the code capacity error model, Blossom and neural network based decoder reached exactly the same performance due to the simplicity of the error model and the small code distances that were tested. For the depolarizing error model, the neural network based decoder outperformed Blossom from 2% to 38% for various physical error rates for all code distances tested. For the case of the noisy error syndrome measurements, we only tested for d=3, and the improvement achieved by the neural network based decoder ranged from 4% to 43%. The generalization was shown through comparison with a Look-Up Table (LUT) that consisted of all the training samples and was significantly under-performing compared to the neural network based decoder. For small code distances like d=3, most and in some cases all of the state space is used as training samples, therefore there is not any difference between the decoding performance of the neural network based decoder and the LUT. However, as the code distance increases and a small fraction of the state space is being used as training samples, the neural network based decoder significantly outperforms the LUT decoder. We reach improvement up to 99%. Moreover, we present a theoretical estimation of the execution time of such a decoder in a hardware implementation. We argue that a hardware implementation would be fast enough to decode based on the given time budget for error correction. Based on the structure of the neural network that was used, we upper bound the execution time to ~3.6-18μsec for a d=3 rotated surface code. We reach that result by using a high-level synthesis tool which permits hardware-synthesizable code to be generated from C [49], but does not ensure that the resulted code is optimized for speed. The available time budget for quantum error correction is dependent to the quantum technology and the fidelity of the quantum operations, however this result seems to be in the order of magnitude that most quantum technologies require.

Following the work of Chapter 4, there were many other implementation designs proposed by various research groups. All of the proposed designs can be categorized in two distinct categories: i) decoders that only include neural networks and predict corrections directly at the physical level and ii) decoders that include a classical module working alongside a neural network and predict corrections at the logical level. Most of these approaches are focused in reaching the highest decoding performance possible, but in our research we focus on creating a decoder with the smallest possible execution time while reaching at least equivalent decoding performance to Blossom. In Chapter 5, we compare both approaches and argue about the advantages and disadvantages of each design. We analyze the details of each approach and compare them in terms of execution time and decoding performance. Also, we provide an analysis about the tuning of the neural network parameters and provide a set of guidelines that can be used when such neural network based decoders need to be created. We select the best design out of the ones compared and test it against Blossom algorithm for the depolarizing error model with noiseless error syndromes. We reach an improvement between 2% and 37% compared to Blossom, in terms of decoding performance. We also show efficient decoding for the circuit noise model with noisy error syndrome measurements. Since the main goal of this research is to have high speed decoders, we avoid designs that use only a neural network module to act as the decoder, due to their need to repeat the prediction step. This repetition is affecting the execution time of the decoder in a polynomial way as the physical error rate increases.

The main challenge of neural network based decoders is the ability to scale efficiently as the code distance scales. In Chapter 6, we present a distributed way of decoding, that attempts to solve this issue. We propose the division of the error correction code into small areas, where a neural network based decoder performs the decoding in a local way. Error information is obtained from each small area, which is used by the neural network to identify whether the classical decoding module will provide the desired corrections. We show that there is no significant loss in the decoding performance compared to the design without the distributed decoding, while providing the same dataset to both decoders. Furthermore, we note that based on the construction of the classical decoding module, the introduction of an additional neural network can be beneficial. The added neural network in our design operates as a controlling mechanism that decides how the neural networks will be used based on the input data. Our simulations have shown that the same decoding performance can be achieved through the distributed decoding approach, while providing the ability of increasing the amount of relevant training samples in the original neural network. Therefore, the same improvement between 2% and 37% compared to Blossom is achieved with the distributed decoding approach.

Finally, in Chapter 7, we provide conclusions about our research and give some outlook for future work.

2

BACKGROUND

This chapter presents the background information about the topics discussed in this thesis. In Section 2.1 we provide the quantum computing basics. Following that, in Section 2.2 we describe different aspects of quantum error correction. In Section 2.2.1, we introduce an overview of the QEC codes and in Section 2.2.2 we present the surface code, which is the QEC code that we used throughout this thesis. In Section 2.2.3, we explain how decoding is performed in the surface code and in Section 2.3 we describe the error models that were used in our work. Finally, since we developed decoders based on neural networks, in Section 2.4 we provide some background information about neural networks.

2.1. THE POWER OF QUANTUM COMPUTING

The basic unit of information in quantum mechanics is called quantum bit or qubit. Contrary to the classical bit that only exists in a binary state of 0 and 1, the qubit has the ability to exist in a ***superposition*** of these two basis states, $|0\rangle$ and $|1\rangle$, with the superposition state ψ given by:

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

where

$$\alpha, \beta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

The state of a single qubit can be described graphically as a vector in the Bloch sphere as depicted in Figure 2.1.

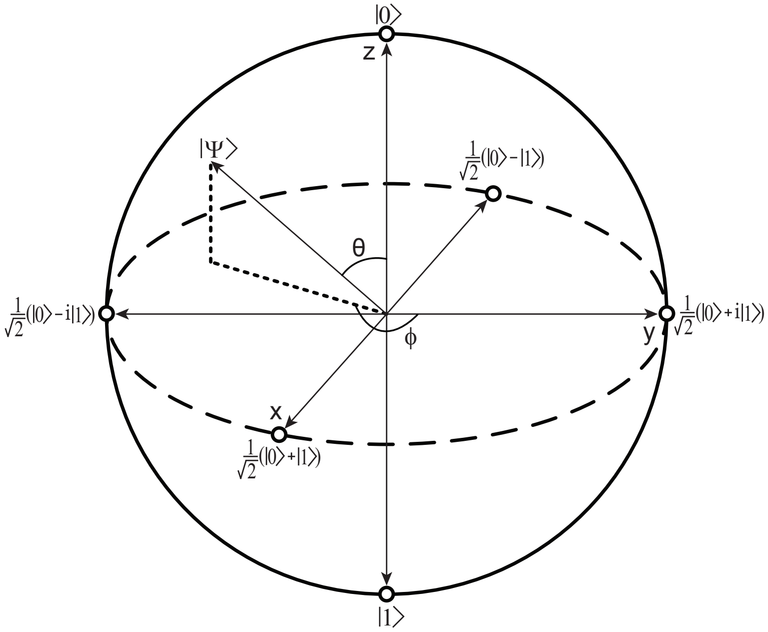

|0⟩
z
|Ψ⟩
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
θ
$\frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$
ϕ
y
$\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$
x
$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
|1⟩

Figure 2.1: Visualization of a single qubit state in the Bloch sphere

The Bloch sphere is a tool that visualizes the state of a single qubit. It is a unit sphere and the basis state $|0\rangle$ and $|1\rangle$ corresponds to the intersection of the sphere with the positive and negative z-axis, respectively. Any superposition state is described as a point on the sphere. Other common states besides the basis states are on the x-axis and the y-axis, which are the $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and the $\frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle)$, $\frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle)$, respectively.

A superposition state can be represented, by a complex-valued two-vector

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.3}$$

Any state in the Bloch sphere as depicted in Figure 2.1 can be mathematically given by:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle, \tag{2.4}$$

with φ, θ satisfying

$$\varphi \in [0, 2\pi) \quad \text{and} \quad \theta \in [0, \pi]. \tag{2.5}$$

In classical computers, readout of the bit by direct measurement will provide the bit value with 100% probability when the measurement process is error-free. In quantum computers, due to the superposition state of the qubits, the measurement process is non-deterministic. Readout of the qubit state will result in a probabilistic result based on the coefficients α and β according to eq. 2.1 and 2.2. That is, when a qubit is measured, a result "0" is obtained with probability $|\alpha|^2$ or a result "1" is obtained with probability $|\beta|^2$. Note that the measurement result is a binary value that cannot be used to reconstruct the original superposition state.

In classical computers, a system with n bits can represent 2^n possible states. This system can be in one of the 2^n states at a time. On the other hand, in quantum computers, a system with n qubits that exist in a superposition state can represent the system state as the tensor product of each qubit state:

$$\begin{aligned}|\psi\rangle = &(a_0 |0_0\rangle + b_0 |1_0\rangle) \otimes \\ &\cdots \otimes (a_{n-2} |0_{n-2}\rangle + b_{n-2} |1_{n-2}\rangle) \otimes (a_{n-1} |0_{n-1}\rangle + b_{n-1} |1_{n-1}\rangle)\end{aligned} \tag{2.6}$$

This state shown in eq. 2.6 is known as a product state since it consists of the product of all the independent 2^n qubit states.

If we expand the product state, we get a superposition of all states:

$$|\psi\rangle = \alpha_{0\cdots00} |0_{n-1} \cdots 0_1 0_0\rangle + \cdots + \alpha_{1\cdots10} |1_{n-1} \cdots 1_1 0_0\rangle + \alpha_{1\cdots11} |1_{n-1} \cdots 1_1 1_0\rangle, \tag{2.7}$$

where

$$\alpha_{i_{n-1}\cdots i_1 i_0} = \prod_{k=0}^{n-1} [(1 - i_k) \cdot a_k + i_k \cdot b_k], \quad i_k \in \{0,1\}, \tag{2.8}$$

It can be easily verified that the state $|\psi\rangle$ is also normalized:

$$\sum_{i_{n-1}=0}^{1} \cdots \sum_{i_1=0}^{1} \sum_{i_0=0}^{1} |\alpha_{i_{n-1}\cdots i_1 i_0}|^2 = 1. \tag{2.9}$$

The product state in eq. 2.6 can always be expanded to eq. 2.7, however the reverse is not always possible, especially when qubits are not independent of each other. This phenomenon is called ***entanglement*** and suggests that the state of one qubit cannot be described independently based on the state of the others. The immense power and parallelism that quantum computers exhibit is the outcome of the superposition and entanglement.

PERFORMING QUANTUM COMPUTATION

Quantum computation can be performed in various ways. The most common models are the measurement based quantum computing model [50], the adiabatic quantum computing model [51] and the topological quantum computing model [52] and the quantum circuit. The last one is the most popular one.

The ***quantum circuit model*** performs quantum computation as a series of reversible quantum gates. A quantum circuit can be described as a set of quantum operations that are going to be performed in a system of qubits. There are three procedures that need to be employed, namely preparation of an input quantum state in a computational basis, unitary evolution of the quantum state while gates are applied and measurement, which will probabilistically provide the output state [50].

In this thesis, we are using the quantum circuit model of computation.

SINGLE-QUBIT GATES

A ***single-qubit gate*** acts on a single qubit and alters its state. A single-qubit gate is represented as a rotation $R_{\hat{n}}(\theta)$ on the Bloch sphere along the axis $\hat{n} = (n_x, n_y, n_z)$ ($|\hat{n}|^2 = 1$) by an angle θ. The representation of this rotation by the unitary matrix is given by:

$$R_{\hat{n}}(\theta) = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right)(n_x X + n_y Y + n_z Z) \tag{2.10}$$

where I, X, Y and Z are the Pauli matrices that represent the Pauli gates

$$I \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \; Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.11}$$

The symbolic representation of these gates is given in Figure 2.2:

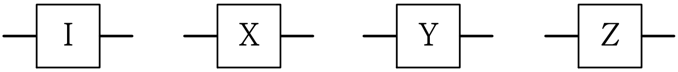


Figure 2.2: Symbolic representation of the Pauli matrices.

The effects of single-qubit Pauli gates to a qubit state are given below:

$$I: \begin{matrix} |0\rangle \mapsto |0\rangle \\ |1\rangle \mapsto |1\rangle \end{matrix}, \; X: \begin{matrix} |0\rangle \mapsto |1\rangle \\ |1\rangle \mapsto |0\rangle \end{matrix}, \; Z: \begin{matrix} |0\rangle \mapsto |0\rangle \\ |1\rangle \mapsto -|1\rangle \end{matrix}, \; Y: \begin{matrix} |0\rangle \mapsto i|1\rangle \\ |1\rangle \mapsto -i|0\rangle \end{matrix} \tag{2.12}$$

Other single-qubit gates that are commonly used in quantum computation are the Hadamard (H) gate, the T gate and the Phase (S) gate, which have the following matrix representation:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad T \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, \; S \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix} \tag{2.13}$$

The symbolic representation of these gates is given in Figure 2.3:

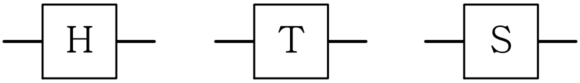


Figure 2.3: Symbolic representation of the Hadamard, T and Phase gate.

MULTI-QUBIT GATES

A ***multi-qubit gate*** acts on multiple qubits at the same time and alters their state. Since multi-qubit gates operate on more than one qubit, their effect cannot be represented in the Bloch sphere. The most common multi-qubit gates are the Controlled-NOT (CNOT) and the Controlled-Phase (CZ) gate, which both act on two qubits.

The CNOT gate is applied to two qubits at the same time. One qubit is the control qubit and the other is the target qubit, that are represented by the top and the bottom one, respectively in Figure 2.4. The state of the target qubit is going to be changed from $|0\rangle$ to $|1\rangle$ and vice-versa, if the state of the control qubit is $|1\rangle$, otherwise the state of the target qubit remains unchanged. This operation is a bit-flip (X) operation on the target qubit state conditioned on the state of the control qubit.

In case of the CZ gate, the state of the control qubit is going to be changed from $|1\rangle$ to $-|1\rangle$, if the state of the target qubit is $|1\rangle$, otherwise the state of the control qubit remains unchanged. This operation is a phase-flip (Z) operation on the control qubit state conditioned on the state of the target qubit.

They then perform the following transformations:

$$CNOT: \begin{array}{c} |00\rangle \rightarrow |00\rangle \\ |01\rangle \rightarrow |01\rangle \\ |10\rangle \rightarrow |11\rangle \\ |11\rangle \rightarrow |10\rangle \end{array}, \quad CZ: \begin{array}{c} |00\rangle \rightarrow |00\rangle \\ |01\rangle \rightarrow |01\rangle \\ |10\rangle \rightarrow |10\rangle \\ |11\rangle \rightarrow -|11\rangle \end{array} \tag{2.14}$$

The corresponding matrix representation is given by:

$$\text{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{CZ} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Multi-qubit gates are essential to create entanglement in the system, since they can map certain product states onto entangled states.

As another example of a multi-qubit gate, we present the Toffoli gate, also known as the Controlled-Controlled-NOT (CCNOT) gate. The Toffoli gate is performed into three qubits and applies a bit-flip operation (X) on the target qubit, if the two control qubits have a value of $|1\rangle$.

The matrix representation of the Toffoli gate is given by:

$$\text{Toffoli} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.15}$$

The circuit representation of the most common multi-qubit gates, the CNOT, the CZ and the Toffoli gate, is shown in Figure 2.4.

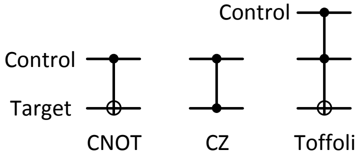


Figure 2.4: Quantum circuit representation of the CNOT, the CZ and the Toffoli gate.

In direct analogy to the classical case, a ***universal quantum gate set*** can be found that includes a set of quantum gates that can be used to perform any quantum operation. More accurately, this gate set should be able to approximate any m-qubit operation U_m to arbitrary precision, where $m \geq 1$. The most commonly-used universal quantum gate set is $\{H, T, CNOT\}$.

QUANTUM CIRCUIT

As we mentioned, in the quantum circuit model of computation, a quantum circuit consists of a sequence of reversible quantum gates that are applied on the qubits. In a quantum circuit, qubits are represented with horizontal lines and quantum operations are represented with blocks. Figure 2.5 shows the application of a single-qubit gate U_s to a qubit q_0 and Figure 2.6 shows the application of a multi-qubit gate U_m to m qubits. If the initial state before the application of the gate is $|\psi\rangle$, then the output can be calculated as $|\psi'\rangle = U_s |\psi\rangle$ and $|\psi'\rangle = U_m |\psi\rangle$, respectively.

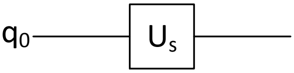


Figure 2.5: Quantum circuit describing the application of a single-qubit gate U_s to qubit q_0.

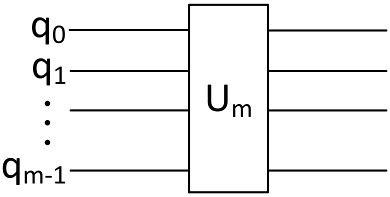


Figure 2.6: Quantum circuit describing the application of a multi-qubit gate U_m to m qubits.

2.2. FAULT TOLERANT COMPUTATION AND QUANTUM ERROR CORRECTION

As mentioned in Section 1.2, quantum error correction is required to correct errors that accumulate in the quantum system as it evolves. However, encoding physical qubits to logical qubits and decoding in order to find the errors are not sufficient on their own to allow reliable quantum computation and storage.

Due to the extra resources that quantum error correction introduces, more faulty operations are inserted. Furthermore, propagation of errors as quantum gates are applied is inevitable. Therefore, we need another mechanism that does not allow errors to spread uncontrollably in ways that we cannot predict. If that is the case, then decoding will fail to identify errors and preserve the system in the desired quantum state.

The procedure of performing reliable computation and storage while errors are present as the system is evolving, is called fault tolerant quantum computing. A quantum operation is assumed to be fault tolerant if only one component in a certain operation (e.g. state preparation, measurement, application of a gate, idling) fails, then the failure will cause at most one error in each encoded block of qubits at the output of the operation [42]. Fault tolerant operations and syndrome extraction circuits included in QEC are required, to avoid potential propagation of errors that will quickly lead to a failure of the quantum computation.

Nevertheless, to achieve fault tolerant quantum computation the error rate at which errors are inserted in the quantum system, must be below a certain threshold. The threshold theorem states that a quantum computer in the presence of noise can perform close to ideal quantum computation, if the failing probability for the noisy system is less than a certain threshold [42, 53]. The threshold value is dependent on the quantum error correcting code, the error model and the decoder selected. The highest threshold reached so far is 1% for the surface code assuming faulty operations [2].

2.2.1. QUANTUM ERROR CORRECTING CODES

In order to protect quantum information against errors, a quantum error correcting code needs to be used. Many families of QECCs have been developed providing different advantages, such as simple code structure and transversal implementation of gates. Many of the codes developed belong to more than one family, in an effort to gain advantages from multiple families of QECCs. A non-exhaustive representation of the most popular QECCs and the families they belong to, can be seen in Figure 2.7. QECCs differ in the way that the quantum information is represented in the state space of many physical qubits, the way that fault-tolerant gates are performed and the way that the error information through quantum measurements is obtained.

The codes found in Figure 2.7 are namely: Calderbank-Shor-Steane (CSS) code [12, 54], Shor code [11], low-density parity-check (LDPC) code [55], Bacon-Shor code [56], 2D color code [57], 3D color code [58], Steane code [12–14], surface code [59, 60], subsystem surface code [61], gauge color code [62], code deformation in toric code [1] and condensed matter systems [60].

Although these QECCs differ in many aspects, there are some principles that need to be followed by any QECC. The encoding procedure should be simple and effective, making the code fault tolerant to avoid fast accumulation of errors. More accurately, the code should be able to operate below a certain threshold, so that the extra overhead in resources added by QEC will increase the decoding performance of the code. Moreover, the code should be defined by its operators which

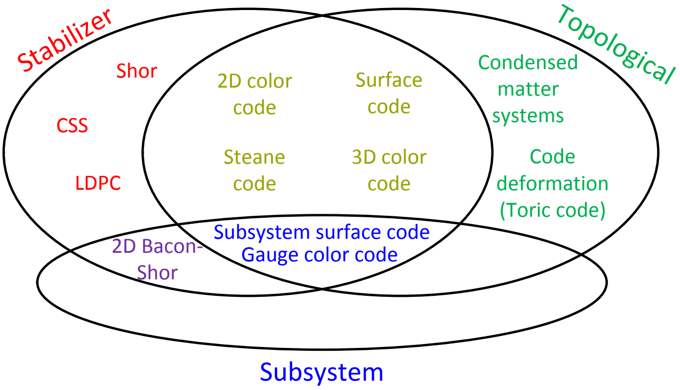


Figure 2.7: Popular quantum error correcting codes and their respective families

indicate how the encoded state changes and any geometrical constraints should be provided to the decoder to assist with the decoding process. Finally, the code should be easily abstracted to dimensions higher than 2.

2.2.2. SURFACE CODE

Many quantum error correcting codes have been proposed, but throughout this thesis we only consider the surface code [1, 2, 63–69], one of the most promising QEC codes. The surface code is a Topological Stabilizer code that has a simple structure, local interactions between qubits and is proven to have high tolerance against errors. It is usually defined over a 2D lattice of qubits [69, 70], although higher dimensions can be used.

Topological codes distribute the qubits in a certain topology and exploit the local interactions between qubits. The greatest advantage of topological codes is the simple and scalable geometry in which qubits are allocated and the high error threshold exhibited by this family of codes. Stabilizer codes are based on the stabilizer formalism and use stabilizer operators (parity-checks) to perform error detection and correction. The main advantage is that they provide a clear and simple framework which can be exploited by various quantum error correcting codes. The surface code inherits the advantages out of both families of codes, thereby making it a promising solution as a quantum error correcting code.

The surface code consists of two types of physical qubits, ones that store quantum information, known as data qubits, and others that can be used to detect errors in the logical qubit through their measurement, known as ancillary or ancilla qubits. Since quantum errors are continuous, we need to discretize them to make the identification process easier. We discretize quantum errors into Pauli bit-flip ($\hat{X}$) errors, phase-flip ($\hat{Z}$) errors and bit- and phase-flip errors ($\hat{Y}$). Based on that, the surface code only needs to identify and correct these two types of errors. Therefore, only X-type and Z-type ancilla qubits are required, which identify phase-flip and bit-flip errors, respectively. An example of a 2D surface code is presented in Figure 2.8.

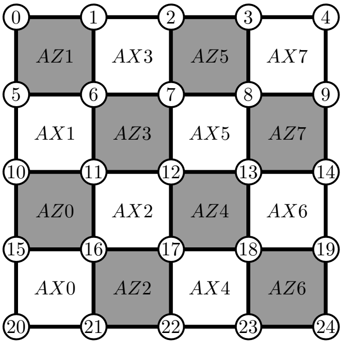


Figure 2.8: Surface code structure. Data qubits are placed in the corners, X-type ancilla qubits are placed inside the white squares and Z-type ancilla are placed inside the grey squares.

In order to detect both type of errors, each ancilla qubit is entangled with its neighboring data qubits based on the circuit presented in Figure 2.9. This circuit is used to collect the ancilla measurements for the surface code and therefore is known as the error syndrome measurement (ESM) circuit. It signifies a surface code cycle which includes the preparation of each ancilla in the appropriate state, followed by 4 CNOT gates that entangle each ancilla qubit with its 4 neighboring data qubits. ESM ends with the measurement of all ancilla qubits in the appropriate basis. In principle, all ancilla measurements in the lattice can be performed in parallel, therefore obtaining the error information for the whole lattice in one surface code cycle.

The measurement result of the ancilla is a parity-check, which is a value that is calculated as the parity between the state of the data qubits connected to it. Note that the parity-checks are used to identify errors in the data qubits without having to measure the data qubits explicitly and collapse their state. The state of the ancilla qubit at the end of every parity-check is collapsed through the ancilla measurement, but is initialized once more in the beginning of the next error correction cycle. The collection of parity-checks out of one or multiple error correction cycles is known as the error syndrome. The error syndrome is provided to the decoder to identify the location and type of errors and propose corrections that erase these errors. Note that, the parity-checks must commute with each other, anti-commute with errors and commute with the logical operators.

A critical issue when gathering the error syndrome, is the order in which we perform the CNOT gates in the ESM circuit presented in Figure 2.9. As described in [3], a correctable error might be generated, but due to the order of the CNOTs, it can propagate to more errors, leading to the creation of a logical error. There exist a schedule of the CNOT gates in the ESM circuit that will alleviate this issue, which visually looks like an "S" or a "Z" shape in terms of the order of the CNOTs on the data qubits. By following that schedule, no correctable errors will propagate catastrophically to the rest of the qubits, so a logical error will be avoided, thus making the circuit fault-tolerant.

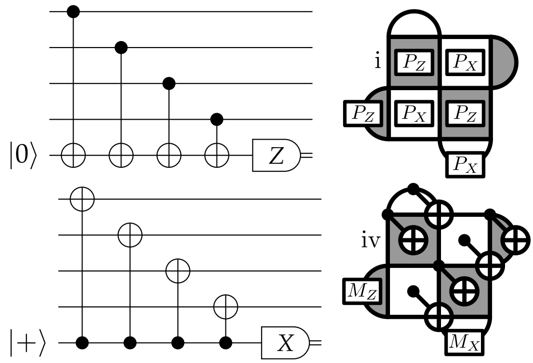
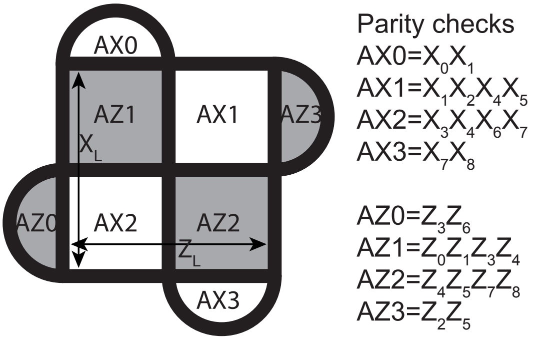


Figure 2.9: Error syndrome measurement circuit for the distance-3 rotated surface code [1–3]. Left: Measurement circuit for individual Z tiles (top) and X tiles (bottom), including an ancilla qubit to be placed at the center of each tile as seen at the right side. Ancilla qubits are prepared in the $+1$-eigenstate of the appropriate basis, four CNOT gates are executed, and the ancilla qubits are measured in the appropriate basis. Right: Interleaving of separate parity check measurements, including late preparation and early measurement for weight-two parity checks.

There are two main ways that encoding is performed in the surface code, known as planar encoding [60] and encoding with defects [63], however, in this thesis we focus only on planar encoding. Planar surface code requires less number of qubits compared to the defect for the same level of protection [63]. Therefore, planar surface code seems to be a better approach for short-term experimental platforms [40, 71]. One of the smallest surface codes that constitutes a logical qubit is presented in Figure 2.10. This is the rotated surface code that was introduced in [72]. It consists of 9 data qubits placed at the corners of the square tiles and 8 ancilla qubits placed inside the square and semi-circle tiles. Each ancilla qubit can interact with its neighboring 4 (square tile) or 2 (semi-circle tile) data qubits, which is shown through the parity-checks of Figure 2.10.



Parity checks

$AX0=X_0X_1$

$AX1=X_1X_2X_4X_5$

$AX2=X_3X_4X_6X_7$

$AX3=X_7X_8$

$AZ0=Z_3Z_6$

$AZ1=Z_0Z_1Z_3Z_4$

$AZ2=Z_4Z_5Z_7Z_8$

$AZ3=Z_2Z_5$

Figure 2.10: Rotated surface code with code distance 3. Data qubits are placed at the corners of the tiles and are enumerated from 0 to 8 (D0-D8). X-type ancilla are placed in the center of the white tiles and Z-type ancilla are placed in the center of grey tiles.

A logical qubit in the surface code is defined by its logical operators ($\bar{X}$,$\bar{Z}$) that determine how the logical state of the qubit can be changed. Any operator of the form $X^{\otimes n}$ or $Z^{\otimes n}$ that creates a chain that span both boundaries of the same type can be regarded as a logical operator, with n being the number of data qubits that are included in the logical operator. Typically, the logical operator with the smallest n is selected.

An important feature of any QECC is the code distance. Code distance (d), describes the degree of protection against errors. Distance can be regarded as the minimum weight of a Pauli operator commuting with all parity-checks and acting non trivially on the system containing the quantum information. More accurately, it is the minimum number of physical operations required to change the logical state [44, 73]. In surface code, the degree of errors (d.o.e.) that can be successfully corrected, is related to the code distance and is calculated according to the following equation:

$$\text{d.o.e.} = \lfloor \frac{d-1}{2} \rfloor \tag{2.16}$$

Therefore, for the d=3 surface code of Figure 2.10, only single errors can be successfully corrected.

Surface Code is a very promising quantum error correcting code with many advantages. It has a simple structure with local constraints between qubits, therefore requiring only nearest neighbor interactions. Most of the gates can be performed transversally (theoretically) in planar encoding and in general gates do not have much spatial overhead. Furthermore, the dominant gate is the CNOT or equivalently the CZ, which has experimentally been implemented with a fidelity of 99%. Moreover, the highest known threshold (1%) has been achieved with the surface code.

However, there are also some limitations. It has up to 4 parity check operators per qubit, therefore requiring more resources compared to other QECCs. Also, due to the increased number of qubits, it produces a large volume of error information that requires fast decoding. Another limitation is that it requires a distillation process [63], in order to have a universal set of gates.

2.2.3. DECODING THE SURFACE CODE

As we just mentioned, after running a surface code cycle, the parity-checks will be forwarded to the decoding algorithm. These binary values will be combined in order to identify the location and type of errors in the code. In the rotated surface code, each ancilla performs a parity-check of the form of $X^{\otimes 4}/Z^{\otimes 4}$ (square tile) and $X^{\otimes 2}/Z^{\otimes 2}$ (semi-circle tile), as presented in Figure 2.9. When the state of the data qubits involved in a parity-check has not changed, then the parity-check will return the same value as in the previous surface code cycle. In the case where the state of an odd number of data qubits involved in a parity-check is changed compared to the previous surface code cycle, the parity-check will return a different value than the one of the previous cycle ($0 \leftrightarrow 1$). The change in a parity-check in consecutive cycles is known as a detection event.

Assuming no ancilla errors, a single data qubit error will cause two neighboring parity-checks to indicate two detection events (Z error in the bottom of the lattice in Figure 2.11), unless the error occurs at the corner of the lattice which will lead to only one parity-check indicating one detection event (Z error in the top corner of the lattice in Figure 2.11). Multiple data qubit errors that occur near each other form chains of errors (X errors in Figure 2.11), which causes only two detection events located at the parity-checks existing at the endpoints of the error chain [1, 63, 73].

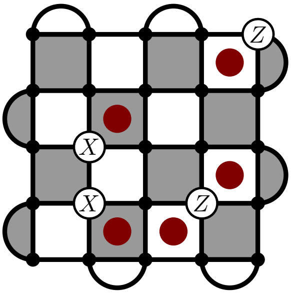


Figure 2.11: Rotated surface code with code distance 5. Errors are shown with X or Z on top of the data qubits and detection events that correspond to these errors are shown with red dots.

In the case that the measurement process is imperfect, a different type of errors will be present. A measurement during the readout of the ancilla value is known as a measurement error. When a measurement readout is misinterpreted, the decoder will get information about an error that does not exist, since it was just the misinterpretation of the ancilla value. In that case, a correction might be applied where no error existed and vice-versa. The way that a measurement error is observed is by comparing the measurement values of multiple consecutive surface code cycles for the same parity-check, as presented in Figure 2.12.

In the case where the error probability for a data qubit error is equal to the error probability for a measurement error, d surface code cycles are deemed enough to successfully identify measurement errors [74]. When a measurement error is successfully identified, no correction is required.

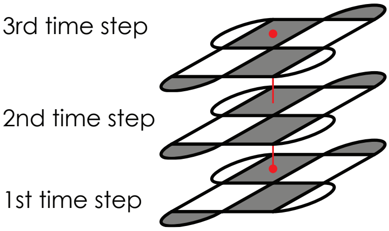


Figure 2.12: Rotated surface code with code distance 3 at consecutive time steps. The alternating pattern on the measurement value of the same parity-check, indicates the presence of a measurement error.

Thus, through observation of the parity-checks throughout multiple surface code cycles, identification of errors is made in space (data errors) and in time (measurement errors). The decoder receives the error syndrome out of multiple surface code cycles and analyzes the detection events. Then, it proposes a set of corrections that will cancel out the errors that have accumulated.

However, totally suppressing the noise is unfeasible, since the decoder might misinterpret the information coming from the error syndrome. The main reason for such misinterpretations, comes from the fact that the surface code is a degenerate code. This degeneracy means that different sets of errors create the same error syndrome. Therefore, based on the physical error rate of the quantum operations, different sets of errors are more likely than others. This puts an extra assumption to the decoder, since it should output different corrections based on the error probability. Based on all these reasons, it is evident that no decoder can perfectly suppress all noise.

Evaluating the decoding performance is done by calculating the logical error rate for a large range of physical error rates. The logical error rate is calculated as the ratio of logical errors created over the number of surface code cycles run. Plotting a graph containing the physical error rates and the corresponding logical error rates, provides a clear picture about the decoding performance, as seen in Figure 2.13.

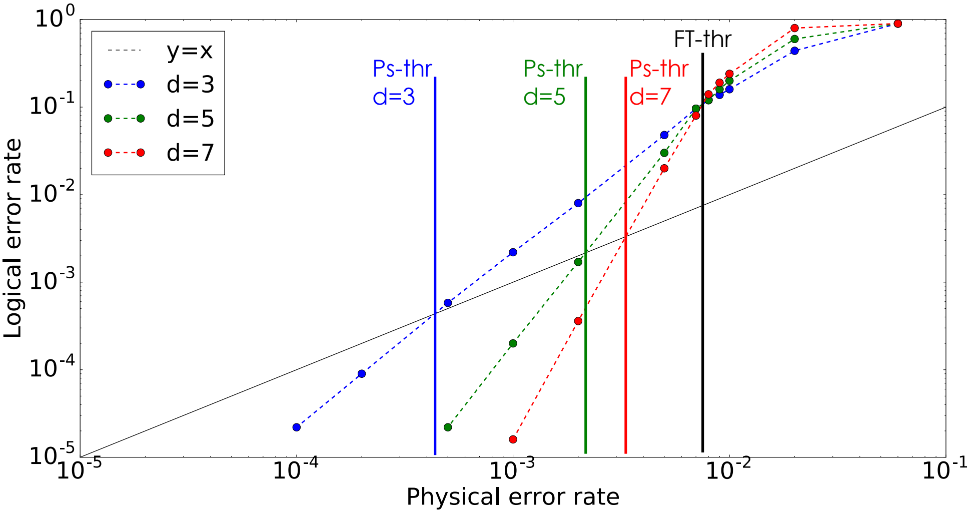


Figure 2.13: Decoding performance indicating the threshold of the surface code and the pseudo-thresholds of each code distance.

The decoding performance of a decoder as well as the level of protection of a QEC code for a given error model can be characterized by the metric known as ***threshold***. Threshold is defined as the point of intersection between all the curves representing the logical error rates of multiple code distances and constitutes the highest physical error rate at which the extra resources of quantum error correction are aiding the error identification process, see FT-th dashed line in Figure 2.13.

In Figure 2.13, we also point out the ***pseudo-threshold*** values for d=3,5 and 7 with the Ps-th d=3, Ps-th d=5 and Ps-th d=7 lines, respectively. The pseudo-threshold is defined as the highest physical error rate that the quantum device should operate, in order for error correction to be beneficial for the given code distance. Operating at higher than the pseudo-threshold probabilities will cause worse decoding performance, due to the extra faulty operations inserted by QEC. Pseudo-threshold is defined as the point of intersection between the curve consisting of the logical error rates for a given code distance with the curve where the physical error rate is equal to the logical error rate ($y = x$).

Note that both the threshold and the pseudo-threshold depend on the QECC, the decoding strategy and the error model. In order to obtain a physical error rate versus logical error rate graph, as shown in Figure 2.13, several surface code cycles are performed in which errors are inserted based on a specified error model. In Section 2.3, we will describe the different error models that we used in the simulations performed for this thesis.

As we mentioned in Section 1.3, there are certain challenges in decoding such as the limited decoding time, the decoding performance and the scalability of the decoder.

2.2.4. QUANTUM ERROR DECODERS

As we just explained, the decoding process is about identification of the quantum errors. The main parameters that define a "good" decoder are the high decoding performance, the ability to efficiently scale to large code distances and the small execution time.

There already exist classical decoders that can reach good decoding performance, enough to make fault-tolerant quantum computing possible. Some of the popular classical decoding algorithms are the ***maximum-likelihood algorithm*** [75], the ***Blossom algorithm*** [76–78], the ***Markov Chain Monte Carlo*** (MCMC) [79, 80] and the ***Renormalization Group (RG)*** algorithm [81, 82].

The maximum-likelihood algorithm investigates the most probable error that has occurred that produces the observed error syndrome. This process can reach high decoding accuracy but is extremely time consuming especially as the code distance increases. The execution time scales as $O(n\chi^3)$, with χ being an approximation parameter, as given in [75].

The Blossom algorithm is a graph matching algorithm that is searching for the least number of errors that can produce the observed error syndrome. It creates a graph containing the detection events that were observed and performs a Minimum Weight Perfect Matching (MWPM), in order to find the minimum set of corrections [83]. Blossom can reach slightly lower decoding performance than the maximum-likelihood decoder, but still good enough to be used in experiments. The execution time scales linearly with the number of qubits [84], but still might not meet the small execution time requirements of contemporary experiments. However, there exist an optimized version of the Blossom algorithm that claims a constant average processing time per detection round, which requires dedicated hardware [78].

The Markov Chain Monte Carlo method is based on the selective application of a stabilizer in the lattice, that will correct an error with a certain probability. It is based on the approximation of the Metropolis algorithm [85]. MCMC starts from a certain initial minimum weight error configuration by selecting a random stabilizer and observing the hypothetical changes in the code as if the selected stabilizer was hypothetically applied. Based on some probability, this stabilizer will be selectively applied [79]. The MCMC decoder achieves high decoding performance, by ensuring that all errors appearing in such a Markov chain belong to the same class of errors and are compatible with the same error pattern. However, this decoding approach leads to a super-polynomial running time complexity [79].

Renormalization Group decoding provides a good solution for the decoding of large quantum systems, because decoding is performed in a local manner through distributed regions throughout the lattice. The RG algorithm can be highly parallelized and the scaling is reported to be $log(l)$, for an lxl code [82]. Nevertheless, the decoding performance reported so far is not as high as the one of other classical algorithms.

Summarizing, some decoders can achieve high decoding performance but are not scalable, while others can not reach equally good decoding performance but are fast. Then, the main challenge is to have a decoder that can reach high decoding performance while requiring small execution time. A comparison in terms of the decoding performance and the running time complexity is summarized in Table 2.1. We should mention that the decoding performance is quantified as the threshold value achieved for toric or surface code for the depolarizing error model with noiseless error syndrome measurements and the running time complexity is the one reported in the initial implementation of these decoders. Note that approximate implementations exist that report improvements compared to the reported numbers of Table 2.1.

Table 2.1: Comparison between decoders

Decoder	Decoding performance (Threshold)	Running time complexity
MLD	0.18 [75]	O(nχ^3)
Blossom	0.148 [83]	O(n)
MCMC	0.17 [79]	O(L^4)
RG	0.164 [81]	O(log n)

where χ is an approximation parameter, n is the number of qubits and L is the code distance.

In most current quantum technologies, the time budget for error correction and decoding is small, due to the erroneous nature of the qubits and the imperfect application of quantum operations. Therefore, a high speed version of an efficient decoder is required. In an attempt to develop such kind of a decoder, researchers came up with neural network based decoders. It was quickly proven that such kind of decoders that incorporate neural networks, have a constant execution time after being trained and that can reach equivalent decoding performance to most classical decoders.

A large variety of neural network based decoders has been recently proposed [86–96], making them a potential candidate for decoding. The main challenge of these decoders is the ability to scale to large code distances, without significant loss in the decoding performance. This issue arises from the fact that as the code distance increases, the dataset required to train the algorithm is increasing exponentially. This issue is going to be tackled in Chapter 6.

2.3. QUANTUM ERRORS

Due to the decoherence, the imperfect application of quantum operations and other sources of errors, the quantum state is altered, which we describe as errors being inserted in the quantum system. As we mentioned in Section 1.2, in order to identify these continuous errors we perform a discretization process that deduces every error into Pauli errors. We model these errors by using error models that assign an error probability to each quantum operation. In that way, the classical computer that performs the error detection can identify such Pauli errors. When errors are identified, a set of corrections is proposed by the decoder to erase them. This type of error correction is known as active, because we are taking action against the errors that were observed in the quantum system.

However, there is another form of error correction known as error mitigation, which as the name suggests attempts to minimize or even completely suppress these error mechanisms, so that no errors are generated. Techniques such as decoherence free subspaces [97], composite pulse sequences [98], error extrapolation [99], quasi-probability decomposition [100] and many more are used to avoid many of the error inducing mechanisms.

2.3.1. ERROR PROPAGATION AND TRANSFORMATION

It is important to study how the propagation of Pauli errors occurs after the application of relevant gates. In this thesis, we only consider quantum memories with logical qubits encoded with the surface code. Therefore, the only operations required are the ones in the ESM circuit (see Figure 2.9) and the Pauli gates used as correction gates to the Pauli errors. All gates included in the ESM circuit map Pauli errors to Pauli errors, which means that the propagation of errors through the gates is deterministic. Also, the transformation of an error into a different type of error, while going through a gate, is completely deterministic, therefore we can keep track and predict the propagation and transformation of errors throughout the surface code cycles.

The only gates in the ESM circuit are the Hadamard (H) and the CNOT gate. We present the propagation relations, while skipping i) the $\hat{Y}$ errors, since $\hat{Y} = i\hat{X}\hat{Z}$, and ii) the rest of the combinations for the CNOT, since they are easily computed from these four.

(i) HX = ZH

(ii) HZ = XH

(v) CNOT(I⊗X) = (I⊗X)CNOT

(vi) $\mathrm{CNOT}(X \otimes I) = (X \otimes X)\mathrm{CNOT}$

(vii) $\mathrm{CNOT}(I \otimes Z) = (Z \otimes Z)\mathrm{CNOT}$

(viii) $\mathrm{CNOT}(Z \otimes I) = (Z \otimes I)\mathrm{CNOT}$

2.3.2. SIMULATED ERROR MODELS

In order to define how errors are introduced, different error models are described that model quantum noise in different ways. In our research we used three different error models, in order to obtain a more complete picture of the decoding performance of the decoder. Note that as already mentioned, the decoding performance depends on the error model.

In this section, we will analyze the three main error models that we used, namely: the independent X and Z, the depolarizing and the circuit noise model.

INDEPENDENT X AND Z ERROR MODEL

The independent X and Z model inserts independently bit-flip ($\hat{X}$) and phase-flip ($\hat{Z}$) errors on qubits, with the same error probability $p = p_X = p_Z$. Therefore, the probability for simultaneous bit- and phase-flip ($\hat{Y}$) error is the product of these probabilities $p_Y = p_X^2 = p_Z^2$.

Due to the errors occurring independently, we can keep track of each type of errors separately. Furthermore, since the error probabilities for bit-flip and phase-flip are equal, the expected decoding performance, one can take into consideration only one of them and assume to have similar distribution of errors for the other. Since bit-flips and phase-flips occur independently, the decoder can create two separate cases for each type of error, making the decoding problem simpler. In this case, the decoding performance is expected to be almost identical for bit-flips and phase-flips.

The independent $\hat{X}$ and $\hat{Z}$ model cannot be considered a realistic error model, however, it is used due to its simple structure and limited assumptions. As an example, we refer to the Surface code, for which the correction of phase-flips can be considered equivalent to correcting bit-flips after rotating the lattice $\frac{\pi}{2}$ from its initial position [75]. We do not expect that the errors in the real quantum system will be independent with equal probabilities for bit-flip and phase-flip errors, rather have highly correlated errors occurring with some bias based on the quantum technology.

DEPOLARIZING ERROR MODEL

The depolarizing model is one of the most frequently used error models mainly due to its simple construction. There are two variants of this model, known as the symmetric and asymmetric depolarizing error model. The symmetric depolarizing error model inserts bit-flip ($\hat{X}$), phase-flip ($\hat{Z}$) and both bit- and phase-flip ($\hat{Y}$) errors with equal probability $p = p_X/3 = p_Z/3 = p_Y/3$. The asymmetric depolarizing error model inserts errors with different probabilities for each type of Pauli error $p_X/3 \neq p_Z/3 \neq p_Y/3$ [42]. This error model provides some correlation between bit-flip and phase-flip errors, but still cannot be considered a realistic error model.

CIRCUIT ERROR MODEL

The circuit error model is still a theoretical model, but is considered the most realistic one compared to the other two. It assumes that all operations that occur are faulty. Therefore, based on the type of quantum operation, noise is modelled in a different way. In our work, we focused on quantum memories with qubits encoded based on the surface code. The circuit that performs the error syndrome measurements in the surface code consists of state preparation, measurement, Hadamard and CNOT gates.

The state preparation fails with probability p and results in the wrong prepared state. Measurement fails also with probability p and results in the wrong report of the parity-check. Errors on qubits are usually modelled based on the symmetric depolarizing error model, therefore each type of Pauli error (p_X, p_Z, p_Y) fails with the same probability $p/3$. The Hadamard gate inserts errors in the same way as they are inserted on the qubits. The CNOT gate fails for each of the fifteen potential erroneous cases (IX, ZZ, XY, etc) with probability $p/15$.

In all three error models, noise is simulated by application of a perfect quantum operation, followed by an error channel that inserts errors based on the described error probabilities.

In all of the aforementioned error models, the process of error syndrome measurement might be considered perfect or imperfect. During perfect error syndrome measurement only a single error correction cycle is required, whereas during imperfect error syndrome measurement multiple error correction cycle are required. In this thesis, we assume that the probability of a data qubit error is equal to the probability of a measurement error, therefore d cycles of error correction are sufficient to identify all types of errors.

2.4. NEURAL NETWORKS

Since we developed a neural network based decoder, we present some background information on neural networks in this section.

Artificial neural networks have been shown to reach high application performance and constant execution time after being trained on a set of data generated by the application. An artificial neural network is a collection of weighted interconnected nodes that can transmit signals to each other. The receiving node processes the incoming signal and sends the processed result to its connected node(s). The processing at each node is different based on the different neural network parameters being used. A common representation of neural network is presented in Figure 2.14.

There are 3 types of layers in a neural network, namely the input layer, the hidden layer and the output layer. The input and output layer can only be one layer, whereas all the in-between layers are assumed as hidden layers. The nodes of each layer are connected to the nodes of the next layer and every node performs a computation of a non-linear function that will define the contribution of the node to the output of the neural network. All connections between nodes are weighted connections that define the amount of contribution of each node to each connected node at the following layer. The input of every node (inp) is calculated by the

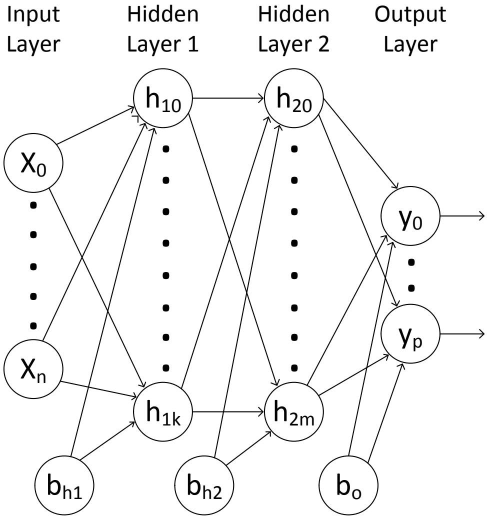


Figure 2.14: Representation of an artificial neural network with 4 layers

following equation:

$$\text{inp} = \sigma(\sum w_i * v_j + b_k) \tag{2.17}$$

where w_i are the weights at each connection, v_j is the output value of the incoming connection out of a node and b_k is the bias of the layer. σ is the non-linear activation function, which is necessary in order to be able to map (non)-linear inputs to non-linear outputs. σ denotes the activation function that is selected, with popular options being the sigmoid, the hyperbolic tangent (tanh) and the rectified linear unit (ReLU).

In this work, we focus on two types of neural networks known as ***Feed-forward neural networks (FFNN)*** and ***Recurrent neural networks (RNN)***. Feed-forward neural networks are considered to be the simplest type of neural network, allowing information to move strictly from input to output, whereas recurrent neural networks are considered to be more sophisticated, including feedback loops. The simple construction of FFNNs makes them extremely fast in applications, however, RNNs are able to produce better results for more complex problems.

In Feed-forward neural networks, input signals x_i are passed to the nodes of the hidden layer h_i and the output of each node in the hidden layer acts as an input to the nodes of the following hidden layer, until the output of the nodes of the last hidden layer is passed to the nodes of the output layer y_i. The weighted connections between nodes of different layers are denoted as W_i and b is the bias of each layer.

In recurrent neural networks there is feedback that takes into account output from previous time steps y_{t-1}, h_{t-1} (see Figure 2.15). RNNs have a feedback loop at every node, which allows information to move in both directions. Due to the feedback nature, recurrent neural networks can identify temporal patterns of widely separated events in noisy input streams.

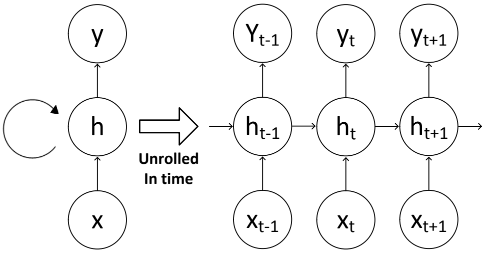

y
h
x
Unrolled
In time
y_{t-1}
y_t
y_{t+1}
h_{t-1}
h_t
h_{t+1}
x_{t-1}
x_t
x_{t+1}

Figure 2.15: A conceptual visualization of the recurrent nature of an RNN.

In this work, Long Short-Term Memory (LSTM) cells are used as the nodes of recurrent neural networks (see Figure 2.16). In an LSTM cell there are extra gates, namely the input, forget and output gate that are used in order to decide which signals are going to be forwarded to another node. W is the recurrent connection between the previous hidden layer and current hidden layer. U is the weight matrix that connects the inputs to the hidden layer. $\widetilde{C}$ is a candidate hidden state that is computed based on the current input and the previous hidden state. C is the internal memory of the unit, which is a combination of the previous memory, multiplied by the forget gate, and the newly computed hidden state, multiplied by the input gate [101].

The equations that describe the behaviour of all gates in the LSTM cell are described in Figure 2.16.

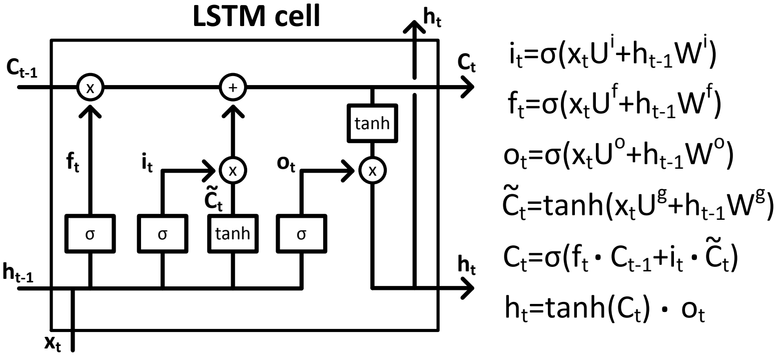

LSTM cell
h_t
C_{t-1}
C_t
x
+
tanh
f_t
i_t
x
o_t
x
$\tilde{C}_t$
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
$i_t=\sigma(x_tU^i+h_{t-1}W^i)$
$f_t=\sigma(x_tU^f+h_{t-1}W^f)$
$o_t=\sigma(x_tU^o+h_{t-1}W^o)$
$\tilde{C}_t=\tanh(x_tU^g+h_{t-1}W^g)$
$C_t=\sigma(f_t \cdot C_{t-1}+i_t \cdot \tilde{C}_t)$
$h_t=\tanh(C_t) \cdot o_t$

Figure 2.16: Structure of the LSTM cell and equations that describe the gates of an LSTM cell.

The way that neural networks solve problems is not by explicit programming, rather "learning" the solution based on given examples. There exist many ways to "teach" the neural network how to provide the right answer, however in this work we are focusing on ***supervised learning***. Learning is a procedure which involves the creation of a map between an input and a corresponding output and in supervised learning the (input, output) pair is provided to the neural network. During training, the neural network adjusts its weights in order to provide the correct output based on the given input. Theoretically, at the end of training, the neural network should be able to infer the right output even for inputs that were not provided during training, which is known as ***generalization***.

Training is stopped when the neural network can sufficiently predict the right output to each training input. However, a definition of the closeness between the desired value and the predicted value needs to be defined. This metric is known as ***cost/loss function*** and guides the neural network towards the desired outcome by estimating the closeness between the predicted and the desired value. The cost function is calculated at the end of every training iteration after the weights have been updated. The cost function that we used is known as ***mean squared error***, which tries to minimize the average squared error between the desired output and the predicted output, given by

$$cost = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.18}$$

where n is the number of data, Y_i is the target value and $\hat{Y}_i$ is the predicted value.

The procedure in which the weights are updated during training in order to minimize the cost function is known as ***backpropagation***. Backpropagation is a method that calculates the gradient of the cost function with respect to the weights, through the process of stochastic gradient descent. In order to be able to use neural networks to find solutions to a variety of applications (linear and non-linear), it is required to have an non-linear ***activation function*** at the processing step of every node. This function defines the contribution of this node to the subsequent nodes that it is connected to.

In the following Chapters we will present the implementation of different neural network based decoders. The first implementation shown in Chapter 4 is based on a Feed-Forward neural network with only one hidden layer. More decoder designs based on different types of neural networks such as Feed-Forward and Recurrent are described and compared in Chapter 5.

3

PAULI FRAMES FOR QUANTUM COMPUTER ARCHITECTURES

The Pauli frame mechanism allows Pauli gates to be tracked in classical electronics and can relax the timing constraints for error syndrome measurement and error decoding. When building a quantum computer, such a mechanism may be beneficial, and the goal of this chapter is not only to study the working principles of a Pauli frame but also to quantify its potential effect on the logical error rate. To this purpose, we implemented and simulated the Pauli frame module which, in principle, can be directly mapped into a hardware implementation. Simulation of a surface code 17 logical qubit has shown that a Pauli frame can reduce the error rate of a logical qubit up to 70% compared to the same logical qubit without Pauli frame when the decoding time equals the error correction time, and maximum parallelism can be obtained.

3.1. INTRODUCTION

Quantum computing is an emerging technology that promises to solve problems which are intractable by classical computers. Quantum computers exploit quantum phenomena for computational purposes using qubits. Various implementations of qubits and small quantum systems already exist, and they share one property: qubit states are fragile. Qubits interact with the environment and information stored in the qubits tends to get corrupted, which is known as decoherence. As a result, qubits cannot reliably store information for a long time, and quantum operations are error prone.

To enable quantum computing using quantum systems with high error rates, Quantum Error Correction (QEC) was introduced [11]. QEC allows quantum states to be encoded in logical qubits and errors to be detected based on error syndromes that are obtained by executing Error Syndrome Measurement (ESM) circuits. The error syndromes are decoded using classical algorithms which identify the most likely errors in the system. By using QEC, we can satisfy the demands of quantum algorithms to have qubits with low error rates. Besides from the benefits, QEC introduces overhead which creates new challenges. ESM and decoding should be performed in as short time as possible to reduce the overhead of QEC. The requirement of fast error decoding introduces high demands on classical algorithms and computational devices.

The concept of Pauli frames was proposed in [46] to loosen the timing constraints on ESM and decoding. A Pauli frame allows detected errors to be tracked in classical electronics, making it unnecessary to apply corrections on qubits. When using a Pauli frame fewer gates need to be applied, and the execution of ESM and decoding can be performed in parallel instead of sequential. Hence, the timing constraints on ESM and decoding are relaxed, making it easier to implement fully functional QEC. As the overhead of QEC is reduced, the error rate of logical qubits can in principle also be reduced.

The contributions of the chapter are as follows: (i) we implement and simulate the working principles of a Pauli frame for a Surface Code 17 (SC17) logical qubit and (ii) quantify under what conditions the Pauli Frame Unit (PFU) reduces, up to 70%, the logical error rate.

This chapter is organized as follows: Section 3.2 provides a background and introduces the relevant quantum concepts used throughout this chapter. Section 3.3 presents the working principles and applications of Pauli frames. Section 3.4 introduces the heterogeneous Quantum Computer Architecture as proposed by [102]. Our simulation software and setup is explained in Section 3.5 while the simulation results are presented in Section 3.6. We conclude the chapter in Section 3.7.

3.2. BACKGROUND

While classic bits can only be in a 0 or 1 state at a certain point in time, qubits can be in a *superposition* of both. Qubits can be in a linear combination of the two basis states, $|0\rangle$ and $|1\rangle$, and are therefore represented as: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ are complex probability amplitudes. The sum of all probabilities within a system is 1, therefore: $|\alpha|^2 + |\beta|^2 = 1$. By defining $|0\rangle = [1 \quad 0]^T$ and $|1\rangle = [0 \quad 1]^T$ as a computational basis, we can represent a qubit as a vector $|\psi\rangle = [\alpha \quad \beta]^T$. When a qubit is measured in the computational basis, it is projected into the $|0\rangle$ or $|1\rangle$ state with probabilities $|\alpha|^2$ or $|\beta|^2$, respectively.

The second feature of qubits that extends the capabilities of classical bits is *entanglement*. Qubits can be entangled with each other, which means that the superposition state of the entangled qubits cannot be represented as a tensor product of individual qubit states.

To manipulate a qubit state, we use quantum gates which can be expressed as unitary matrices. Quantum gates are represented as a $2^n \times 2^n$ unitary matrix where n equals the number of qubits the gate acts on. A few common single qubit gates with their corresponding matrices are shown in Equation (3.1). Examples of common two-qubit gates are the $CNOT$ and the CZ gate. Quantum gates together with initialization and measurement operations can be combined into a quantum circuit to perform quantum computations.

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \; H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3.1}$$

We list three groups of quantum gates [73] which are in our interest: Pauli gates, Clifford gates, and Non-Clifford gates. The *Pauli gates* are a basic group of single qubit gates which includes gates such as the X and Z gate. The *Clifford group* is finite and is defined as the normalizer of the Pauli group which means that for every Clifford gate C and Pauli gate P there exists a Pauli gate P' such that $CP = P'C$. Examples of Clifford gates are the H and $CNOT$ gate. All quantum gates that are not in the Clifford and Pauli group are known as *non-Clifford* gates, such as the T and $T^{\dagger}$ gate. Both Clifford and Non-Clifford gates are required for universal quantum computing, as explained in [42].

3.2.1. QUANTUM ERROR CORRECTION

There are different ways to implement qubits physically, and all of them have one factor in common: physical qubits suffer from *decoherence*. A qubit loses its state in a short period which makes it hard to maintain a quantum state for a long time. Also, the execution of operations on physical qubits are not perfect and can introduce errors. To enable meaningful quantum computation with high fidelity, QEC was introduced [11]. In QEC a quantum state can be encoded redundantly by entangling multiple physical qubits which form a logical qubit. This logical qubit may have lower error rates than their underlying physical qubits. These error rates are also referred to as the Logical Error Rate and Physical Error Rate. A popular QEC code is the *surface code* [63] which is derived from Kitaev's toric code [60].

In this section, we will focus on the surface code with 17 qubits encoding a single logical qubit which we will refer to as the SC17.

Figure 3.1 shows a schematic overview of a SC17 logical qubit consisting of 9 *data qubits* (blue) and 8 *ancilla qubits* (X/Z ancilla qubits in red/green) while the lines between qubits indicate the allowed two-qubit interactions. The nine data qubits encode the logical qubit state, and the eight ancilla qubits are used to detect possible errors. As described in [3, 63], we can use the eight ancilla qubits to measure the parities among the data qubits, resulting in 8-bit of parity data where X/Z ancilla qubits yield information about Z/X errors. This process, which exclusively contains Clifford gates and initialization/measurement operations, is referred to as an ESM and the resulting 8-bit data after measuring the ancilla qubits is known as an 8-bit error syndrome. Error syndromes can be processed by a *decoder* which performs a classical graph algorithm and returns the most likely error happened on the data qubits. The identified errors, which are always a combination of X and Z errors, can be corrected by performing Pauli gates on the data qubits.

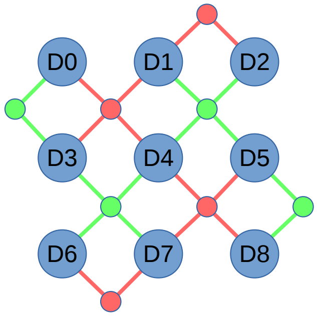


Figure 3.1: Schematic of a SC17 logical qubit.

Figure 3.2 presents the different steps of the QEC process for a SC17 logical qubit of which the variable descriptions are shown in Table 3.1. In the presented schedule, we first perform a logical operation which is followed by r rounds of ESM. The obtained error syndromes are given to a decoder that outputs a set of errors most likely happened. The errors are corrected, and the cycle is repeated.

The presented schedule performs decoding at run-time enabling the execution of logical non-Clifford gates, which is required by universal quantum computing. Based on Figure 3.2, we have $t_{\mathrm{cycle}} = t_{\mathrm{lop}} + t_{\mathrm{ec}} + t_{\mathrm{d}} + t_{\mathrm{c}}$. For current superconducting qubits, t_{ec} and t_{d} will dominate t_{cycle} [103]. To maximize efficiency, t_{ec} and t_{d} should be as short as possible which puts major time constraints on error correction and decoding. In the next section, we will discuss Pauli frames, which is a technique that can ease the time constraints on error correction and decoding.

Variable	Description
t_{ESM}	Time to perform the ESM circuit.
r	Rounds of ESM per cycle.
t_{ec}	Total time for error correction: $r \cdot t_{\mathrm{ESM}}$.
t_{d}	Time required for decoding.
t_{c}	Time to perform corrections on data qubits.
t_{lop}	Time reserved for logical operations.
t_{cycle}	Total time for a single cycle of the system.

Table 3.1: Variables used in the execution schedules.

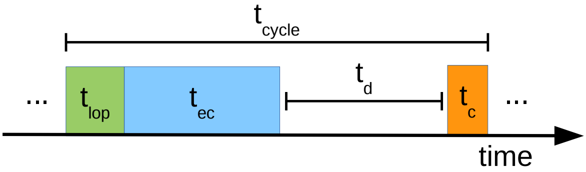


Figure 3.2: Execution schedule of a SC17.

3.3. PAULI FRAMES

The basic idea of Pauli frames is to track Pauli gates in classical electronics instead of applying them on qubits. For every qubit in the system, there exists a *Pauli record* that tracks its corresponding Pauli gates. The Pauli records of all qubits in a quantum system together form a Pauli frame. This idea was first proposed in [46] but has also been discussed in [73, 104–107]. Previous research on Pauli frames mainly discusses the theoretical working principles, but do not take into account their implementation. In this section, we present the basic mechanism of a Pauli frame and study the impact of a Pauli frame in the context of a heterogeneous QCA as proposed by [102].

A single Pauli record R_q tracks all the Pauli gates that are applied on qubit q. Due to the mathematical properties of Pauli gates, every set of tracked Pauli gates can be reduced to one of the elements in the set $\{I, X, Z, XZ\}$. Hence, every Pauli record $R \in \{I, X, Z, XZ\}$ and requires a two-bit memory. As a result, a system with n qubits requires $2n$ bits of memory for the Pauli frame.

To be able to update Pauli records at run-time, the Pauli frame needs to be aware of all operations applied on the qubits. Therefore the Pauli frame can be seen as a quantum operation filter.

All qubit operations and returned measurement results pass through the filter and can be modified by the Pauli frame. To make the Pauli frame system suitable for universal quantum computing it should be able to handle five types of quantum operations.

(i)Initialization of a qubit q will result in clearing the Pauli record of the corresponding qubit, $R_q = I$. (ii)A measurement operation on qubit q passes the Pauli frame system, but the returned measurement result m_q can be corrected based on the current state of its Pauli record. For instance, if $m_q = +1$ and $R_q = X$ then $-m_q$ is returned. (iii)Pauli gates are directly stored in the Pauli frame and do not require to be physically applied on the qubits. (iv)As mentioned in Section 3.2, the group of Clifford gates is defined as the normalizers of the Pauli group which means that Clifford gates map Pauli records to new valid Pauli records. After mapping the Pauli records, the Clifford gate is still applied on its target qubits. (v)The execution of non-Clifford gates requires the Pauli records of the target qubits to be *flushed* (i.e. physically apply the Pauli gates stored in R_q on qubit q and clear the Pauli record $R_q = I$) before the non-Clifford gate can be executed. The execution steps for the different qubit operations are summarized in Table 3.2.

3.4. A QUANTUM COMPUTER ARCHITECTURE WITH PAULI FRAME

Multiple papers [46, 73, 104–107] have covered the topic of Pauli frames, and [108–111] have discussed various architectures for quantum computer software and hardware, but no practical implementations of a Pauli frame for future quantum computers have been presented so far. In this section, we provide a high-level description of how a Pauli frame can be implemented as part of a QCA and we will discuss the expected benefits which directly relate to the logical error rate.

Operations	Execution steps
Initialization to $\lvert 0\rangle$	1. Set Pauli record of target qubit to I. 2. Initialize target qubit to $\lvert 0\rangle$.
Measurement	1. Measure target qubit. 2. Correct measurement result based on Pauli record.
Pauli gates	1. Map Pauli record of target qubit.
Clifford gates	1. Map Pauli record(s) of target qubit(s). 2. Apply Clifford gate on target qubit(s).
Non-Clifford gates	1. Flush Pauli record(s) of target qubit(s). 2. Apply non-Clifford gate on target qubit(s).

Table 3.2: Execution steps for different operations when using a Pauli frame.

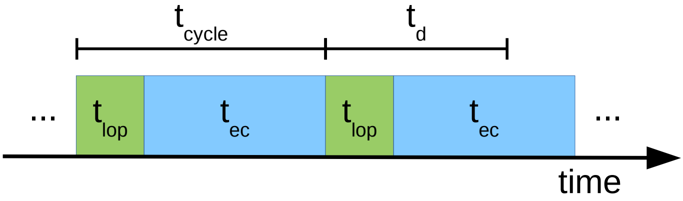
3.4.1. BENEFITS

The most interesting application of a Pauli frame is to use it for physical qubits in combination with QEC. In such a structure, correction gates, which are all Pauli gates, can be directly stored in the Pauli frame, reducing the number of gates being applied on the qubits. Also, logical Pauli gates can be directly stored in the Pauli frame and do not need to be applied on the physical qubits. To quantify the potential impact such a mechanism can have, we analyzed some benchmarks provided with the ScaffCC compiler [112] and found that the resulting quantum circuits contain up to 6% Pauli gates.

We found that compiled quantum programs contain 20 to 50% non-Clifford T and $T^\dagger$ gates which require flushing of the involved Pauli records. Flushing can be prevented by applying T and $T^\dagger$ gates using particular ancilla states and Clifford circuits as discussed in [42, 63]. So the first benefit is that Pauli gates can be processed faster and with a fidelity of 100% which can potentially reduce the error rate of a logical qubit.

The second benefit is directly related to the first but taps in QEC as correction gates for detected errors are always Pauli gates, and ESM circuits only contain Clifford gates. Hence, we can track correction gates without the need to flush while performing ESM.

As a result, the QEC system does not have to wait for the decoder to generate corrections and apply them before execution can continue. By eliminating this dependency, we can create a new execution schedule which is shown in Figure 3.3. The new schedule effectively removes the time reserved for applying corrections and allows parallel execution of error correction and decoding. For the new schedule with Pauli frame $t_{\mathrm{cycle}}|_{\mathrm{PF}} = \max\left(t_{\mathrm{ec}} + t_{\mathrm{lop}}\ ,\ t_{\mathrm{d}}\right)$. As a consequence of the more efficient schedule, we can perform the same number of cycles in less time compared to the system without Pauli frame, potentially reducing the LER. On top of that, the new schedule also eases the timing constraints on t_{ec} and t_{d}.

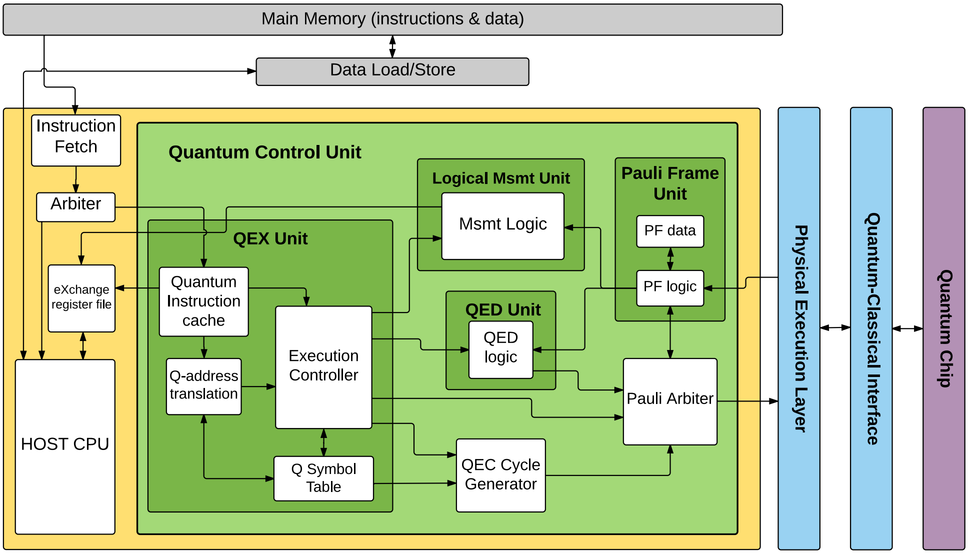


t cycle
t d
...
t lop
t ec
t lop
t ec
...
time

Figure 3.3: Execution schedule with Pauli frame.

3.4.2. IMPLEMENTATION

A heterogeneous QCA is proposed in [102] which supports the execution of QEC and logical operations for a single SC17 logical qubit implemented with transmon qubits. Figure 3.4 shows a simplified version of the proposed architecture which focuses on the QCU part of the QCA. The QCU decodes the instructions belonging to the QISA, inserts QEC routines, and manages feedback control. The QCU can communicate with the host CPU where classical computations are executed. The QCU outputs a sequence of timed quantum operations which are forwarded to the PEL.



Main Memory (instructions & data)
Data Load/Store
Instruction Fetch
Arbiter
eXchange register file
HOST CPU
Quantum Control Unit
QEX Unit
Quantum Instruction cache
Q-address translation
Execution Controller
Q Symbol Table
Logical Msmt Unit
Msmt Logic
QED Unit
QED logic
QEC Cycle Generator
Pauli Frame Unit
PF data
PF logic
Pauli Arbiter
Physical Execution Layer
Quantum-Classical Interface
Quantum Chip

Figure 3.4: Simplified architecture of a QCA targeting a SC17 logical qubit.

The PFU consists of a Pauli frame (PF data) and mapping logic (PF logic) and works closely together with the Pauli arbiter. Figure 3.5 shows a detailed schematic of the PFU. All quantum operations will pass the Pauli arbiter which will decide if the operation is forwarded to the PFU, the PEL, or both. We distinguish the following different situations. (i) Pauli gates are only forwarded to the PFU where the PF logic module will store it in the Pauli record of the target qubit. (ii) Clifford gates are forwarded to both the PFU and the PEL. The PF logic module will map the

Pauli records of the target qubits to new valid records based on the type of Clifford gate. (iii) Initialization operations are also forwarded to both the PFU and the PEL where the PF logic module will reset the Pauli record of the target qubit. (iv) Non-Clifford gates are directly forwarded to the PFU which flushes the Pauli records of the target qubits and forwards the pending Pauli gates to the Pauli arbiter. The Pauli arbiter again forwards the received Pauli gates and the non-Clifford gate to the PEL. (v) Measurement operations received by the Pauli arbiter are only forwarded to the PEL. After the PEL has performed the measurement, the outcome is forwarded to the PF logic block which will read the Pauli record of the corresponding qubit and correct the measurement outcome if required. The corrected measurement result is forwarded to other parts of the QCU.

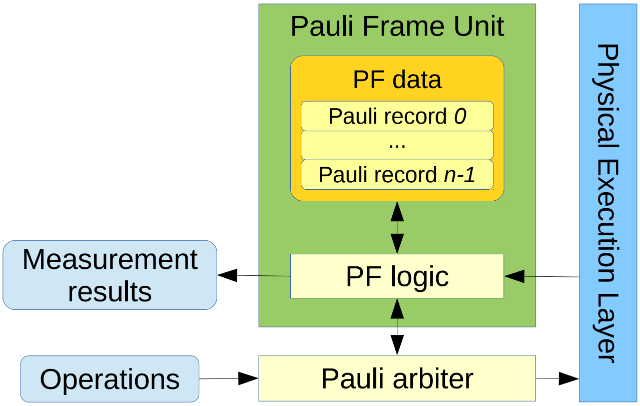


Figure 3.5: High level architectural view of the PFU.

3.5. SIMULATION SETUP

To quantify the impact of the Pauli frame mechanism, we developed the QPDO which allows us to simulate a SC17 logical qubit with and without a Pauli frame. QPDO is a software package that can simulate quantum execution platforms and has a layered structure where each layer can implement different functionality. Layers can be combined to create various control stacks allowing simulation of different platforms. Simulations are performed by supplying a stream of operations to a control stack.

Quantum simulations are not conducted by QPDO, but by external tools which are connected to QPDO as a back-end simulation layer. QPDO connects to the universal QX Simulator [113] which allows simulation of arbitrary quantum circuits. The second simulation back-end is the CHP stabilizer simulator [114] which allows efficient simulation of Clifford circuits based on the Gottesman-Knill theorem [21].

QPDO represents a quantum circuit as a set of qubit operations divided into discrete *time slots* where we assume that every operation takes a single time slot to execute.

Operations in a single time slot are executed in parallel and qubits can only be assigned to one operation per time slot. During our simulations, the ESM circuits for X and Z ancilla qubits are performed in parallel as shown in [3]. The result is an ESM circuit containing a total of 48 operations divided over eight time slots. Hence, $t_{\text{ESM}} = 8$ time slots. Table 3.3 summarizes which time slot contains what operations.

Time slot	Description
1	Initialize X ancilla qubits.
2	Initialize Z ancilla qubits and apply H gates on X ancilla qubits.
3-6	$CNOT$ gates data and ancilla qubits.
7	Apply H gates on X ancilla qubits.
8	Measure all ancilla qubits.

Table 3.3: The ESM circuit used in our test setup.

To introduce errors in our quantum system, we developed a QPDO layer that implements the symmetric depolarizing error model as presented in [3, 42]. In this model, the PER p is the probability of an error occurring while executing a single operation on a physical qubit where idling is also considered to be an operation. The depolarizing model assumes that errors are independent and that the error probability is the same for each quantum operation.

Decoding of the error syndromes is done using a RBLUT decoder as presented in [3]. The RBLUT decoder is specifically designed for the SC17 and uses a sliding window of three ESM rounds to detect errors. Every window uses one ESM result from the previous window as shown in Figure 3.6 which means that every cycle contains two rounds of ESM. Hence, $r = 2$ and $t_{\text{ec}} = 16$ time slots. All corrections can always be applied in a single time slot, therefore $t_{\text{c}} = 1$ time slot.

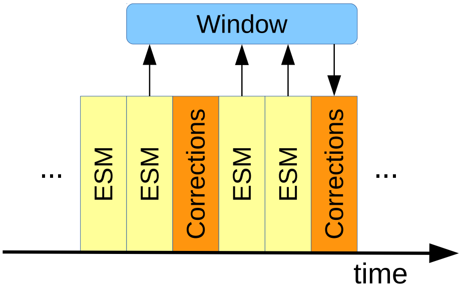


Figure 3.6: ESM results used for successive decoding windows.

The Pauli frame is implemented as a layer which allows us to add a Pauli frame to any platform easily. The control stack used for our simulations consists of a CHP simulation back-end with an error layer on top. The simulations that use a Pauli frame also add a Pauli frame layer on top of the error layer. The full control stack used for our simulations is shown in Figure 3.7.

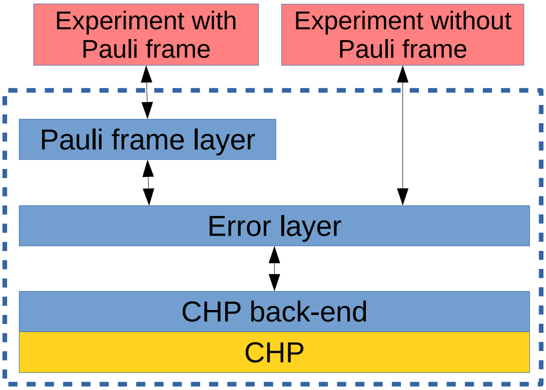


Figure 3.7: The QPDO control stack used for the simulations.

3.5.1. LOGICAL ERROR RATE CALCULATION

By simulation, we can find the LER P_L of a SC17 logical qubit for different values of PER p. In [3], P_L is defined as the probability of a logical error happening within a single cycle where no logical operations are performed, $t_{\mathrm{lop}} = 0$ time slots. During a simulation, a SC17 logical qubit is initialized to an error-free state before we repeatedly execute cycles. After every cycle, we check if a logical error occurred. This procedure is repeated while counting the number of cycles executed C and the total number of logical errors detected m until m reaches a predefined maximum value. P_L corresponding to a given p can be written as:

$$P_L\big|_p = \frac{m}{C}\,. \tag{3.2}$$

3.6. RESULTS

We performed separate LER simulations for X_L/Z_L errors, with a maximum of 20 logical errors per simulation, using the test setup shown in Figure 3.7. Simulations were performed for a PER ranging from 1.0×10^{-4} to 1.0×10^{-2} with a step size of 1.0×10^{-4}. For every PER we take 50 samples with and without Pauli frame, and the average of the resulting LER per sample yields our final result. We found that the combination of 50 samples and a maximum of 20 logical errors per simulation yields good precision in reasonable simulation time.

We performed simulations with decode time $t_{\mathrm{d}} = \{0,8,16,24\}$ time slots, equivalent to $\{0,0.5,1,1.5\} \cdot t_{\mathrm{ec}}$, and the resulting graph for X_L errors is shown in Figure 3.8.

The graph for Z_L errors is not shown since it is equivalent to the graph for X_L errors, which is expected when using a symmetric depolarizing error model. Results for the test setup without Pauli frame are plotted with squares while the results with Pauli frame are plotted with circles. The results for the test setup with Pauli frame and $t_{\mathrm{d}} = \{0,8,16\}$ time slots are equivalent and therefore only one of them is plotted. We also added a black dashed line indicating the points where the physical error rate is equal to the logical error rate ($x = y$) and vertical dashed lines which indicate the intersection between the linear interpolated results and the line $x = y$, also known as the *pseudo-threshold*. For a PER lower than the pseudo-threshold, the LER is lower than the PER which means that we benefit from using QEC.

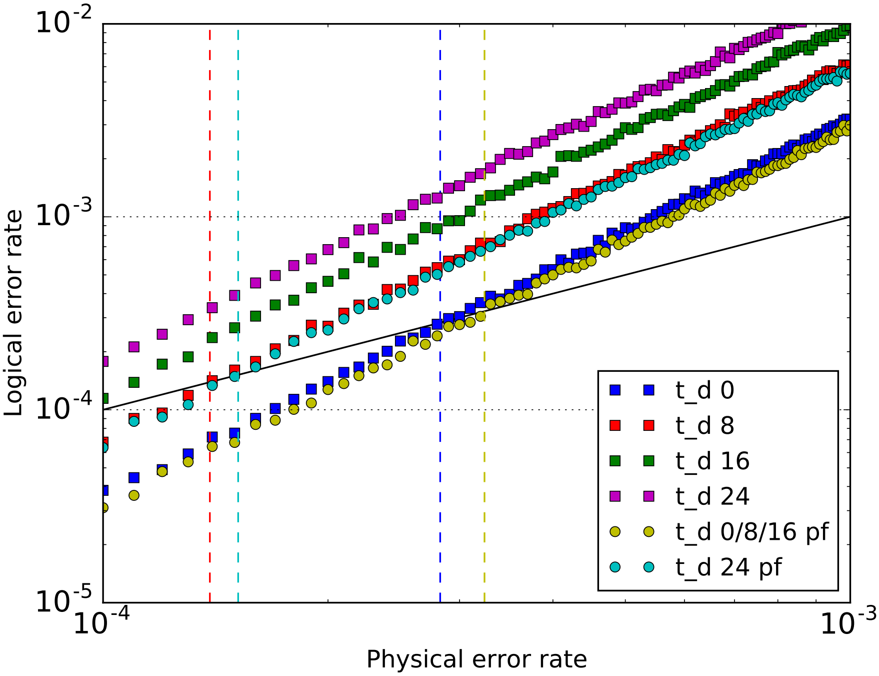


Figure 3.8: Calculated LER for X_L errors for d=3 rotated surface code under circuit noise error model. The points denoted with circles are assuming the use of a Pauli frame and the points with squares do not. Each curve depicted with a different color, assumes a different t_d as described in the legend.

From Figure 3.8, we can see that for every value of t_d the system with Pauli frame has a lower LER than the system without Pauli frame. For the system without Pauli frame, the LER increases directly when t_d increases while for the system with Pauli frame the LER only increases when $t_d > t_{ec}$ where in our simulations $t_{ec} = 16$ time slots. The Pauli frame effectively reduces the cycle time t_{cycle} by allowing ESM and decoding to be performed in parallel which results in a reduced LER.

To see if the reduction in t_{cycle} by using a Pauli frame is proportional to the observed reduction in LER P_L, we plotted the relative reduction by using a Pauli frame $R_{PF}()$ for both over the range $t_d = [0,40]$ time slots where the relative reductions are defined as:

$$R_{\mathrm{PF}}\left(t_{\mathrm{cycle}}\right) = 1 - \frac{t_{\mathrm{cycle}}|_{\mathrm{PF}}}{t_{\mathrm{cycle}}|_{\mathrm{without\ PF}}} , \tag{3.3}$$

$$R_{\mathrm{PF}}\left(P_L\right) = 1 - \frac{P_L|_{\mathrm{PF}}}{P_L|_{\mathrm{without\ PF}}} . \tag{3.4}$$

Figure 3.9 shows the resulting graphs where for the relative reduction in LER (red squares) only data for $t_{\mathrm{d}} = \{0,8,16,24\}$ time slots is available.

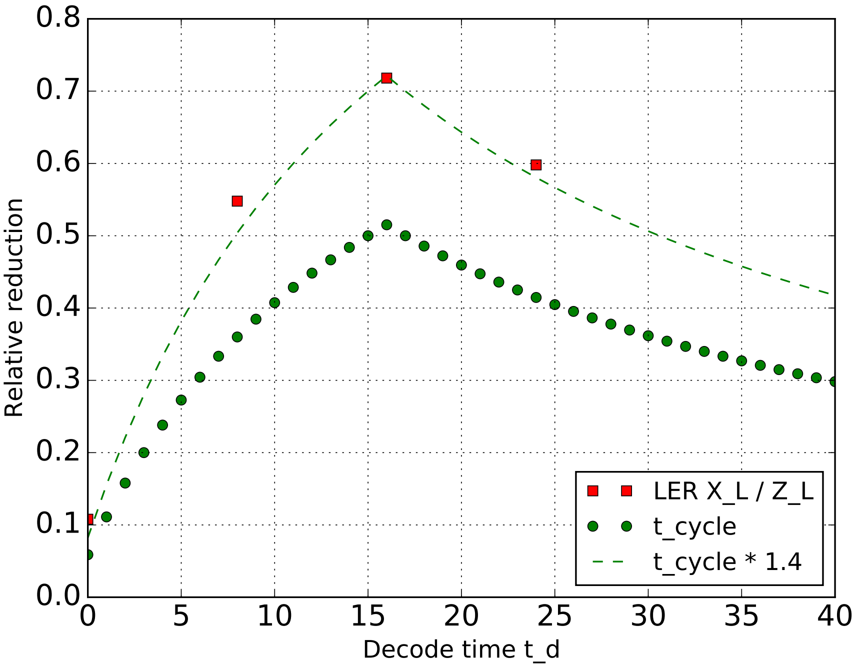


Figure 3.9: Relative reduction in t_{cycle} and LER by using a Pauli frame for different t_{d}.

From Figure 3.9, we can see that the reduction in LER appears to be proportional to the reduction in t_{cycle} with a proportionality constant ≈ 1.4. The maximum relative reduction by using a Pauli frame can be found at $t_{\mathrm{d}} = 16$ time slots which is the point where $t_{\mathrm{d}} = t_{\mathrm{ec}}$ and maximum parallelism can be obtained.

3.7. CONCLUSIONS

We presented an implementation of a PFU and quantified through simulation what the potential benefits are of using it. Based on our analysis of the application of a Pauli frame for systems using QEC, we can conclude that a Pauli frame enables us to perform ESM and decoding in parallel. Hence we can create a more efficient execution schedule for a SC17 logical qubit that reduces the cycle time t_{cycle}. Besides that, the new execution schedule also relaxes the timing constraints on the ESM and decoder. Simulation has shown that as a result of the reduced t_{cycle}, the LER of a SC17 logical qubit can be reduced up to 70% when the time required for error correction equals the decoding time $t_{\text{ec}} = t_{\text{d}}$. On the basis of these results, including a PFU in e.g. a QCA such as proposed in [102] pays off in terms of reduction in the LER. This payoff is maximal when the decoder time is equal to the time needed for error correction as there is no idle time induced by either operation. The effect goes down if those times are no longer equal. Future work will involve verifying if the observations also hold for SC49 and to embed a Pauli frame in a larger architectural simulation platform.

3.8. ACKNOWLEDGMENTS

We would like to express our gratitude to Dan Iorga and Nader Khammassi for their contribution to the simulation platform and simulation back-ends. We also would like to thank Ben Criger for the relevant feedback. Besides from that, we would like to thank all our colleagues for their support and input.

The contents of this chapter are based on the following paper:

L. Riesebos, X. Fu, S. Varsamopoulos, C. G. Almudever, and K. Bertels. **Pauli Frames for Quantum Computer Architectures**, *Proceedings of the 54th Annual Design Automation Conference (DAC'17)*, ACM, 2017, p. 76.

4

DECODING SMALL SURFACE CODES WITH FEEDFORWARD NEURAL NETWORKS

Surface codes reach high error thresholds when decoded with known algorithms, but the decoding time will likely exceed the available time budget, especially for near-term implementations. To decrease the decoding time, we reduce the decoding problem to a classification problem that a feedforward neural network can solve. We investigate quantum error correction and fault tolerance at small code distances using neural network-based decoders, demonstrating that the neural network can generalize to inputs that were not provided during training and that they can reach similar or better decoding performance compared to previous algorithms. We conclude by discussing the time required by a feedforward neural network decoder in hardware.

4.1. INTRODUCTION

Quantum computing has emerged as a solution to accelerate various calculations using systems governed by quantum mechanics. Such calculations are believed to take exponential time to perform using classical computers. Initial applications where quantum computing will be useful are simulation of quantum physics [115], cryptanalysis [8, 116] and unstructured search [10], and there is a growing set of other quantum algorithms [117].

Simple quantum algorithms have been shown to scale better than classical algorithms [118–120] for small test cases, though larger computers are required to solve real-world problems. The main obstacle to scalability is that the required quantum operations (state preparations, single- and two-qubit unitary gates, and measurements) are subject to noise, therefore quantum algorithms cannot run with perfect fidelity. This requires quantum computers to use active error correction [11, 121] to achieve scalability, which in turn requires a classical co-processor to infer which corrections to make, given a stream of measurement results as input. If this co-processor is slow, performance of the quantum computer may be degraded (though recent results [122] suggest that this may be mitigated).

The remainder of this chapter is organized as follows. In Section 4.2, we discuss the need for a fast classical co-processor. In Section 4.3, we give a brief summary of existing techniques to perform decoding quickly, and follow this in Section 4.4 with the introduction of a new technique based on feedforward neural networks. We examine the accuracy of the proposed decoder in Section 4.5, and conclude by discussing its speed in Section 4.6. Information regarding quantum error correction, the surface code and neural networks are summarized in chapter 2.

4.2. NEED FOR FAST DECODING

Projective measurement of the logical qubits and classical feedforward of the measurement values are key ingredients in universal fault-tolerant quantum computing. To calculate the bit which we feed forward, we need to decode. Thus, it is necessary to correct errors frequently during a computation.

While the decoding takes place at the classical co-processor, we could either continue running rounds of syndrome measurement or stop and wait for the decoding to be concluded. If we stop the computation, errors will build up until they become uncorrectable. This takes an amount of time which depends on the implementation in question ($\sim 10\,\mu$s in current superconducting circuits, for example [123]). On the other hand, if we continue measuring syndromes, we will build a *backlog* of data that produces a more difficult decoding problem in the future. The ideal case would be a decoder that decodes d rounds of syndrome measurement in less time than the time needed to perform the measurements themselves. In superconducting circuits, the time for a single round of syndrome measurement is 800 ns [71].

There are many techniques that provide high performance decoding. In the following section, we summarize some of them.

4.3. RELATED WORK

To decrease decoding time when correcting time-dependent errors, the “overlapping recovery” method was introduced in [1]. This method divides the measurement record into *windows*, defined as a set of $\sim d$ consecutive error correction cycles. In the overlapping recovery technique, syndrome changes are matched either to each other (pairwise) or to a time boundary placed immediately after the last round of syndrome measurement. At the next window, the syndrome changes matched to the time boundary are forwarded to the following window, in order to identify chains of errors which cross the boundary. This reduces the backlog problem mentioned earlier, by allowing the decoding problem to be solved incrementally.

To further reduce the backlog, Fowler [78] has parallelized the Blossom algorithm, using message-passing between local processors to replace slow subroutines. This technique produces accurate corrections, resulting in a high threshold error rate, and is scalable to large code distances. However, in the near future, only small code distances will be experimentally viable, so it is likely that a heuristic approach will perform well.

One such approach is taken in [3]. In this paper, the authors have designed a heuristic-based decoder that resembles the parallelized MWPM decoding for a distance-3 Surface Code with a window of 3 error correction cycles. The simple structure of this heuristic algorithm makes it easily programmable to hardware, decreasing the decoding time. The main drawback of this algorithm is that it cannot easily be extended to higher code distances, so an alternate method is required.

Currently, machine learning techniques are being explored as possible alternate decoding techniques for both classical LDPC codes [124, 125] and quantum codes, independently of the need for high-speed decoding. One such technique is being used in [86]. The authors of this paper use a stochastic neural network (or Boltzmann machine) to decode stabilizer codes. They optimize the neural network to fit a dataset that includes the errors and their respective syndromes. The network then models the probability distribution of the errors in the dataset and generates prospective recovery error chains when a syndrome is input. Many networks are produced for a variety of physical error probabilities p, so when an error syndrome is obtained, a random recovery chain of errors is sampled from the distribution corresponding to the known value of p. While this method was as accurate as MWPM decoding for simple error models, repeated sampling is required in order to produce an error that conforms with the syndrome, which takes unknown time.

To achieve high accuracy in bounded time, we use a simpler machine learning technique, the feed-forward neural network, which we introduce and apply to the decoding problem in the next section.

4.4. NEURAL NETWORK DECODER

To apply machine learning techniques to surface code decoding, we first reduce the decoding problem to a well-studied problem in machine learning; classification. Classification problems consist of a set of (generally high-dimensional) inputs, each of which is associated with a (generally low-dimensional) label. The goal is to optimize the assignment of known labels to known inputs (a process called *training*) so that unknown inputs can also be correctly labeled.

To reduce the decoding problem to a classification problem, we decompose an error E into three multi-qubit Pauli operators:

$$E = S \cdot C \cdot L, \tag{4.1}$$

where S is a stabilizer, C is any fixed Pauli which produces the syndrome $\vec{s}$ (also known as a *pure error* [126]), and L is a logical Pauli operator of the surface code, see Figure 4.1.

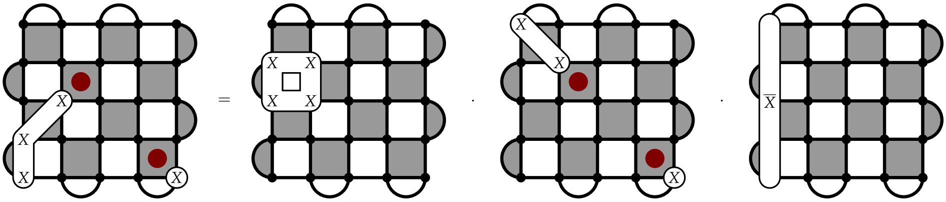


Figure 4.1: A surface code error E decomposed into three components; a stabilizer S, a fixed Pauli C which produces the same syndrome as E, and a logical operator L.

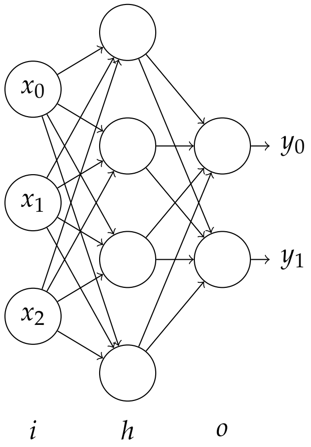
Any decoder which provides a correction $E' = S' \cdot C \cdot L$, in which the stabilizer in the correction is different from that in the actual error, does not lead to a logical error. This implies that S can be assigned arbitrarily with no impact on decoder accuracy. Also, it is possible to produce a pure error by parallel table look-up, since each bit of the syndrome can be assigned a unique pure error, independently of the other bits. We call the apparatus that produces this error the *simple decoder*. Since pure errors can be determined quickly in this fashion, the neural network only has to identify L, which can take one of four values; $\mathbb{I}$, $\bar{X}$, $\bar{Y}$, or $\bar{Z}$. These four values can be used as labels in a classification problem.

To solve this problem, we use feed-forward neural networks, which are widely regarded as the simplest machine learning technique [127]. A feed-forward neural net can be described graphically or functionally, see Figure 4.2. The artificial neurons which comprise the network are devices which store a length-m internal array of *weights* $\vec{w}$, as well as a *bias* b. Upon receiving a length-m input $\vec{x}$, they calculate $(1+\exp(-(\vec{w}\cdot\vec{x}+b)))^{-1}$ and either broadcast the result to a subsequent layer of neurons, or output the result directly if the neuron in question is part of the *output layer*. Such output is typically denoted $\vec{y}$, and its accuracy is measured using an appropriate cost function.

A typical cost function, which we use in this work, is the average cross-entropy:

$$\langle H(p,y)\rangle \propto -\sum_{(\vec{p},\vec{x})\in T} \vec{p}\cdot\ln(\vec{y}(\vec{x})), \tag{4.2}$$

where T is the training set, consisting of desired (‘target’) distributions $\vec{p}$ and input values $\vec{x}$. The cross-entropy is a measure of the divergence between two probability (or frequency) distributions, differing from the Kullback-Leibler divergence [128] only in terms that depend solely on $\vec{p}$. To minimize the cross-entropy, we use stochastic gradient descent, as implemented in the Tensorflow library [129]. To produce a training set, we use direct sampling at a single physical error probability, where the Blossom algorithm produces a logical error rate of $\sim 25\%$. This physical error probability is chosen so that a large variety of error syndromes can be produced while still ensuring that correction is possible. For small surface codes, it is possible to sample the entire set of possible syndromes, we limit the size of the training set to at most 10^6 samples for larger codes. This training set size provides relatively fast training and high accuracy, as seen in Section 4.5.



$$\vec{y} = \sigma\left(\hat{W}_o\sigma\left(\hat{W}_h\vec{x}+\vec{b}_h\right)+\vec{b}_o\right)$$

Figure 4.2: The graphical and functional descriptions of a feed-forward neural network. In the graphical description (left), inputs x_j are passed to neurons in a *hidden layer*, and each of these neurons outputs $\sigma(\vec{w}\cdot\vec{x}+b)$, where $\vec{w}$ and b are a local set of weights and a bias, and $\sigma(x)$ is a non-linear *activation function* (we use $\sigma(x)=(1+\exp(-x))^{-1}$ for all neurons considered in this work). The final outputs y_k can be rounded to $\{0,1\}$, and interpreted as a class label. In the functional picture, the weights and biases are assembled into matrices and vectors, respectively, allowing the output vector to be expressed as a composition of functions acting on the input vector.

In the following section, we compare the performance of our decoder to the performance of Blossom and the performance of the partial lookup table (PLUT), which contains the error syndromes and corrections from the training set. In the event that the input syndrome is not in the training set, the PLUT decoder multiplies the correction from the simple decoder by the most frequent logical operator, $\mathbb{I}$. The comparison in terms of performance is based on the logical error rate of each decoder for specific code distances and error models.

4.5. RESULTS

In the proposed decoder, we provide the error syndrome to both the simple decoder and the neural network. As presented in Table 4.1, the size of the input for the neural network is equal to the number of required syndrome bits, depending on the error model, and only one hidden layer was used for all networks. The number of nodes in the hidden layer was decided based on the performance of the neural network during training and testing.

QEC Error Models			
Code Capacity Noise			
Code distance	Input nodes	Hidden nodes	Output nodes
3	4	10	2
5	12	90	2
7	24	512	2
Depolarizing Noise			
3	8	128	4
5	24	660	4
7	48	256	4

FT Error Models			
Phenomenological Noise			
Code distance	Input nodes	Hidden nodes	Output nodes
3	16	768	4
Depolarizing & Measurement Noise			
3	32	768	4
Circuit Noise			
3	32	704	4

Table 4.1: Layer sizes for the neural networks used throughout this work. The number of input nodes is determined by the number of syndromes in the quantum error correction scenario, using only X (or Z) syndrome bits for independent X/Z errors, and all syndrome bits for depolarizing errors. For fault tolerance error models, d rounds of measurement are followed by readout of the data qubits, and calculated stabilizer eigenvalues are included in the input. The output layer is restricted to two nodes for independent X/Z errors, since logical X/Z errors are also independent. In all other scenarios, four nodes are used to discriminate between $\mathbb{I}$, $\bar{X}$, $\bar{Y}$, and $\bar{Z}$. The number of nodes in the hidden layer is determined by analyzing the performance of the resulting decoder empirically.

We test the proposed decoder against Blossom and the PLUT decoder for two classes of error models, called quantum error correction (QEC) and fault tolerance (FT). Quantum error correction (QEC) error models approximate noise only on data qubits and fault tolerance (FT) error models approximate noise on all qubits, gates and operations, therefore requiring multiple rounds of measurement to find all errors. The code capacity model inserts only X or only Z errors with probability p in the data qubits. The depolarizing model places $X/Y/Z$ errors with equal probability, $p/3$, on the data qubits. For these error models only one cycle of error correction is required to find all errors.

For fault tolerance error models, the probability of an error occurring on a qubit and the probability of a measurement error is the same, therefore the minimum number of rounds of measurement is taken to be d. Instead of data qubit and measurement errors, the circuit noise model assumes that all operations and gates are noisy. Each single-qubit gate is followed by depolarizing noise with probability $p/3$ and each two-qubit gate is followed by a two-bit depolarizing map where each non-$\mathbb{I} \otimes \mathbb{I}$ two-bit Pauli has probability $p/15$. Preparation and measurement locations fail with probability p, resulting in a prepared -1-eigenstate or measurement error, respectively.

In our simulations, more than 10^6 error correction cycles were run per point, and each point has a confidence interval of 99.9%. The percentage of the most frequent error syndromes that were used as training cases for the code capacity error model were 100% ($d = 3$), 72.46% ($d = 5$), 2.75% ($d = 7$), see Figures 4.3, 4.4, and 4.5, respectively. The percentage of the most frequent error syndromes that were used as training cases for the depolarizing error model were 100% ($d = 3$), 0.98% ($d = 5$), 3×10^{-7}% ($d = 7$), see Figures 4.6, 4.7, and 4.8, respectively. The percentage of the most frequent error syndromes that were used as training cases for the fault tolerance error models were 30.09%, 0.022% and 0.01% for the code capacity (see Figure 4.9), the depolarizing (see Figure 4.10), and the circuit model (see Figure 4.11), respectively. The performance of our decoder was compared to the Blossom algorithm and the PLUT decoder.

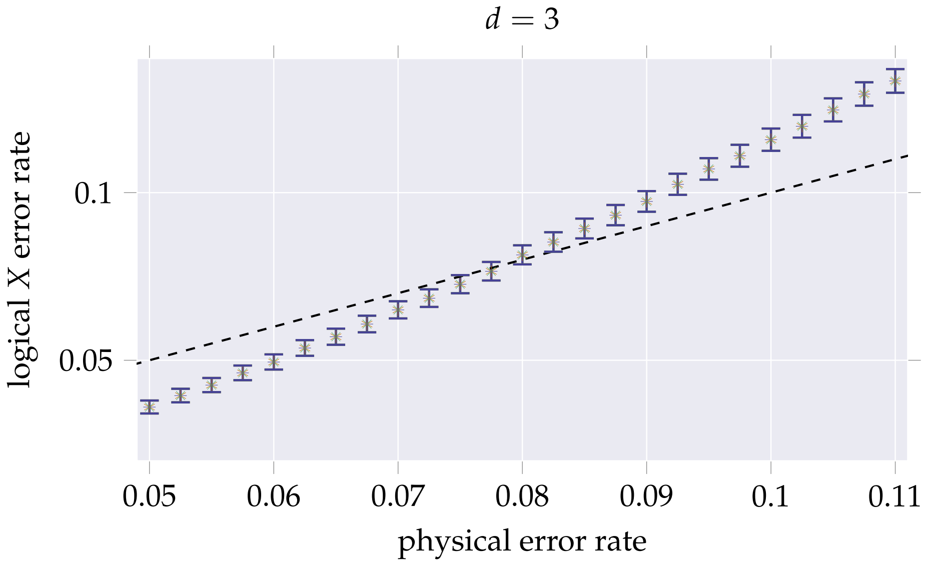


Figure 4.3: Code capacity error model without measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$. All points of all three curves are lying on top of each other.

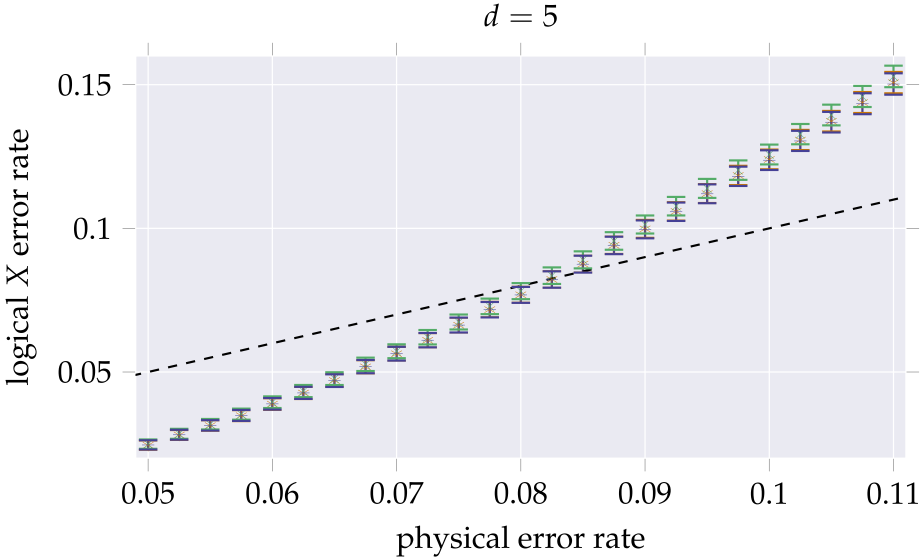


Figure 4.4: Code capacity error model without measurement errors for Surface Code with distance 5. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

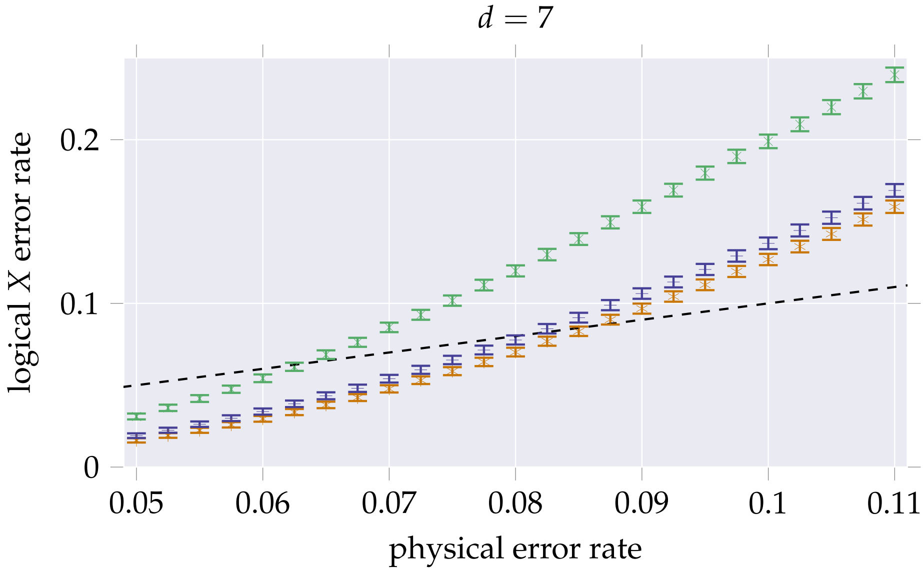


Figure 4.5: Code capacity error model without measurement errors for Surface Code with distance 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

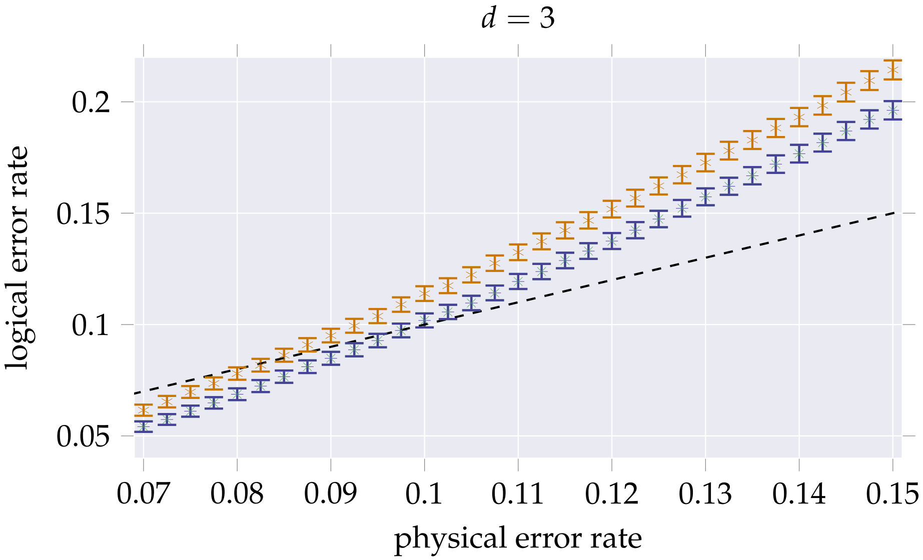


Figure 4.6: Depolarizing error model without measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

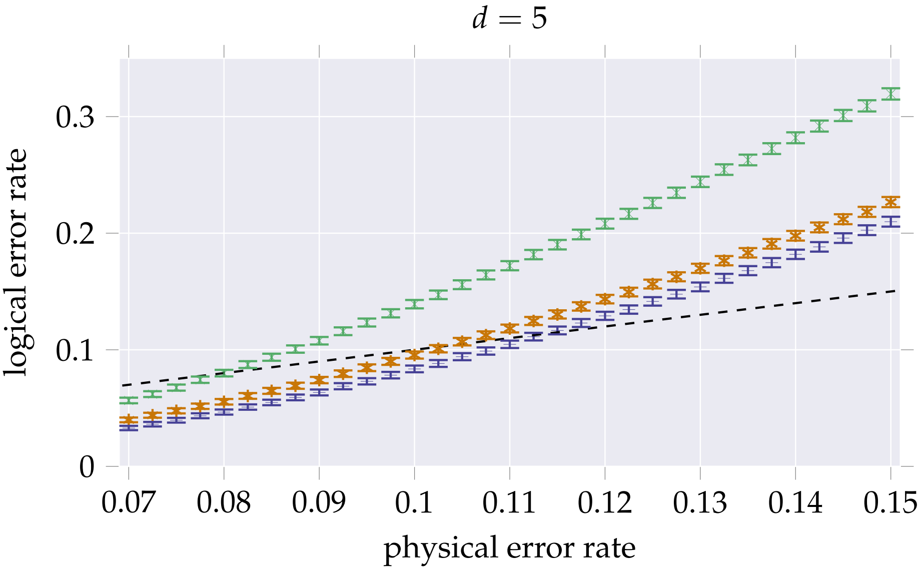


Figure 4.7: Depolarizing error model without measurement errors for Surface Code with distance 5. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

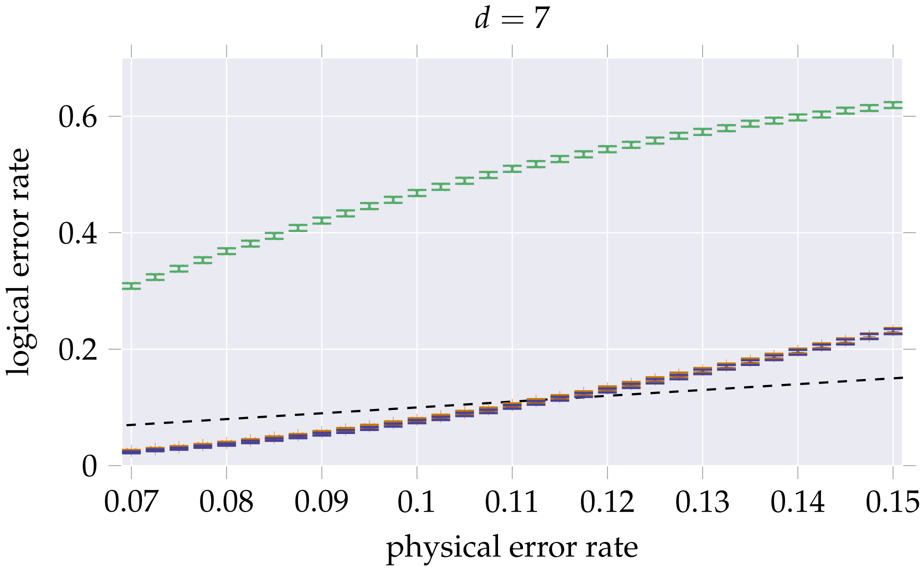


Figure 4.8: Depolarizing error model without measurement errors for Surface Code with distance 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

In the QEC error models, see Figures 4.3 to 4.8, we observe a clear trend. In both error models, as the distance increases the performance of our decoder remains similar to Blossom, and becomes much better than the PLUT-based decoder. This demonstrates that the neural networks of our decoder can successfully correct error syndromes that were not included in training. At small code distances, almost all possible error syndromes were used in training, resulting in identical performance from both the PLUT and our decoder. However, going to larger distances while using a small set of error syndromes for training, leads to sub-optimal decoding by the PLUT decoder.

It is known that, for the code capacity error model, Blossom can reach near-optimal accuracy, therefore it is sufficient for our decoder to reach similar accuracy. There are correctable errors (with weight ≤ 3) in distance 7 that are not included in the training set and the neural network is not generalizing correctly. Therefore, the performance is $\sim 1\%$ worse than Blossom's. However, for the depolarizing error model, Blossom is known to misidentify Y errors, since it performs the decoding for X and Z errors separately, treating a Y error as two distinct errors. Thus, if we train our decoder to take Y errors into account as weight-1 errors, the performance will be better than Blossom's. In the depolarizing model, there are still a few weight 3 errors that are being mis-identified, however the existence of higher weight errors in the training set, that are being corrected properly, account for the slightly better performance compared to the Blossom decoder.

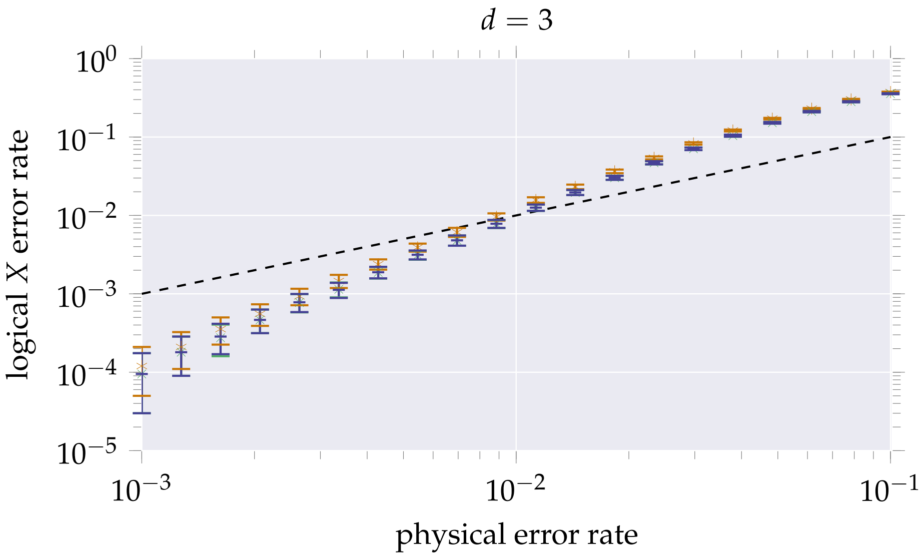


Figure 4.9: Code capacity error model with measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

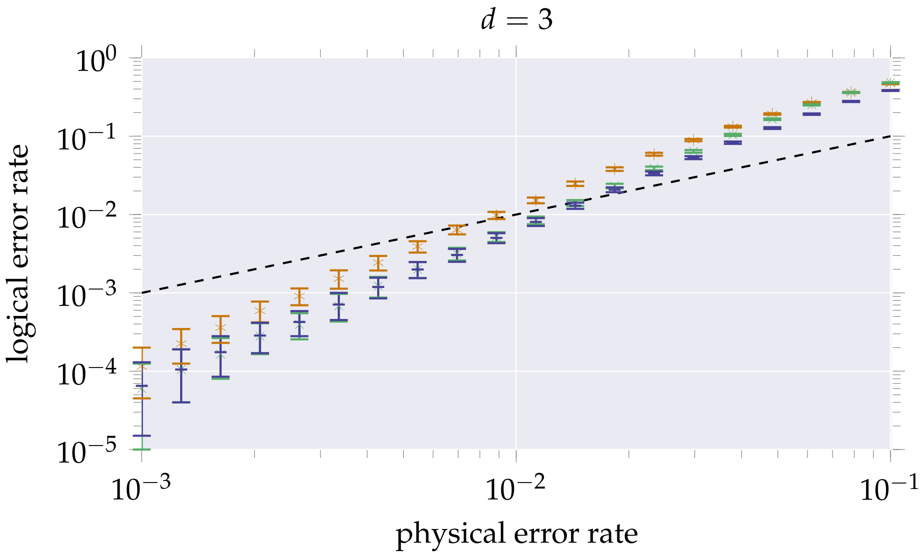


Figure 4.10: Depolarizing error model with measurement errors for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

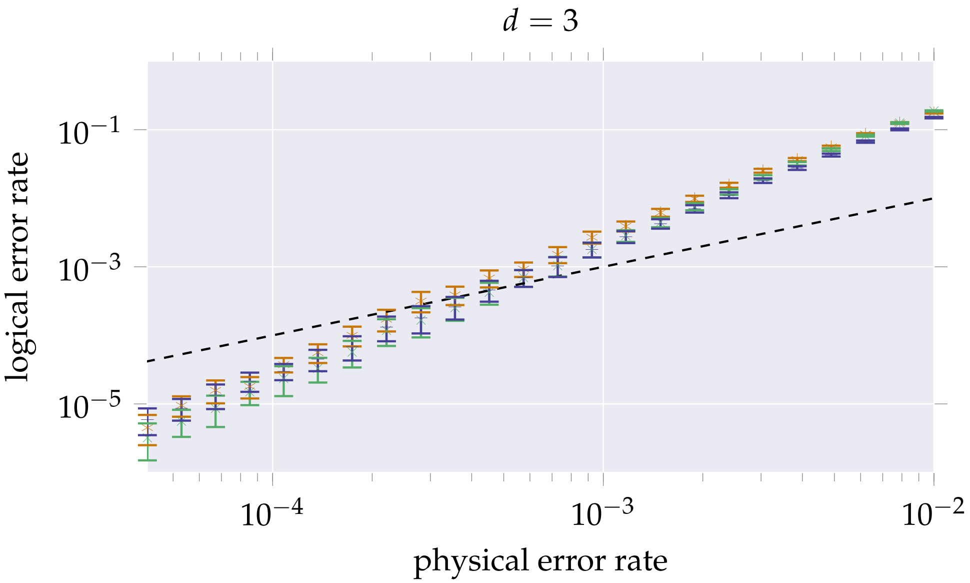


Figure 4.11: Circuit error model for Surface Code with distance 3. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green). The black dashed line represents the points that the physical error rate is equal to the logical error rate $x = y$.

In the fault tolerance scenario, see Figure 4.9, 4.10 and 4.11 due to the small code distance, all decoders reach a similar level of performance. Specifically, for the code capacity and the depolarizing error model, only a small amount of error syndromes was necessary to reach Blossom's performance. The circuit metric required more syndromes, however slightly better performance was achieved in this case as well.

It is encouraging that the neural network based decoder can achieve similar performance to Blossom. However, the main reason that such a design is proposed is to accelerate the decoding time. In the following section, we provide an estimation of the speed of the neural network based decoder in hardware, and discuss the implications for future research.

4.6. DISCUSSION AND CONCLUSION

To estimate runtime scaling of a neural network decoder of the type introduced above, we first note that the runtime is dictated by the sizes of the input, hidden and output layers. The input layer grows quadratically with the distance of the surface code in question, and the size of the output layer is constant. The size of the hidden layer, however, is selected through trial and error, so it cannot be predicted a priori.

Given a hidden layer size, we can estimate the required runtime, using the graphical description of neural networks (see Figure 4.2). Firstly, the output of each neuron can be evaluated independently, so the runtime is dominated by the time needed to take an inner product between the weight vector and an intermediate state in the network. Each multiplication can be performed independently, and summation requires a logarithmic-depth circuit composed of adders, arranged in a tree-like structure. This implies that any sub-exponential growth in the size of the hidden layer will result in sub-linear growth in the required decoding time.

To get a better idea of how these scaling arguments function in practice, we can bound the performance of a neural network decoder, and compare with similar bounds using the Blossom algorithm, running on a CPU. We consider an average instance of the decoding problem for the $d = 3$ rotated surface code correcting circuit noise, in which three syndrome differences occur during repeated measurement, leading to a graph with 6 vertices and 9 edges. We also neglect the runtime of the simple decoder, since, for the distance-3 code, it can be reduced to a simple look-up table.

To lower-bound the time required for the Blossom algorithm, we hard-code the relevant graph data into a C program and calculate the matching. This requires ~ 500 ns, indicating that the Blossom algorithm can compute a correction on the timescale required for the distance-3 code. However, this neglects the time required to translate syndrome data from experimental hardware into the data structure required by the algorithm. To upper-bound this time, we write a second program which reads the graph data from a file, resulting in a runtime of $\sim 6\,\mu$s, slightly too slow to be compatible with an 800 ns syndrome measurement time.

To lower-bound the runtime of the neural network, we make the assumption that each arithmetic operation will require one clock cycle of the appropriate FPGA, typically 1–5 ns [130]. The network which corrects errors from the circuit model requires two multiplications (one for the hidden layer, and one for the output layer), 15 addition steps ($\lceil \log_2(32) \rceil + \lceil \log_2(704) \rceil$), and two evaluations of the sigmoid function, for a total of 19 serial steps, requiring 19–95 ns. To obtain an upper bound, we estimate the number of clock cycles required using a high-level synthesis tool [49], which permits hardware-synthesizable code to be generated from C, but does not ensure that the resulting code is optimized for speed. This results in an estimate of ~ 3600 clock cycles, requiring ~ 3.6–$18\,\mu$s, again slightly too slow to be compatible with an 800 ns syndrome measurement time. Either decoder would require only minor optimization to attain compatibility at distance 3, an ideal topic for immediate future work.

In concert with this, we can extend the proposed decoder to the case where syndromes from a finite window are fed forward, as in [1]. In addition, we can begin testing the applicability of feedforward neural networks to surface codes with larger distance, as well as to alternate topological codes for which existing decoders do not attain high accuracy and speed simultaneously [57, 131, 132].

If feedforward neural networks can efficiently decode a large range of topological codes, it will also be interesting to determine the scope of their applicability. There are known families of sparse and/or high-rate stabilizer codes [55, 133, 134] for which fast and accurate decoders are not known. Aside from concatenated codes, which can be optimally decoded by fast message passing [126], neural networks provide an interesting avenue to quickly test the performance of small codes, with ~ 50 stabilizer measurements.

In conclusion, feedforward neural networks provide a fast and accurate method to decode small surface codes, both for performing quantum error correction, as well as fault-tolerant operations. Given that the hardware requirements and anticipated runtime are relatively low, we expect feedforward neural network decoders to be usable in the near term.

4.7. ACKNOWLEDGMENTS

We thank Tom O'Brien, Paul Baireuther and Brian Tarasinski for useful discussions, and Imran Ashraf for his timely and invaluable assistance with high-level synthesis tools for FPGA programming. We acknowledge financial support from Intel in the context of the QuTech-Intel collaboration.

The contents of this chapter are based on the following paper:

S. Varsamopoulos, B. Criger, K. Bertels, **Decoding small surface codes with feedforward neural networks**, *Quantum Science and Technology*, vol. 3, num. 1, pp. 015004, IOP Publishing, 2018.

5

DESIGNING NEURAL NETWORK BASED DECODERS FOR SURFACE CODES

Recent works have shown that small distance quantum error correction codes can be efficiently decoded by employing machine learning techniques like neural networks. Various techniques employing neural networks have been used to increase the decoding performance, however, there is no framework that analyses a procedure in designing such a neural network based decoder. The objective of this work is to present a detailed framework that investigates how neural network parameters affect the decoding performance, the training and execution time of a neural network based decoder. We focus on parameters such as the size of the training dataset, the structure, and the type of the neural network, the batch size and the learning rate used during training. We compare various decoding strategies and showcase how these influence the objectives for different error models. For the noiseless syndrome measurements, we reach up to 50% improvement against the benchmark algorithm (Blossom) and for the noisy syndrome measurements, we show efficient decoding performance up to 97 qubits for the rotated Surface code. Furthermore, we show that the scaling of the execution time is linear with the number of qubits, which is a significant improvement compared to existing classical decoding algorithms.

5.1. INTRODUCTION

Quantum computers are a promising solution to a class of complex problems that classical supercomputers cannot currently solve or require an immensely large amount of time to solve. However, since quantum computing is still in its early stages, classical computers are still the driving force, using the prototypes of quantum computers as accelerators for specific applications.

In the recent past, there is an increasing dominance of heterogeneous, multi-core architectures with multiple processors. In such architectures, a classical core processor interacts with different co-processors such as Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), Tensor Processing Units (TPUs) and in this case a quantum processor. Quantum computers require both classical and quantum computing components, because it needs a lot of monitoring and control from the classical part and all quantum applications consist of classical and quantum logic. Typically, when developing such an architecture, one has to develop a full stack going from algorithms up to the chip implementation.

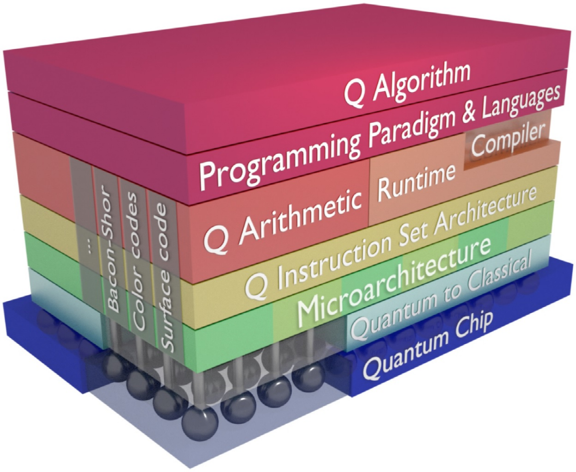

Q Algorithm
Programming Paradigm & Languages
Compiler
Runtime
Q Arithmetic
Q Instruction Set Architecture
Microarchitecture
Quantum to Classical
Quantum Chip
...
Bacon-Shor
Color codes
Surface code

Figure 5.1: Overview of the quantum computer system stack

Figure 5.1 provides an overview of the quantum system stack consisting of the following layers [102]:

- The top layers involve the quantum algorithms alongside the language constructs and compilers that are required to generate a series of instructions that belong to the Quantum Instruction Set Architecture (QISA).
- The micro-architecture layer translates these instructions into pulses to operate in the quantum chip. These pulses are sent through the quantum to classical interface.
- As can be seen by the 3^{rd} dimension of Figure 5.1, quantum error correction (QEC) is a key part when building a fault-tolerant quantum accelerator and it affects several layers such as the micro-architecture.

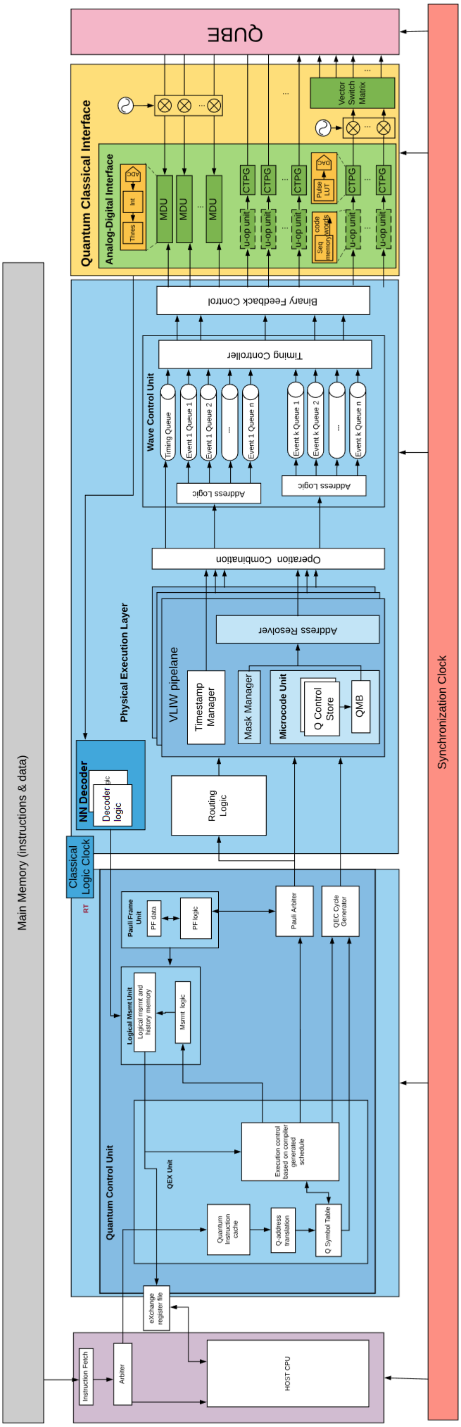

QUBE
Quantum Classical Interface
Analog-Digital Interface
ADC
Int
Thres
MDU
MDU
MDU
CTPG
u-op unit
Seq memory
code words
Pulse LUT
DAC
Vector Switch Matrix
Binary Feedback Control
Timing Controller
Wave Control Unit
Timing Queue
Event 1 Queue 1
Event 1 Queue 2
Event 1 Queue n
Event k Queue 1
Event k Queue 2
Event k Queue n
Address Logic
Address Logic
Operation Combination
Physical Execution Layer
VLIW pipeline
Timestamp Manager
Mask Manager
Microcode Unit
Q Control Store
QMB
Address Resolver
NN Decoder
Decoder logic
Routing Logic
Classical Logic Clock
RT
Quantum Control Unit
Pauli Frame Unit
PF data
PF logic
Pauli Arbiter
QEC Cycle Generator
Logical Msmt Unit
Logical msmt and history memory
Msmt logic
QEX Unit
Execution control based on compiler generated schedule
Quantum Instruction cache
Q-address translation
Q Symbol Table
eXchange register file
Instruction Fetch
Arbiter
HOST CPU
Main Memory (instructions & data)
Synchronization Clock

Figure 5.2: Overview of the quantum micro-architecture [4]

As can be seen in Figure 5.2, different micro-architectural blocks are required to keep track of the errors and identify the location and type of these errors such as the Decoding block and the Pauli frame unit. In this work, we are focused on the Decoder logic block of Figure 5.2 that is part of the QEC process, as we will explain in the next paragraphs.

Constant active quantum error correction (QEC) is regarded as necessary in order to perform reliable quantum computation and storage due to the unreliable nature of current quantum technology. Qubits inadvertently interact with their environment even when no operation is applied, forcing their state to change (decohere). Moreover, application of imperfect quantum gates results in the introduction of errors in the quantum system. Quantum error correction is the mechanism that reverses these errors and restores the state to the desired one.

Quantum error correction involves an encoding and a decoding process. Encoding is used to enhance protection against errors by employing more resources (qubits). In this chapter, we limit ourselves to ***surface code*** encoding [135]. Decoding is the process that is used to identify the location and type of error that occurred. As part of quantum error correction, decoding has a limited time budget that is determined by the time of a single round of error correction. In the case that the decoding time exceeds the time of quantum error correction, either the quantum operations are stalled or a backlog of inputs to the decoding algorithm is created [123]. Many classical decoding algorithms have been proposed with the most widely used being the ***Blossom algorithm*** [76]. Blossom algorithm has been shown to reach high decoding accuracy, but its decoding time scales polynomially with the number of qubits [1], which can be problematic for large quantum systems needed to solve complex problems. However, there are optimized versions of Blossom for topological codes, that report linear scaling with the number of qubits [84] and even a parallel version stating that the average processing time per detection round is constant independent of the size of the system [78]). We propose the employment of neural networks, since they exhibit constant execution time after being trained without any parallelization required and have been proven to provide better decoding performance than many classical decoding algorithms[86–90, 94].

In this work, we investigate various designs of neural network based decoders and compare them in terms of the decoding accuracy, training time and execution time. We analyze how different neural network parameters like the size of the training dataset, the structure and type of the neural network, the batch size and the learning rate used during training, affect the accuracy, the training and execution time of such a decoder.

The rest of the chapter is organized as follows: in section 5.2, we explain how the different designs of neural network decoders work, as found in literature. Section 5.3, presents the guidelines of utilization of the neural network parameters. In section 5.4, we provide the results with the best neural network decoder for the different error models. Finally, in section 5.5, we draw our conclusions about this research. Information regarding quantum error correction, the surface code and neural networks are summarized in chapter 2.

5.2. DESIGNING NEURAL NETWORK BASED DECODERS

Neural network based decoders for quantum error correcting codes have been recently proposed [86–90, 94]. There are two categories in which they can be divided: i) decoders that search for exact corrections at the physical level and ii) decoders that search for corrections that restore the logical state. We are going to refer to the former ones as ***low level decoders*** [86, 87] and the latter ones as ***high level decoders*** [88–90, 94].

Since there are multiple techniques in designing a neural network based decoder, there is merit in figuring out which is the best design strategy. To achieve that, we implemented both designs as found in the literature and investigated their differences and similarities. Furthermore, we evaluate the decoding performance achieved with both decoders and investigate their training and execution time. We provide an analysis for various design parameters.

5.2.1. INPUTS/OUTPUTS

Low level decoders take as input the error syndrome and produce as output an error probability distribution for each data qubit based on the observed syndrome. Therefore, a prediction is made that attempts to correct exactly all physical errors that have occurred.

High level decoders take as input the error syndrome and produce as output an error probability for the logical state of the logical qubit. Based on this scheme, the neural network does not have to predict corrections for all data qubits, rather just for the state of the logical qubit, which makes the prediction simpler. This is due to the fact that there are only 4 options as potential logical errors, $\bar{I}, \bar{X}, \bar{Z}, \bar{Y}$, compared to the case of the low level decoder where the output is equivalent to the number of data qubits. Moreover, trying to correctly predict each physical error requires a level of high granularity which is not necessary for error correcting codes like the surface code.

5.2.2. SAMPLING AND TRAINING PROCESS

During the sampling process, multiple error correction cycles are run and the corresponding inputs and outputs for each decoder are stored. Due to the degenerate nature of the surface code, the same error syndrome might be produced by different sets of errors. Therefore, we need to keep track of the frequency of occurrence of each set of errors that provide the same error syndrome.

For the low level decoder, based on these frequencies, we create an error probability distribution for each data qubit based on the observed error syndrome. For the high level decoder, based on these frequencies, we create an error probability distribution for each logical state based on the observed error syndrome.

When sampling is terminated, we train the neural network to map all stored inputs to their corresponding outputs. Training is terminated when the neural network is able to correctly predict at least 99% of the training inputs. Further information about the training process are provided in section 5.3.

5.2.3. EVALUATING PERFORMANCE

Designs for low level decoders typically include a single neural network. To obtain the predicted corrections for the low level decoder, we sample from the probability distribution that corresponds to the observed error syndrome for each data qubit, and predict whether a correction should be applied at each data qubit. However, this prediction needs to be verified before being used as a correction, because the proposed corrections must generate the same error syndrome as the one that was observed. Otherwise, the corrections are not valid (see Figure 5.3a-b). Only when the two error syndromes match, the predictions are used as corrections on the data qubits (see Figure 5.3a-c). If the observed syndrome does not match the syndrome obtained from the predicted corrections, then the predictions must be re-evaluated by re-sampling from the probability distribution. This re-evaluation step makes the decoding time non-constant, which can be a big disadvantage. There are ways to minimize the average amount of re-evaluations, however this is highly influenced by the physical error rate, the code distance and the strategy of re-sampling.

In Figure 5.3, the decoding procedure of the low level decoder is described with an example. On 5.3a, we present an observed error syndrome shown in red dots and the bit-flip errors on physical data qubits (shown with X on top of them) that created that syndrome. On 5.3b, the decoder predicts a set of corrections on physical data qubits and the error syndrome resulting from these corrections is compared against the observed error syndrome. As can be seen from 5.3a and 5.3b, the two error syndromes do not match therefore the predicted corrections are deemed **invalid**. On 5.3c, the decoder predicts a different set of corrections and the corresponding error syndrome to these corrections is compared against the observed error syndrome. In the case of 5.3a and 5.3c, the predicted error syndrome matches the observed one, therefore the corrections are deemed **valid**.

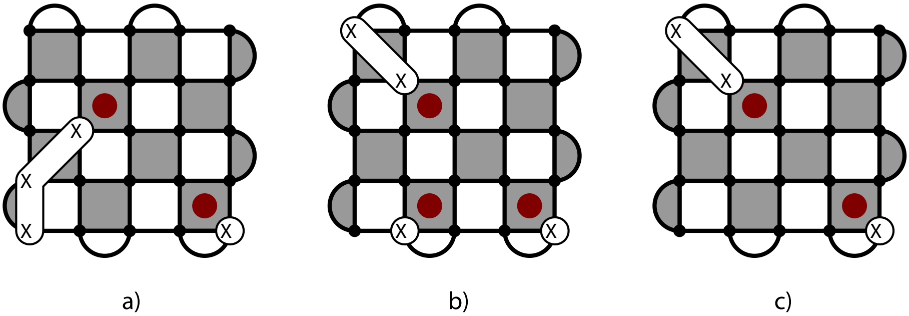


Figure 5.3: Description of the decoding process of the low level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) **Invalid** data qubits corrections and the corresponding error syndrome. (c) **Valid** data qubits corrections and the corresponding error syndrome.

Designs for high level decoders typically involve two decoding modules that work together to achieve high speed and high level of decoding performance. Either both decoding modules can be neural networks [89] or one can be a classical module and the other can be a neural network [88, 90], as described in the previous chapter. The classical module of the latter design will only receive the error syndrome out of the last error correction cycle and predict a set of corrections. In our previous experimentation [88], which is described in the previous chapter, this classical module was called ***simple decoder***. The corrections proposed by the simple decoder do not need to exactly match the errors that occurred, as long as the corrections correspond to the observed error syndrome (valid corrections). The other module which in both cases is a neural network, should be trained to receive the error syndromes out of all error correction cycles and predict whether the corrections that are going to be proposed by the simple decoder are going to lead to a logical error or not. In that case, the neural network outputs extra corrections, which are the appropriate logical operator that erases the logical error. The output of both modules is combined and any logical error created by the corrections of the simple decoder will be canceled due to the added corrections of the neural network (see Figure 5.4).

Furthermore, the simple decoder is purposely designed in the simplest way in order to remain fast, regardless of the quality of proposed corrections. By adding the simple decoder alongside the neural network, the corrections can be given at one step and the execution time of the decoder remains small, since both modules are fast and operate in parallel.

In Figure 5.4, the decoding procedure of the high level decoder is described with an example. On 5.4a, we present an observed error syndrome shown in red dots and the bit-flip errors on physical data qubits (shown with X on top of them) that created that syndrome. On 5.4b, we present the decoding of the classical module known as simple decoder. The simple decoder receives the last error syndrome of the decoding procedure and proposes corrections on physical qubits by creating chains between each detection event and the nearest boundary of the same type as the parity-check type of the detection event. In Figure 5.4b, the corrections on the physical qubits are shown with X on top of them, indicating the way that the simple decoder functions. The simple decoder corrections are always deemed **valid**, due to the fact that the predicted and observed error syndrome always match based on the construction of the simple decoder. In the case of Figure 5.4a-b, the proposed corrections of the simple decoder are going to lead to an $\bar{X}$ logical error, therefore we use the neural network to identify this case and propose the application of the $\bar{X}$ logical operator as additional corrections to the simple decoder corrections, as presented in 5.4c.

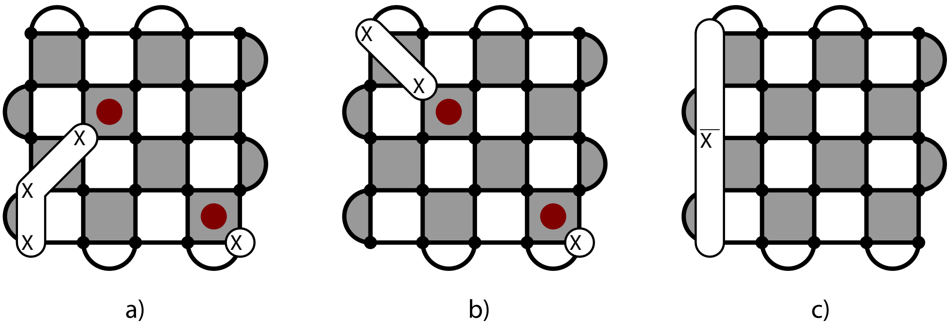


Figure 5.4: Description of the decoding process of the high level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) Corrections proposed by the simple decoder for the observed error syndrome. (c) Additional corrections in the form of the $\bar{X}$ logical operator to cancel the logical error generated from the proposed corrections of the simple decoder.

5.3. IMPLEMENTATION PARAMETERS

In this section, we implement and compare both types of neural network based decoders as discussed in the previous section and argue about the better strategy to create such a decoder. The chosen strategy will be determined by investigation of how different implementation parameters affect the i) decoding performance, ii) training time and iii) execution time.

- The decoding performance indicates the accuracy of the algorithm during the decoding process. The typical way that decoding performance is evaluated is through lifetime simulations. In lifetime simulations, multiple error correction cycles are run and decoding is applied in frequent windows. Depending on the error model, a single error correction cycle might be enough to successfully decode, as in the case of perfect error syndrome measurements (window = 1 cycle), or multiple error correction cycles might be required, as in the case of imperfect error syndrome measurements (window 1 cycle). When the lifetime simulations are stopped, the decoding performance is evaluated as the ratio of the number of logical errors found over the number of windows run until the simulations are stopped.
- The training time is the time required by the neural network to adjust its weights in a way that the training inputs provide the corresponding outputs as provided by the

training dataset and adequate generalization can be achieved.

- The execution time is the time that the decoder needs to perform the decoding after being trained. It is calculated as the difference between the time when the decoder receives the first error syndrome of the decoding window and the time when it provides the output.

5.3.1. ERROR MODEL

These decoders were tested for two error models, the depolarizing error model and the circuit noise model.

The depolarizing model assigns X,Z,Y errors with equal probability $p/3$, known as depolarizing noise, only on the data qubits. No errors are inserted on the ancilla qubits and perfect parity-check measurements are used. Therefore, only a single cycle of error correction is required to find all errors.

The circuit noise model assigns depolarizing noise on the data qubits and the ancilla qubits. Furthermore, each single-qubit gate is assumed perfect but is followed by depolarizing noise with probability $p/3$ and each two-qubit gate is assumed perfect but is followed by a two-bit depolarizing map where each two-bit Pauli has probability $p/15$, except the error-free case, which has a probability of $1-p$. Depolarizing noise is also used at the preparation of a state and the measurement operation with probability p, resulting in the wrong prepared state or a measurement error, respectively. An important assumption is that the error probability of a data qubit error is equal to the probability of a measurement error, therefore d cycles of error correction are deemed enough to decode properly.

5.3.2. CHOOSING THE BEST DATASET

The best dataset for a neural network based decoder is the dataset that produces the best decoding performance. Naively, one could suggest that including all possible error syndromes, would lead to the best decoding performance, however, as the size of the quantum system increases, including all error syndrome becomes impossible. Therefore, we need to include as little but as diverse as possible error syndromes, which will provide the maximum amount of generalization, thus the best decoding performance, after training.

In our previous experimentation[88], we showed that sampling at a single physical error rate that always produces the fewest amount of corrections, is enough to decode small distance rotated surface codes with a decent level of decoding performance. This concept of always choosing the fewer amount of corrections is similar to the Minimum Weight Perfect Matching that Blossom algorithm uses.

After sampling and training the neural network at a single physical probability, the decoder is tested against a large variety of physical error rates and its decoding performance is observed. We call this approach, the ***single probability dataset*** approach, because we create only one dataset based on a single physical error rate and test it against many. Using the single probability dataset approach to decode various physical error probabilities is not optimal, because when sampling at low physical error rates, less diverse samples are collected, therefore the dataset is not diverse enough to correctly generalize to unknown training inputs.

The single probability approach is valid for a real experiment, since in an experiment there is a single physical error probability that the quantum system operates and at that probability the sampling, training and testing of the decoder will occur. However, this is not a good strategy for testing the decoding performance over a wide range of error probabilities. This is attributed to the degenerate nature of the surface code, since different sets of errors generate the same error syndrome. One set of errors is more probable when the physical error rate is small and another when it is high. Based on the design principles of the decoder, only one of these sets of errors, and always the same, are going to be selected when a given syndrome is observed regardless of the physical error rate. Therefore, training a neural network based decoder in one physical error rate and testing its decoding performance in a widely different physical error rate is not beneficial. The main benefit of this approach lies in the fact that only a single neural network has to be trained and used to evaluate the decoding performance for all the physical error rates that were tested. In the single probability dataset approach, the set with the fewer errors was always selected, because this set is more probable for the range of physical error rates that we are interested in.

To avoid such violations, we created different datasets that were obtained by sampling at various physical error rates and trained a different neural network at each physical error rate taken into account. We call this approach, the ***multiple probabilities datasets*** approach. Each dedicated training dataset that was created by a specific physical error probability is used to test the decoding performance at that same physical error probability and the probabilities close to that, but not all physical probabilities tested. Moreover, by sampling, training and testing the performance for the same physical error rate, the decoder has the most relevant information to perform the task of decoding.

The first step when designing a neural network based decoder is gathering data that will be used as the training dataset. However, as the code distance increases, the size of the space including all potential error syndromes gets exponentially large. Therefore, we need to decide at which point the sampling process is going to be terminated.

Based on the sampling probability (physical error rate), different error syndromes will be more frequent than others. We chose to include the most frequent error syndromes in the training dataset. In order to find the best possible dataset, we increase the dataset size until it stops yielding better results in terms of decoding performance. For each training dataset size, we train a neural network and evaluate the decoding performance.

It is not straightforward to claim that the optimal size of a training dataset is found, because there is no way to ensure that we found the minimum number of training samples that provide the best weights for the neural network, therefore generalization, after being perfectly trained. Thus, we rely heavily on the decoding performance that each training dataset achieves and typically use more training samples than the least amount required.

5.3.3. STRUCTURE OF THE NEURAL NETWORK

While investigating the optimal size of a dataset, some preliminary investigation of the structure has been done, however only after the dataset is defined, the structure in terms of layers and nodes is explored in depth (see Figure 5.5).

A variety of different configurations of layers and nodes needs to be tested, so that the configuration with the highest accuracy of training in the least amount of training time can be adopted. The main factors that affect the structure of the neural network are the size of the training dataset, the similarity between the training samples and the type of neural network.

We found in our investigation that the number of layers selected for training are affected more by the samples, e.g. the similarity of the input samples, and less by the size of the training dataset. The number of nodes of the last hidden layer is selected to be equal to the number of output nodes. The rest of the hidden layers were selected to have decreasing number of nodes going from the first to the last layer, but we do not claim that this is necessarily the optimal strategy.

We implemented both decoder designs with feed-forward and recurrent neural net-

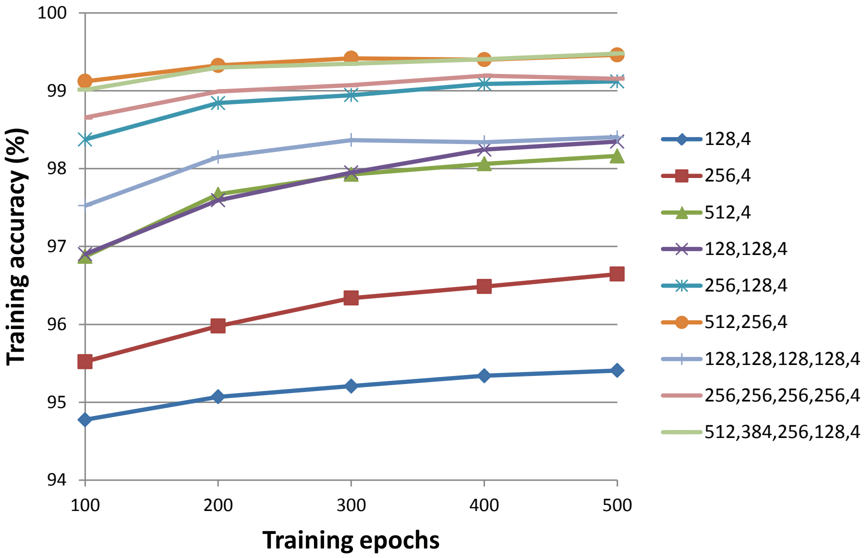


Figure 5.5: Different configurations of layers and nodes for the d=5 rotated surface code for the ***depolarizing error model***. The nodes of the tested hidden layers are presented in the legend. Training stops at 500 training epochs for all configurations, since a good indication of the training accuracy achieved is evident by that point. Then, the one that reached the highest training accuracy continues training until the training accuracy cannot increase any more.

works. The more sophisticated recurrent neural network seems to outperform the feed-forward neural network in both the depolarizing and the circuit noise model. In Figure 5.6, it is evident that even for the small decoding problem of d=3 rotated surface code for the depolarizing error model, the RNN outperforms the FFNN in decoding performance. This is even more obvious at larger code distances and for the circuit noise model, where the recurrent neural network naturally fits better due to its nature. Moreover, training of the FFNN becomes much harder compared to the RNN as the size of the dataset increases, making the experimentation with FFNN even more difficult.

The metric that we use to compare the different designs is the ***pseudo-threshold***. The pseudo-threshold is defined as the highest physical error rate that the quantum device should operate in order for error correction to be beneficial. Operating at higher than the pseudo-threshold probabilities will cause worse decoding performance compared to an unencoded qubit. The protection provided by error correction is increasing as the physical error rate becomes smaller than the pseudo-threshold value, therefore a higher pseudo-threshold for a code distance signifies higher decoding performance.

The pseudo-threshold metric is used when a single code distance is being tested. When a variety of code distances are investigated, then we use the ***threshold*** metric. The threshold is a metric that represents the protection against noise for a family of error correcting codes, like the surface code. For the surface code, each code distance has a different pseudo-threshold value, but the threshold value of the code is only one.

The pseudo-threshold values for all decoders investigated in Figure 5.6 can be found as the points of intersection between the decoder curve and the black dashed line, which represents the points where the physical error probability is equal to the logical error probability

($y = x$). The pseudo-thresholds acquired from Figure 5.6 are presented in Table 5.1.

Table 5.1: Pseudo-threshold values for the tested decoders (d=3) under depolarizing error model

Decoder	Pseudo-threshold
FFNN lld	0.0911
RNN lld	0.0949
FFNN hld	0.0970
RNN hld	0.0969
Blossom	0.0825

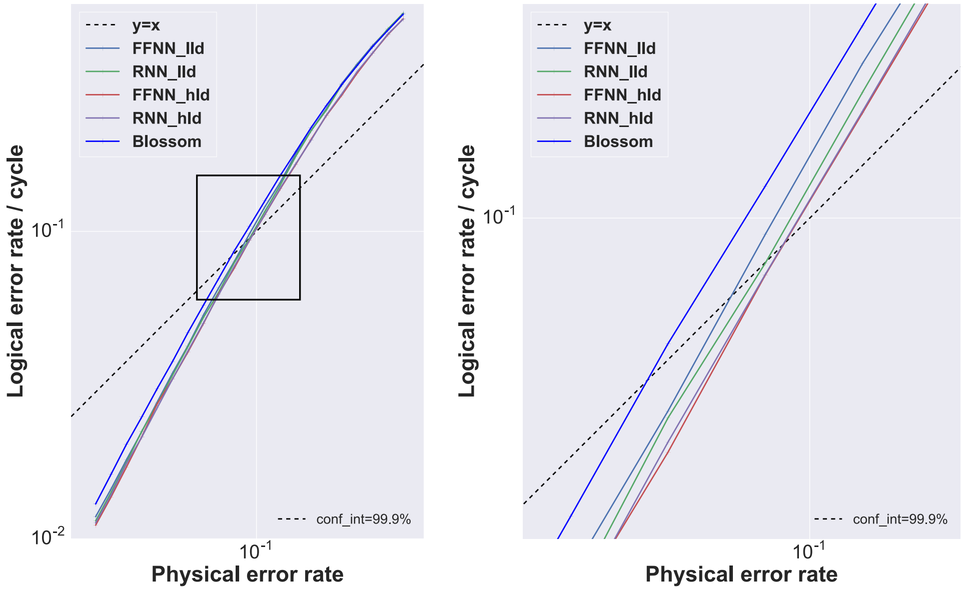


Figure 5.6: Left: Comparison of decoding performance between Blossom algorithm, low level decoder and high level decoder for the ***d=3*** rotated surface code for the ***depolarizing error model***. Right: Zoomed in at the region defined by the square.

The threshold value is defined as the point of intersection of all the curves of multiple code distances, therefore cannot be seen from Figure 5.6, since all curves involve d=3 decoders, but can be found in Figures 5.9 and 5.10 for the depolarizing and circuit noise model, respectively.

Another observation from Figure 5.6 and Table 5.1 is that the high level decoder is outperforming the low level decoder. Although there are ways to increase the decoding performance of the latter, mainly by re-designing the repetition step to find the valid corrections in less repetitions, we found no merit in doing so, since the decoding performance would still be similar to the high level decoder's and the repetition step would still not be eliminated.

Based on these observations, the results presented in Figures 5.9 and 5.10 in the Results section were obtained with the high level decoder with recurrent neural networks.

5.3.4. TRAINING PROCESS

BATCH SIZE

Training in batches instead of the whole dataset at once, can be beneficial for the training accuracy and training time. By training in batches, the weights of the neural network are updated multiple times per training iteration, which typically leads to faster convergence. We used batches of 1000 or 10000 samples, based on the size of the training dataset.

LEARNING RATE

Another important parameter of training that can directly affect the training accuracy and training time is the learning rate. The learning rate is the parameter that defines how big the updating steps will be for each weight at every training iteration. Larger learning rates in the beginning of training can lead the training process to a minimum faster during gradient descent, whereas smaller learning rates near the end of training can increase the training accuracy. Therefore, we devise a strategy of a step-wise decrease of the learning rate throughout the training process. If the training accuracy has not increased after a specified number of training iterations (e.g. 50), then the learning rate is decreased. The learning rates used range from 0.01 to 0.0001.

GENERALIZATION

The training process should not only be focused on the correct prediction of known inputs, but also the correct prediction of inputs unknown to training, known as generalization. Without generalization, the neural network acts as a Look-Up Table (LUT), which will lead to sub-optimal behavior as the code distance increases. In order to achieve high level of generalization, we continue training until the closeness between the desired and predicted value up to the 3^{rd} decimal digit is higher than 95% over all training samples.

TRAINING AND EXECUTION TIME

Timing is a crucial aspect of decoding and in the case of neural network decoders we need to minimize both the execution time and the training time as much as possible.

The training time is proportional to the size of the training dataset and the number of qubits. The number of qubits is increasing in a quadratic fashion, $2d^2 - 1$, and the selected size of the training dataset in our experimentation is increasing in an exponential way, 2^{d^2-1}. Therefore, training time should increase exponentially while scaling up.

However, the platform that the training occurs, affects the training time immensely, since training in one/multiple CPU(s) or one/multiple GPU(s) or a dedicated chip in hardware will result in widely different training times. The neural networks that were used to obtain the results in this work, required between half a day to 3 days, depending on the number of weights and the inputs/outputs of the neural network, on a CPU with 28 hyper thread cores at 2GHz with 384GB of memory.

In our simulations in a CPU, we observed the constant time behavior that was anticipated for the execution time, however a realistic estimation taking into account all the details of a hardware implementation that such a decoder might run, has not been performed by this or any other group yet. The time budget for decoding is different for different qubit technologies, however due to the inadequate fidelity of the quantum operations, it is extremely small for the time being, for example ~700nsec for a surface code cycle with superconducting qubits [71].

In Figure 5.7, we present the constant and non-constant execution time for the d=3 rotated surface code for the depolarizing noise model with perfect error syndrome measurements for the high level decoder and the low level decoder, respectively.

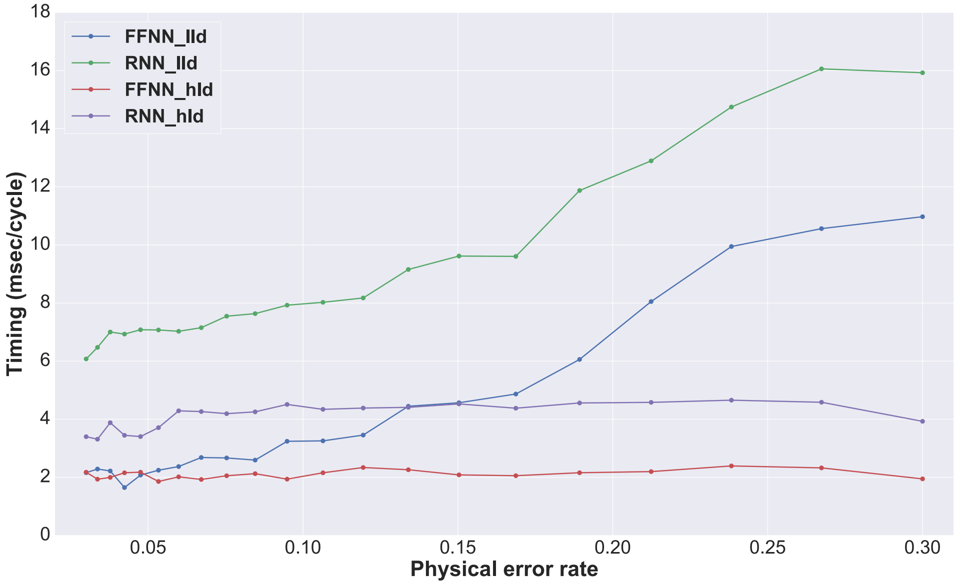


Figure 5.7: Execution time for the high level decoder (hld) and the low level decoder (lld) for Feed-forward (FFNN) and Recurrent neural networks (RNN) for d=3 rotated surface code for the ***depolarizing error model***.

The low level decoder has to repeat its predictions before it predicts a valid set of corrections which makes the execution time non-constant. With careful design of the repetition step, the average number of predictions can decrease, however the execution time will remain non-constant. Based on the non-constant execution time and the inferior decoding performance compared to the high level decoder, the low level decoder was rejected.

Moreover, the recurrent neural network typically uses more weights compared to the feed-forward neural network, which translates to higher execution time. However, the decoding performance and the training accuracy achieved with recurrent neural networks leads to better decoding performance. Thus, we decided to create high level decoders based on recurrent neural networks while taking into account all the parameters mentioned above.

The execution time for high level decoders appears to increase linearly with the number of qubits. This is justified by the fact that as the code distance increases, the operation of the simple decoder does not require more time, since all detection events are matched in parallel and independently to each other, and the size of the neural network increases in such a way that only a linear increase in the execution time is calculated. In Table 5.2, we provide the calculated average time of decoding a surface code cycle under depolarizing noise for all distances tested with the high level decoder with recurrent neural networks.

We are focusing on the scaling of the execution time, instead of the actual value, since these simulations were run as a Python code in a CPU, completely unoptimized. We are confident that a hardware implementation will perform fast enough in a realistic case.

There are factors such as the number of qubits, the type of neural network being used and the number of inputs/outputs of the neural network that influence the execution time. The main advantage against classical algorithms is that the execution time of such neural network based decoders is independent of the physical error probability.

Table 5.2: Average time for surface code cycle under depolarizing error model

Code distance	Avg. time / cycle
d=3	4.14msec
d=5	11.19msec
d=7	28.53msec
d=9	31.34msec

In the following section we are presenting the results in terms of the decoding performance for different code distances.

5.4. RESULTS

As we previously mentioned, the way that decoding performance is tested is by running simulations that sweep a large amount of physical error rates and calculate the corresponding logical error rate for each of them. This type of simulations are frequently referred to as lifetime simulations and the logical error is calculated as the ratio of logical errors found over the error correction cycles performed to accumulate these logical errors.

The design of the neural network based decoder that was used to obtain the results is described in Figure 5.8 for the depolarizing and the circuit error model. For the case of the ***depolarizing error model***, neural network 1 is not used, so the input is forwarded directly to the simple decoder since perfect syndrome measurements are assumed. The decoding process is similar to the one presented in Figure 5.4.

The decoding algorithm for the ***circuit noise model*** consists of a simple decoder and 2 neural networks. Both neural networks receive the error syndrome as input. Neural network 1 predicts which detection events at the error syndrome belong to data qubit errors and which belong to measurement errors. Then, it outputs the error syndrome relieved of the detection events that belong to measurement errors to the simple decoder. The simple decoder provides a set of corrections based on the received error syndrome. Neural network 2 receives the initial error syndrome and predicts whether the simple decoder will make a logical error and outputs a set of corrections which combine with the simple decoder corrections at the output.

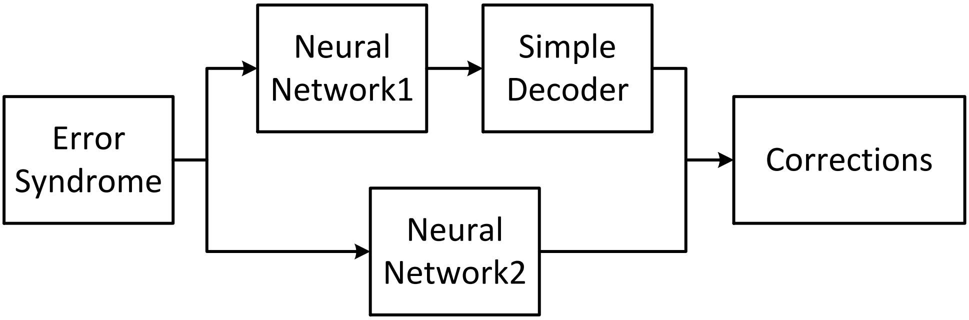


Figure 5.8: The design for the high level decoder that was used for the depolarizing and the circuit noise model.

5.4.1. DEPOLARIZING ERROR MODEL

For the depolarizing error model, we used 5 training datasets that were sampled at these physical error rates : 0.2, 0.15, 0.1, 0.08, 0.05. Perfect error syndrome measurements are assumed, so the logical error rate can be calculated per error correction cycle.

In Table 5.3, we present the pseudo-thresholds achieved from the investigated decoders for the depolarizing error model with perfect error syndrome measurements for different distances. As expected, when the distance increases, the pseudo-threshold also increases. Furthermore, the neural network based decoder with the multiple probabilities datasets exhibits higher pseudo-threshold values, which is expected since it has more relevant information in its dataset.

Table 5.3: Pseudo-threshold values for the depolarizing error model

Decoder	d=3	d=5	d=7	d=9
Blossom	0.08234	0.10343	0.11366	0.11932
Single prob. dataset	0.09708	0.10963	0.12447	N/A
Multiple prob. dataset	0.09815	0.12191	0.12721	0.12447

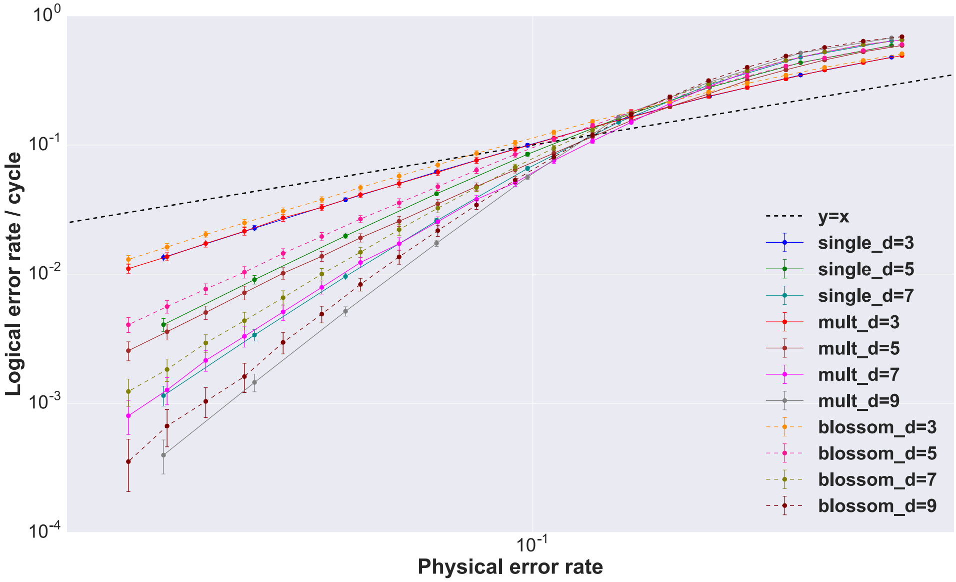


Figure 5.9: Decoding performance comparison between the high level decoder trained on a single probability dataset, the high level decoder trained on multiple probabilities datasets and Blossom algorithm for the ***depolarizing error model*** with perfect error syndrome measurements. Each point has a confidence interval of 99.9%.

As can be seen from Figure 5.9, the multiple probabilities datasets approach is providing better decoding performance for all code distances simulated. The fact that the high level decoder is trained to identify the most frequently encountered error syndromes based on a given physical error rate, results in more accurate decoding information. Another reason for the improvement against the Blossom algorithm, is the ability of identifying correlated errors (-iY=XZ). For the depolarizing noise model with perfect error syndrome measurements, the Blossom algorithm is proven to be near-optimal, so we are not able to observe a large improvement in the decoding performance. Furthermore, the comparison is against the unoptimized version of Blossom algorithm [77], therefore it is mainly performed to get a frame of reference rather than an explicit numerical comparison.

We observe that for the range of physical error rates that we are interested in, which are below the pseudo-threshold, the improvement against Blossom algorithm is reaching up to 18.7%, 58.8% and 53.9% for code distance 3, 5 and 7, respectively for the smallest physical error probabilities tested.

The threshold of the rotated surface code for the depolarizing model has improved from 0.14 for the single probability dataset approach to 0.146 for the multiple probabilities datasets approach. The threshold of Blossom is calculated to be 0.142.

5.4.2. CIRCUIT NOISE MODEL

For the circuit noise model, we used 5 training datasets that were sampled at these physical error rates : 4.5x10^{-3}, 1.5x10^{-3}, 8.0x10^{-3}, 4.5x10^{-4}, 2.5x10^{-4}. Since, imperfect error syndrome measurements are assumed the logical error rate is calculated per window of d error correction cycles.

In Table 5.4, we present the pseudo-thresholds achieved for the circuit noise model with imperfect error syndrome measurements. Again, the neural network based decoder with multiple probabilities datasets is performing better than the single probability dataset. We were not able to use the Blossom algorithm with imperfect measurements for code distances higher than 3, therefore we decided not to include it. However, we note that the results that were obtained are similar to the results in the literature corresponding to the circuit noise model [2, 83].

Table 5.4: Pseudo-threshold values for the circuit noise model

Decoder	d=3	d=5	d=7
Single prob. dataset	3.99x10^{-4}	9.23x10^{-4}	N/A
Multiple prob. dataset	4.44x10^{-4}	1.12x10^{-3}	1.66x10^{-3}

We observe from Figure 5.10 that the results with the multiple probabilities datasets for the circuit noise model are significantly better, especially as the code distance is increased. The case of the d=3 is small and simple enough to be solved equally well by both approaches. The increased decoding performance achieved with the multiple probabilities datasets approach is based on the more accurate information for the physical error probability that is being tested.

The threshold of the rotated surface code for the circuit noise model has improved from 1.77x10^{-3} for the single probability dataset approach to 3.2x10^{-3} for the multiple probabilities datasets approach, that signifies that the use of dedicated datasets when decoding a given physical error rate is highly advantageous.

As mentioned, the single probability dataset is collected at a low physical error rate, for example around the pseudo-threshold value. Therefore, the size of the training dataset is similar for both the single and the multiple probabilities datasets for the low physical error rates. For higher physical error rates, we gather larger training datasets for the multiple probabilities datasets approach, which are also more relevant.

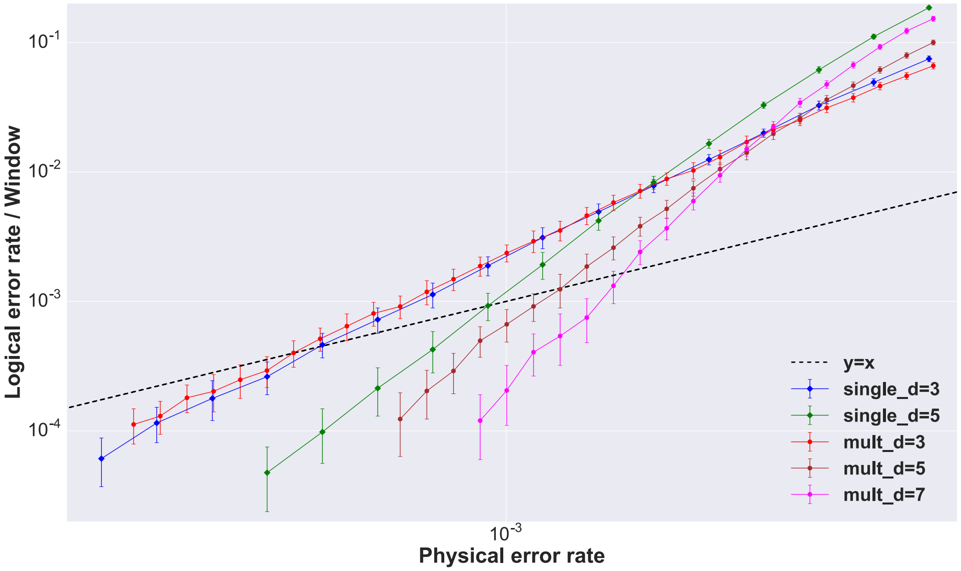


Figure 5.10: Decoding performance comparison between the high level decoder trained on a single probability dataset and the high level decoder trained on multiple probabilities datasets for the ***circuit noise model*** with imperfect error syndrome measurements. Each point has a confidence interval of 99.9%.

The space that needs to be sampled is getting exponentially larger to a point that is infeasible to gather enough samples to perform good decoding beyond d=7. The reason for this exponential growth is due to the way that we provide the data to the neural network. Currently, we gather all error syndromes out of all the error correction cycles and create lists out of them. Then, we provide these lists to the recurrent neural network all-together. Since the recurrent neural network can identify patterns both in space and time, we also provide the error correction cycle that provided that error syndrome (time stamp of each error syndrome). Then, the recurrent neural network is able to differentiate between consecutive error correction cycles and find patterns of errors in them.

In order to obtain efficient decoding regardless of the exponentially large state space, we restrict the space that we sample to the one containing the most frequent error syndromes occurring at the specified sampling (physical) error probability. However, even by employing such a technique, it seems impossible to continue beyond d=7 for the circuit noise model with the decoding approach that we used in this work. At the circuit noise model for d=7 for example, we gather error syndromes out of 10 error correction cycles and each error syndrome contains 48 ancilla qubits. Therefore, the full space that needs to be explored is 2^{10*48}, which is infeasible.

A different approach that minimizes the space that the neural network needs to search would be extremely valuable. A promising idea would be to provide the error syndromes of each error correction cycle one at a time, instead of giving them all-together, and keep an intermediate state of the logical qubit.

5.5. CONCLUSIONS

This work focused on researching various design strategies for neural network based decoders. Such kind of decoders are currently being investigated due to their good decoding performance and constant execution time. They seem to have an upper limit at around 160 qubits, however by designing smarter approaches in the future, we can have neural network based decoders for larger quantum systems.

We emphasized mainly on the design aspects and the parameters that affect the performance of the neural networks and devised a detailed plan on how to approach them. We showed that we can have high decoding performance for quantum systems of about 100 qubits for both the depolarizing and the circuit noise model. We showed that a neural network based decoder that uses the neural network as an auxiliary module to a classical decoder leads to higher decoding performance.

Furthermore, we presented the constant execution time of such a decoder and showed that it increases linearly with the code distance in our simulations. We compared different types of neural networks, in terms of decoding performance and execution time, concluding that recurrent neural networks can be more powerful than feed-forward neural networks for such applications.

Finally, we showed that having a dedicated dataset for the physical error rate that the quantum system operates can increase the decoding performance.

The contents of this chapter are based on the following paper:

S. Varsamopoulos, K. Bertels, C. G. Almudever, **Designing neural network based decoders for surface codes**, *arXiv:1811.12456*, 2018.

6

DECODING SURFACE CODE WITH A DISTRIBUTED NEURAL NETWORK BASED DECODER

There has been a rise in decoding quantum error correction codes with neural network based decoders, due to the good decoding performance achieved and adaptability to any noise model. However, the main challenge is scalability to larger code distances due to an exponential increase of the error syndrome space. Note that, successfully decoding the surface code under realistic noise assumptions will limit the size of the code to less than 100 qubits with current neural network based decoders.

Such a problem can be tackled by a distributed way of decoding, similar to the Renormalization Group (RG) decoders. In this chapter, we introduce a decoding algorithm that combines the concept of RG decoding and neural network based decoders. We tested the decoding performance under depolarizing noise with noiseless error syndrome measurements for the rotated surface code and compared against the Blossom algorithm and a neural network based decoder. We show that similar level of decoding performance can be achieved between all tested decoders while providing a solution to the scalability issues of neural network based decoders.

6.1. INTRODUCTION

Quantum error correction (QEC) is for now considered to be the most time and resource consuming procedure in quantum computation. However, the way that quantum computing is currently envisioned, QEC is necessary for reliable quantum computation and storage. The need for QEC arises from the unavoidable coupling of the quantum system with the environment, which causes the qubit state to be altered (decohere). Altering the quantum state is perceived as errors generated in the quantum system. Through active error correction and fault-tolerant mechanisms, that control error propagation and keep the error rates low, we can have the error-free desired state. Note that, in ***fault-tolerant*** techniques, errors can occur in the quantum system, but do not affect the quantum state in a catastrophic manner [42].

A critical sub-routine of QEC is ***decoding***. Decoding involves the process of identifying the errors that occur in the quantum system and proposing corrections that keep the quantum state error-free. The importance of high speed and accurate decoding lies in the fact that the time budget allowed for error correction is small, since qubits lose their state rapidly. Therefore, if the process of decoding exceeds the error correction time budget, errors will accumulate to the point that the error-free state cannot be retrieved.

Various classical decoding algorithms have been proposed over the years with a few examples of classical decoding algorithms being the Blossom algorithm [76–78, 84], the maximum-likelihood algorithm [75] and the Renormalization Group (RG) algorithm [81, 82]. Recently, there is an increase in the development of ***neural network based decoders*** that either consist exclusively of neural networks [86, 87] or a classical module working together with neural networks [88–91, 93]. Neural network based decoders exist with different designs in the way the decoding is performed and a variety of types of neural networks has been explored, like Feed-forward, Recurrent and Convolutional neural networks.

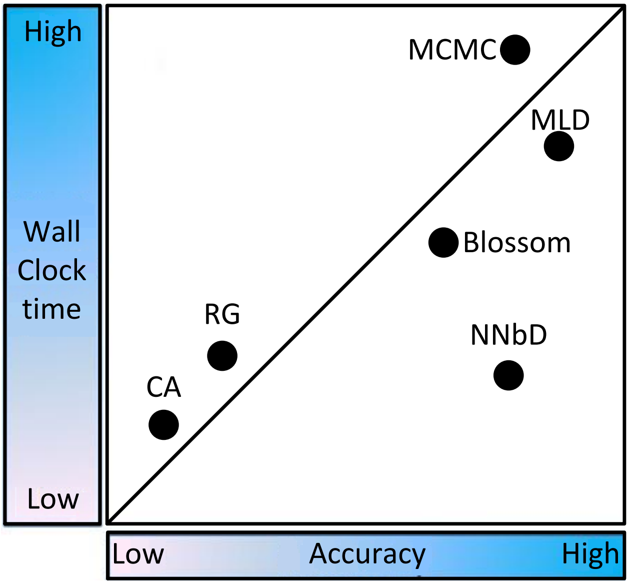


Figure 6.1: Abstract comparison between decoding performance and execution time of various decoding algorithms

In Figure 6.1 we present an abstract comparison between various decoding algorithms based on their decoding performance (Accuracy) and their execution time (Wall clock time), namely the Markov Chain Monte Carlo (MCMC) [79], the Maximum Likelihood Decoder (MLD) [75], the Minimum Weight Perfect Matching (MWPM) [76, 77] that Blossom algorithm is based on, the Neural Network based Decoder (NNbD) [92], the Renormalization Group (RG) [82] and the Cellular Automaton (CA) [136]. Decoding performance is typically calculated as the ratio of the number of logical errors created out of the decoder corrections over the number of error correction cycles run to accumulate these errors. Execution time is defined as the time spent from the moment that the input data arrive at the decoder until the time that the decoder proposes the corrections. As can be seen from Figure 6.1, neural network based decoders can reach equivalent decoding performance as classical algorithms while requiring smaller execution time. This is the main reason that neural network based decoders are explored and various designs have been proposed recently. However, the main issue with such decoders is that scaling to larger quantum systems will be significantly harder compared to classical decoders, due to the required training process of the neural network. As the size of the system increases, more training samples need to be collected and then the neural network has to be trained based on them. The main challenge of NNbDs is that in order to reach similar decoding performance to classical algorithms as the quantum system is increasing, the amount of samples required to be collected increases in an exponential way, which makes the training harder and slower.

In this work, we will present a neural network based decoder that performs decoding in a distributed fashion, therefore providing a solution for the issue of decoding large codes. We should mention that there exist classical algorithms that perform decoding in a distributed way, as can be found in [82] and [78], but in this chapter we will provide a different approach of the distributed decoding concept. In [82], the original idea of RG decoding approach is described and tested. RG decoding is based on the division of the code into small tiles, in which a given number of physical qubits are included and error probabilities about the physical qubits inside all tiles are calculated. Then, these tiles are grouped into larger tiles and the error probabilities about the qubits are updated. This procedure is continued until only a single tile has remained containing all the physical qubits of the system. Based on the updated error probabilities of the largest tile, the decoder can propose a set of corrections. In [78], a distributed decoding approach is described, where the code is divided into small tiles. However, in this case Blossom algorithm is used to decode each tile and based on the result of it and the neighboring information between the tiles, the decoder can propose corrections for the whole code. Each tile is monitored by an Application-Specific Integrated Circuit (ASIC), which is dedicated for the tile.

In our strategy, the code is divided into small overlapping regions, referred to as ***overlapping tiles***, where local information about errors on physical qubits is obtained. Then, this local information is combined and a decoding for the whole code is obtained. We compare our algorithm to the unoptimized version of Blossom algorithm [76, 77] and argue about the decoding performance achieved. Furthermore, we will provide reasoning for the potential high level of parallelization of our algorithm that will be suitable for a high speed hardware implementation without loss of decoding performance. Also, the problem of the exponential increase of the error syndrome space is mitigated, since it is controlled by the selection of the size of the decoded regions. This allows neural network based decoders to successfully decode larger codes.

The rest of the chapter is organized in the following way: in section 6.2 we give a short introduction in the concept of RG decoding. In section 6.3, we present the design of the distributed neural network based decoder and in section 6.4, we provide the results in terms of decoding performance. Finally, in section 6.5, we draw our conclusions about the distributed decoding approach. Information regarding quantum error correction and the surface code are summarized in chapter 2.

6.2. RG DECODING

Our previous efforts were mainly focused on developing neural network based decoders that can achieve better decoding performance than classical decoding algorithms and report a constant execution time for each code distance for all range of physical error probabilities, which scales linearly with the code distance [92]. However, good decoding performance was harder to achieve as the code distance increased. The main problem was the exponential increase of the error syndrome space, which required an immensely large number of training samples in order for the decoder to achieve similar performance to the classical decoding algorithms for d9. We provide the size of the training datasets used for the code distances investigated in [92] for the depolarizing error model in Table 6.1.

Table 6.1: Size of training datasets

code distance	selected dataset size	full dataset size
d=3	256	2^8
d=5	$6 * 10^5$	2^{24}
d=7	$5 * 10^6$	2^{48}
d=9	$2 * 10^7$	2^{80}

A way that the error space can be limited, is through a distributed way of decoding similar to the RG algorithm. By dividing the code in small regions which are going to provide individual information about decoding every region of the code, the decoder can have enough information about decoding the whole code. Limiting the region that we want to locally decode, the error syndrome space is also limited, allowing us to increase the distance of the code without changing the decoding of each region.

RG decoding is similar to decoding concatenated codes, which have various levels of encoding, as can be seen at Figure 6.2.

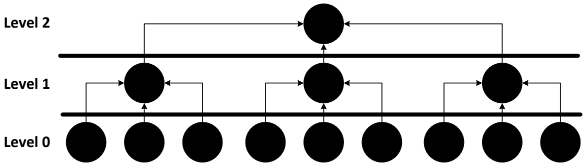


Figure 6.2: Encoding levels of a concatenated code. At level 0 there are nine qubits, that are encoded in three qubits at level 1 and these qubits are encoded in one qubit at level 2. Arrows show the information flow.

In these codes, decoding is achieved by passing the error information concerning the qubits from the lower level to the higher level. The information about errors is updated throughout the encoding levels. The decoding occurs at the last encoding level and a final decision about the logical state is made.

The strategy of RG decoding can be described according to Figure 6.3. At first, the lattice is cut in small (green) tiles and the probability of an error occurring in all qubits included in that tile is evaluated. After gathering the updated error probabilities in the green tiles, the lattice is cut into bigger (red) tiles and the error probability of all qubits included in that tile is evaluated. This process is continued until there is only one tile left that includes all qubits in the code.

The same approach can be applied to surface code. However, the challenge here is that the parity-checks cannot be broken down into constant size tiles in a way that every parity-check corresponds to a single tile. Therefore, we need to use overlapping tiles, which will always include whole parity-checks of the code in a single tile. The boundary qubits that belong to neighboring tiles are treated as independent variables on each tile and the error probability for the same qubit is different depending on the tile. The way that the error probabilities are usually calculated is by belief propagation [81, 82] in the RG approach.

We decided to use the idea of overlapping tiles, but follow a different approach than the RG algorithm as we will explain in the following section.

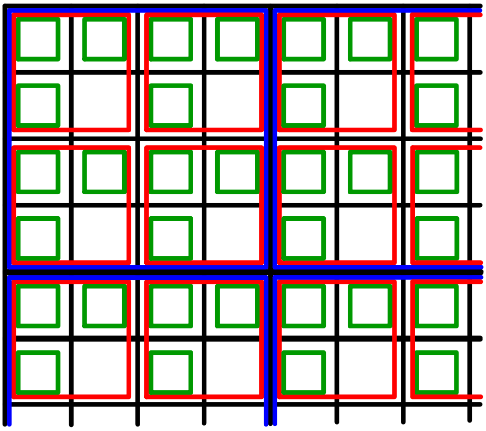


Figure 6.3: Tile segmentation that represents the levels of concatenation in a concatenated code. The smallest level of concatenation is represented by the green tiles, the next level of concatenation is represented by the red tiles, the following level of concatenation is represented by the blue tiles, etc.

6.3. DISTRIBUTED DECODING WITH OVERLAPPING TILES

We developed a neural network based decoder that performs distributed decoding based on the concept of RG decoders. As mentioned, the main idea behind this algorithm is to make neural network based decoders able to successfully decode large code distances. By restricting the decoding in small regions (tiles) of the lattice, the decoder does not have to explore a large error syndrome space, rather just decode every small tile and then combine the information out of all tiles.

The main difference between a distributed neural network based decoder and the RG decoder is that the former only has one level of concatenation. Instead of moving from smaller tile to bigger tile until the whole lattice is a single tile, we segment the lattice into small equally sized tiles that are overlapping with each other, so that each tile includes whole parity-checks of the code. Then, we obtain error information from each individual tile and combine the information out of all tiles to get the error information for the whole lattice. In this case, there is no need to calculate the error probability of all qubits and forward it to the next level of concatenation, rather find a way to combine the information arising from the each tile.

In order to decode based on the distributed decoding approach, we will use the same two-module decoder as was presented in [92]. Our decoding algorithm consists of two modules, a classical decoding module that we call ***simple decoder*** and a neural network. The simple decoder provides a naive decoding for the whole lattice, in which a chain is created between each detection event and its closest boundary of the same type. The corrections arising from the simple decoder occur in the data qubits underneath the chain. An example is provided in Figure 6.4, where AZ5 and ancilla AX4 have indicated the presence of an error in their proximity. The proposed corrections of the simple decoder will be Z5, Z11 arising from ancilla AX4 and X3, X7 arising from ancilla AZ5.

The simple decoder receives the error syndrome for the whole lattice and provides a set of corrections for the whole lattice. This is a fast process since the corrections arising from each detection event are independent from the corrections arising from other detection events, therefore can be parallelized. However, the simple decoder cannot yield high

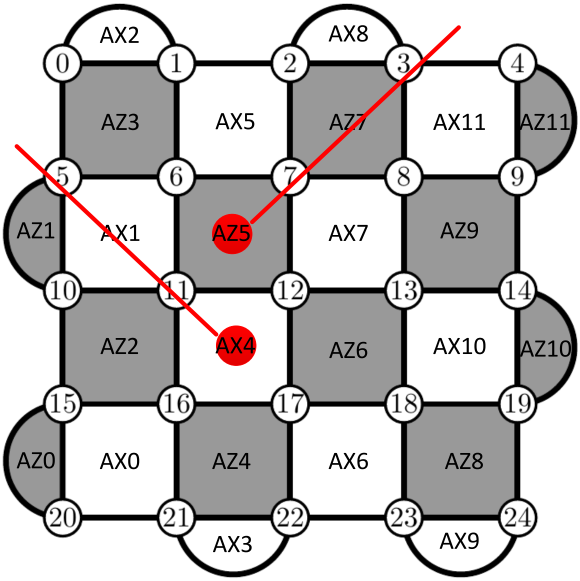


Figure 6.4: Description of the simple decoder operation for the rotated surface code with distance 5. Detection events are presented with the red dots. Red lines indicate which data qubits are going to be corrected.

decoding accuracy on its own, due to its simplistic design.

That is why we also include the neural network that will work as a supervisor to the simple decoder. More accurately, the neural network will be trained to identify for which error syndromes the simple decoder will lead to a logical error. In the case where a logical error will be created out of the simple decoder corrections, the neural network will output the appropriate logical operator that will cancel the logical error out. As we showed in [92], the combination of these two modules will provide high decoding performance.

In order to train the neural network, we create a training dataset by running surface code cycles and storing the error syndrome and the corresponding logical state of the logical qubit after the corrections of the simple decoder are applied. The size of the training dataset varies based on the code distance and the error model. For more information about all the parameters that affect the dataset, we refer the reader to our previous work [92].

In Figure 6.5, we provide an example of the segmentation of a d=5 rotated surface code into four overlapping tiles of d=3 rotated surface codes.

As can be seen from Figure 6.5, each parity-check is included in at most two tiles. The error syndrome obtained for the whole lattice (d=5) is broken down into parts of the error syndrome that correspond to each small tile (d=3). The error syndrome out of one surface code cycle consists of 24 bits, due to the 24 parity-checks of the d=5 code. The error syndrome will be cut into smaller parts of the initial error syndrome that fit the d=3 tiles. Due to inclusion of the shared parity-checks, the bits that are available out of the four d=3 tiles are now 32. Each error syndrome of the d=3 tile corresponds to a part of the complete error syndrome. The error probabilities of the logical state, Prob(I), Prob(X), Prob(Z), Prob(Y), that are associated with the given tile are averaged and the probabilities for the logical state of each tile is provided. Then, the 4 probabilities concerning the logical state of each d=3 tile

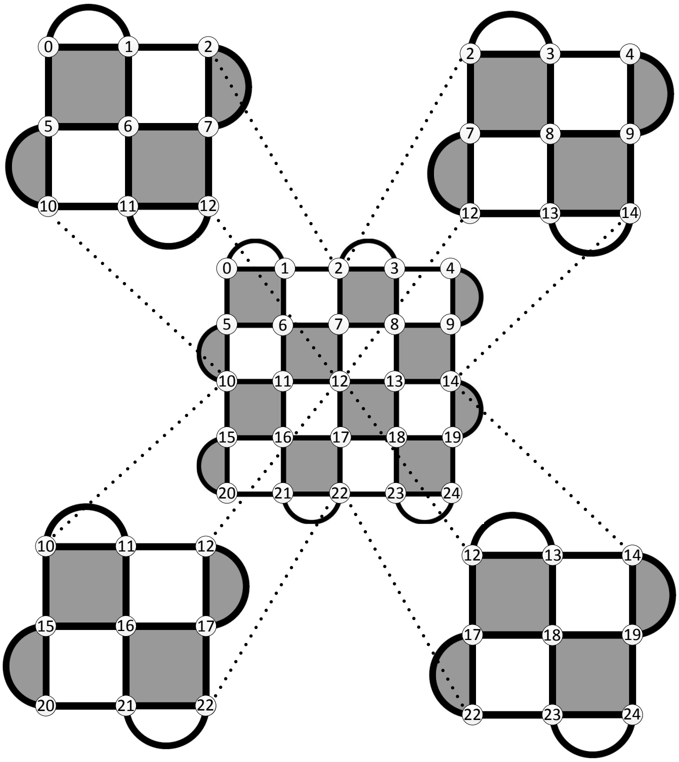


Figure 6.5: Segmentation of a d=5 rotated surface code into four overlapping tiles of d=3 rotated surface codes.

are used as the inputs of the neural network, which will provide at the output the probabilities of the logical state for the whole lattice. Based on the output of the neural network, extra corrections are going to be applied in the form of the appropriate logical operator to cancel any potential logical error created by the simple decoder. The information contained in the 32 bits of the d=3 tiles is now compressed to 16 bits that constitute the inputs of the neural network and represent the probabilities of contribution to the logical state out of every d=3 tile.

6.4. RESULTS

In order to check whether the distributed decoding algorithm can reach similar decoding performance as the other popular decoding algorithms, we tested it against an unoptimized version of the Blossom algorithm [76, 77] and our previous implementation of neural network based decoder [92] for the depolarizing error model with noiseless error syndrome measurements.

The depolarizing error model assumes errors only on the data qubits and perfect error syndrome measurements. Bit-flip (X) errors, phase-flip (Z) errors and both bit- and phase-flip (Y) errors are assumed to be generated with equal probability of $p/3$. Such a simplistic error model is enough to prove that the distributed decoding algorithm that we propose can reach similar decoding performance to other decoding algorithms and that the scalability issues of neural network based decoder are addressed.

The critical aspect of our decoder is the choice of the size of the overlapping tiles. Since, there is only one level of concatenation, contrary to RG decoding, the size of the overlapping tiles plays a significant role in the algorithm. Having a large tile size might provide better decoding, for example decoding a d=9 surface code with d=7 tiles might be more beneficial than decoding with d=3 tiles, since there will be less shared parity-checks and long error chains will be included in a single tile. However, the bottleneck that will make such a case decode poorly in our design, is the inability of the decoder to handle properly the error syndromes unknown to the training dataset. Since it becomes exponentially harder to gather all the possible error syndromes as the code distance increases, the training dataset will be an incomplete set of all potential cases. In the case of an unknown to the training error syndrome, the neural network will not have any meaningful data to make a prediction making the behavior of the neural network inconsistent. Such a case occurs because there is an intermediate step between the cutting of the error syndrome into parts and the averaging of the probabilities of each part.

Based on that, we opted to always divide the lattice into d=3 overlapping tiles, since the d=3 case only consists of 256 different error syndromes. This is an easily obtained complete training dataset, to which any part of error syndrome of any large distance can deconstruct to. All possible error syndromes of the large lattice (d>3) are represented through the d=3 overlapping tiles, without having to explicitly sample all possible error syndromes for the large lattice.

The only downside of using d=3 tiles is that there exist some error syndromes that are highly ambiguous to what logical state they lead. Fortunately, these ambiguous error syndromes are not extremely frequent making the errors arising from this shortcoming rare.

Another benefit of the distributed decoding approach is that the number of inputs required by the neural network is decreased compared to decoding the whole lattice approach. The reduction of inputs of the neural network for the code distances tested are shown in Table 6.2.

Table 6.2: Reduction in required inputs of the neural network

Code distance	Old inputs	New inputs
d=5	24	16
d=7	48	36
d=9	80	64

The comparison of the decoding performance between the distributed decoding, the neural network based decoder from [92] and unoptimized version of the Blossom algorithm for a distance 5, 7 and 9 rotated surface code are presented in Figure 6.6, 6.7 and 6.8, respectively. Each point in these graphs has a confidence interval of 99.9%.

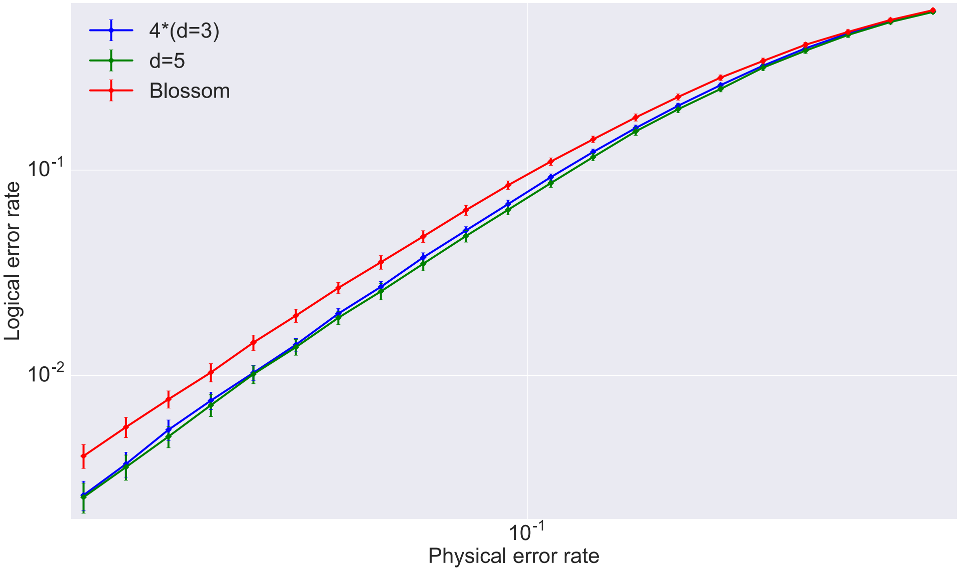


Figure 6.6: Comparison of decoding performance between the distributed decoder with ***four*** overlapping tiles of d=3 rotated surface codes inside a d=5 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green).

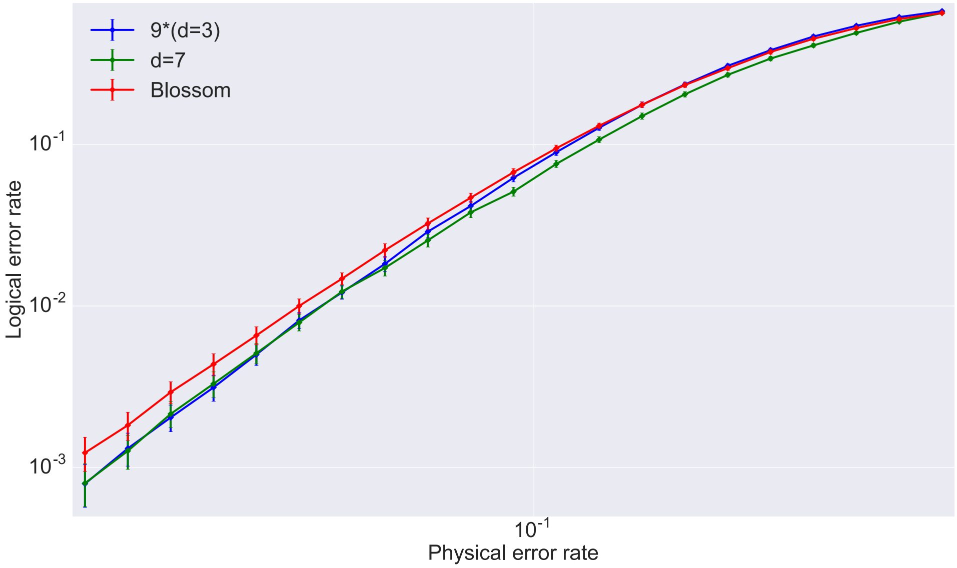


Figure 6.7: Comparison of decoding performance between the distributed decoder with ***nine*** overlapping tiles of d=3 rotated surface codes inside a d=7 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green).

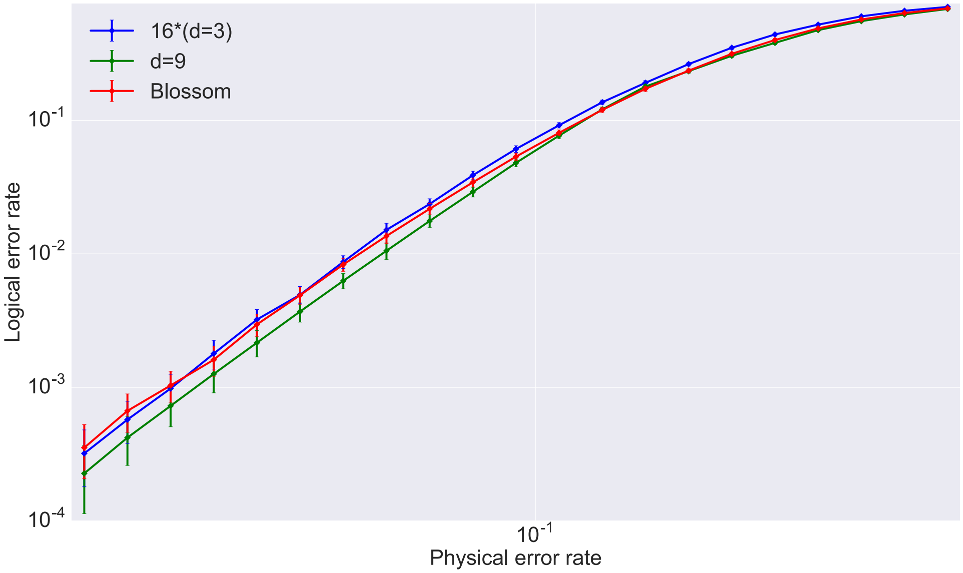


Figure 6.8: Comparison of decoding performance between the distributed decoder with ***sixteen*** overlapping tiles of d=3 rotated surface codes inside a d=9 rotated surface code (blue), the unoptimized version of the Blossom algorithm (red) and the neural network based decoder (green).

As can be seen from Figures 6.6, 6.7 and 6.8, the distributed decoder can reach similar decoding performance to the compared decoders for d=5, 7 and 9, respectively. In order to have a fair comparison between the two neural network based decoders, we used the same dataset to train both decoders, therefore the decoding performance should be comparable. These comparisons were used as a proof-of-concept to verify that a distributed decoding approach is feasible and what limitations are observed.

6.4.1. OPTIMIZING FOR THE SIZE OF TRAINING DATASET

The scalability problem that all neural network based decoders face is based on the exponential increase of the training samples required to efficiently decode. As an extension to our work on neural network based decoders, we propose an alteration to our decoding algorithm in order to increase the important training samples included in the training dataset, without increasing the size of the dataset.

As mentioned, our decoding strategy is based on a two module (simple decoder and neural network) approach, where the neural network exists to increase the decoding performance of the simple decoder. However, the simple decoder can be designed in different ways, which will lead to different decoding performance for different designs. Therefore, an investigation of the performance of the simple decoder is crucial before the training of the neural network.

We observed that for all code distances investigated for the depolarizing error model, the simple decoder provided corrections that would lead to an error free logical state (I) ~42% of the time. In those cases, the neural network would be unnecessary, since it would output the identity operator. Therefore, if we removed the error syndromes that the simple decoder corrects properly from the training dataset, then the dataset could be increased even further, with more relevant error syndromes. The only caveat is that another module, named binary neural network in Figure 6.9, should be included to the decoder which will predict whether the obtained error syndrome will be properly corrected by the simple decoder or not. The binary logic neural network might be implemented in a simpler way, which will make the binary classification task faster, instead of using a recurrent neural network as was chosen for this design.

A flowchart of the optimized algorithm with the inclusion of the extra neural network is presented in Figure 6.9. We divide the operation of the neural network from the original design of distributed decoding, to two neural networks, namely a binary neural network and a neural network for distributed decoding.

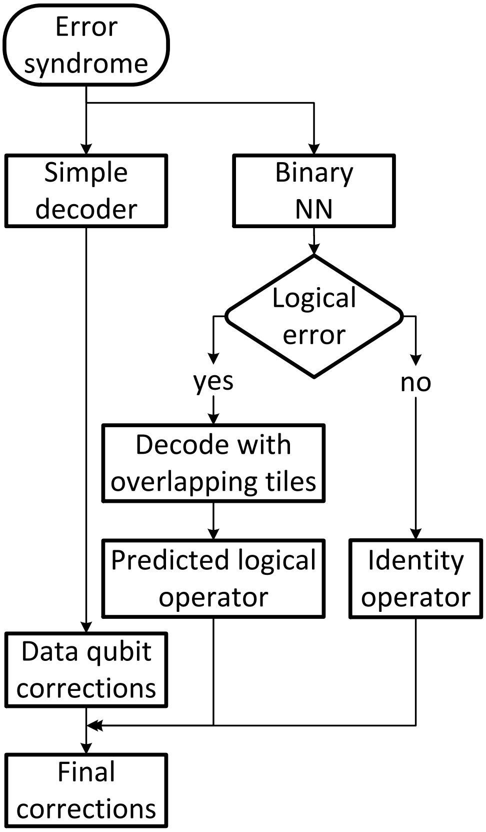

Error syndrome
Simple decoder
Binary NN
Logical error
yes
no
Decode with overlapping tiles
Predicted logical operator
Identity operator
Data qubit corrections
Final corrections

Figure 6.9: Description of the design flow of the optimized version of the distributed decoder.

The binary neural network will predict whether the obtained error syndrome will lead to a logical error or not. The input of the binary neural network is the obtained error syndrome for the whole lattice and the output will be a binary value indicating whether extra corrections need to be applied or not. These extra corrections will arise from the neural network for distributed decoding. This neural network will work similarly to the one in the original unoptimized strategy described in section 6.3, but the training samples will be restricted to the error syndromes that lead to a logical error. The inputs and outputs of this neural network are previously explained. Note that, we need to include all 4 logical states for this neural network, because there is still a probability of an unknown to training input to produce an error free logical state.

The comparison of the decoding performance of this optimized version of the algorithm with the unoptimized one and the benchmarks that were used in this work for the largest code tested (d=9) is presented in Figure 6.10.

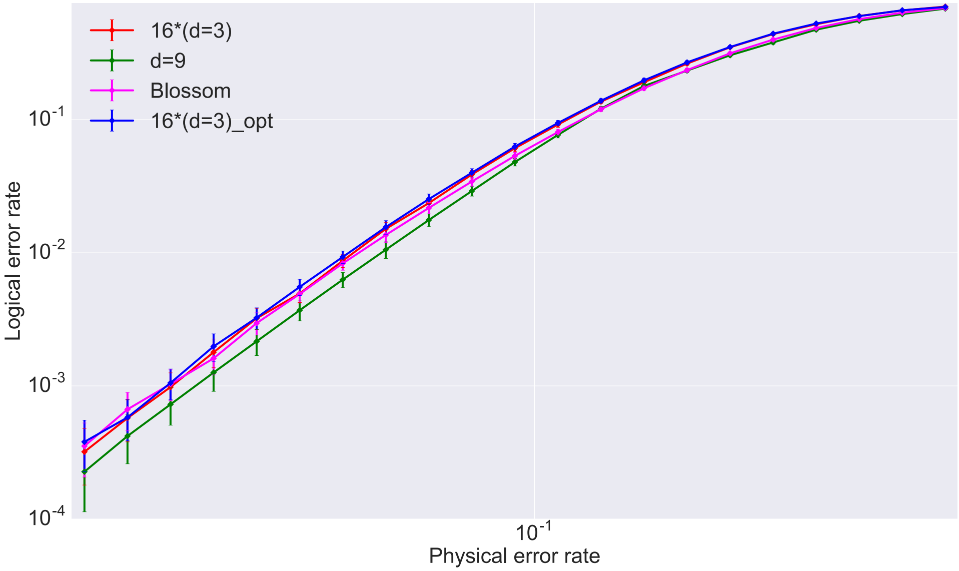


Figure 6.10: Comparison between the optimized version of the distributed decoding (blue) to the unoptimized version (red), the unoptimized version of the Blossom algorithm (pink) and the neural network based decoder (green).

As expected, the optimized version with the two neural networks cannot achieve better decoding performance than the unoptimized version, since we kept the same training dataset for both designs in order to have a fair comparison. The binary neural network has the same dataset as the unoptimized version, but the neural network for distributed decoding only includes the ~58% of error syndromes that lead to a logical error.

An important clarification is that the optimization is mentioned in the context of the potential increase of the training dataset and not in terms of better decoding performance. However, the fact that we reached the same level of decoding performance with both designs, suggests that we can make these optimizations without any loss of decoding performance.

6.5. CONCLUSIONS

We presented a decoding algorithm that performs decoding in a distributed manner that can achieve similar decoding performance to existing decoders, like the Blossom decoder and the neural network based decoder for d=5,7 and 9. Furthermore, due to the distributed way of decoding and the deduction in the neural network inputs, larger codes can be potentially decoded. The problem of the exponential increase of the training dataset is mitigated through the distributed decoding strategy, where any error syndrome can be decomposed to smaller d=3 tiles. However, large quantum systems will still require large amounts of training samples. Moreover, in terms of execution time, we assume that a highly parallel implementation for both the simple decoder and the neural network, can potentially achieve a high speed implementation of the algorithm. Finally, we provide an alternative version of the distributed decoding strategy that can reach the same level of decoding performance as the original algorithm. The advantage of this alternative is the capability of using larger training datasets compared to other neural network based decoders, making it easier to achieve better decoding performance for higher code distances.

The contents of this chapter are based on the following paper:

S. Varsamopoulos, K. Bertels, C. G. Almudever, **Decoding surface code with a distributed neural network based decoder**, arXiv:1901.10847, 2019.

7

CONCLUSIONS AND FUTURE OUTLOOK

In this chapter the overall conclusions for this thesis are summarized and future research directions are suggested.

7.1. CONCLUSIONS

Due to the fragile nature of current qubit technology, constant and active quantum error correction is required to achieve reliable quantum computation and storage. The goal of this research was focused on the development of high decoding performance and high speed decoders that can be used for the surface code, as well as other similar quantum error correcting codes. Emphasis was put on both requirements, since most of the classical decoders only focus on one of them. Furthermore, since the quantum systems that are going to be used in the near-future only consist of a small number of qubits with limited lifetime, dedicated decoders that can reach high decoding performance while obeying the limited time budget for quantum error correction is key for demonstrating fault tolerance. Currently, in the case of the superconducting qubits, the surface code cycle is reported to take ~700 nsec for a quantum chip containing 17 qubits. Therefore, the target for the execution time of the decoder is at most 700 nsec in this scenario.

We began our investigation with a dedicated rule-based decoder for the surface code with 17 qubits, the smallest logical qubit composed with the rotated surface code, that was incorporated in a platform called Quantum Platform Development Framework (QPDO). The goal of this platform was to study the working principles of a Pauli frame and to quantify its potential effect on the decoding performance. QPDO offers the capability of varying the execution time of the decoder and the execution time of the quantum operations. Also, in QPDO error syndrome measurements and decoding can be performed in parallel, thus creating a more efficient execution schedule for the surface code that reduces the cycle time t_{cycle}. The corrections proposed by the decoder do not need to be applied, since we can keep track of both the Pauli errors and Pauli corrections in a Pauli frame in the classical computer where the decoder exists. The proposed execution schedule relaxes the timing constraints on the error syndrome measurements and the decoder. As shown from our simulations, the highest decoding performance is reached when the time required for error correction equals the decoding time $t_{ec} = t_d$, based on the reduced t_{cycle}. However, this kind of decoder is restricted to the number of qubits that can be successfully decoded due to its rule-based approach. It should be noted that this rule-based approach resembles the decoding of the Blossom decoder for the 17 qubit surface code.

In order to go to higher code distances while keeping the execution time of the decoder less than the time budget provided, we decided to create decoders that incorporate neural networks. Neural networks have been shown to have constant execution time after being trained and to be able to adapt in complex problems. We designed a two-module decoder, which included a classical module and a neural network. We named this configuration neural network based decoder. We proved that such a decoder can achieve similar or even better decoding performance compared to classical decoders for a variety of error models. colorredWe showed that the neural network based decoder can outperform Blossom up to ~40% for a variety of error models and code distances. Also, to show evidence of generalization by the neural network, we presented in our graphs the decoding performance of a partial Look-Up Table that included the training samples used by the neural network. We observe that the neural network based decoder significantly outperforms the PLUT as the code distance increases reaching even 99% at one case. Furthermore, we calculated the execution time of a neural network based decoder and argued about the speed that can be achieved in a hardware chip like an FPGA or an ASIC. Both the classical module and the neural network are highly parallelizable and fast modules by construction. colorredWe naively implemented the neural network based decoder in hardware and shown that we can reach the same order of magnitude in terms of execution time (~3.6 musec) based on the required time budget for various quantum technologies at the moment. colorredMoreover, we presented in our simulations the constant execution time of a neural network based decoder for a given distance. The execution time was increasing linearly as the code distance increased, which was mainly due to the increase of the size of the neural network.

We proved that neural network based decoders can adapt to any noise model, since the neural network functionality is based on creating a map between the input and output data, requiring no knowledge about the underlying error model. This is extremely useful at the moment, since certain sources of noise like leakage are not fully modelled in current error models. The only requirement in decoding any encoding scheme with a neural network based decoder is to know how the error syndrome is obtained (what is the error syndrome measurement process) making it a viable solution for any encoding scheme and error model. A comparison was then performed between the two main neural network based decoder design approaches. One approach involves a classical decoder working in parallel with a neural network, which acts as a supervisor to the classical decoder. The other approach involves only neural networks to perform the decoding. We argued that it is advantageous to start with a classical decoding module and improve on its decoding performance with a neural network rather than having a decoding algorithm that allows the neural network to perform the decoding on its own. In that way there is no ambiguity provided by the probabilistic prediction of the neural network and no complex rules need to be introduced in the design of the algorithm, since the prediction of the neural network concerns the proposed corrections of the classical module. In the case of a decoder consisting only of a neural network, the probabilistic prediction occurs for all physical qubits, which will lead to multiple prediction rounds until a valid prediction is obtained, making the execution time non-constant and on average larger than the decoder containing a classical module and a neural network. The non-constant time will be affected by the design of the repetition step of the prediction process and the physical error rate. In our simulations the prediction step had to be repeated between 6 and 16 times for d=3 surface code under depolarizing error model, however we note that the repetition step was not optimized to minimize the repetitions.

Although both type of neural network based decoders discussed can reach equivalent decoding performance to optimal decoders like the Maximum Likelihood Decoder as has been demonstrated in literature, we decided to choose the decoder with the classical module working together with the neural network due to the constant execution time achieved for a given code distance. With this decoder we presented improvement against Blossom that reached up to ~40% for different error models.

Neural network based decoders require sampling and training based on data obtained from the problem, unlike classical decoders. As the quantum system increases, the amount of data required to be gathered and trained are exponentially increasing, imposing a limit to the size of the quantum system that can be efficiently decoded. As presented in Chapter 5, the size of the training dataset is increasing exponentially with the linear increase of the code distance. Therefore, a limit is imposed to the size of the training dataset that can lead us to at least equivalent decoding performance as Blossom or other classical decoders. We argue that the limit for the depolarizing error model with noiseless error syndromes is d=9 with a state space of $3.1 * 10^{144}$ and for the circuit noise model with noisy error syndrome measurements is d=7 with a state space of $2 * 10^{480}$. To counteract this problem, we proposed a distributed decoding approach that divides the code into small regions and then decoding occurs locally in each region. We propose that by using such a distributed decoding approach, we can potentially decode large code distances without significant loss in decoding performance. We have shown that when the training dataset is the same, the original version of the neural network based decoder and the version that performs distributed decoding can reach almost identical performance. We argue that distributed decoding is the way that the scalability challenge of neural network based decoders can be solved.

The main challenge of neural network based decoders is that every time that some aspect of the problem changes (quantum error correction code, code distance, error model), sampling, training and evaluating the decoder needs to be repeated. Moreover, there is a large amount of neural network parameters that need to be specifically tuned when the problem changes. A careful study of the design choices is required to maximize the performance of the decoder. However, if we perform sampling and training in hardware, the time required for these processes will be decreased compared to the time required in software. Finally, if the hardware resources allow us to include multiple neural networks, then this can potentially increase the decoding performance. As we presented, dividing the task of decoding to smaller tasks that are distributed to many neural networks can be beneficial.

7.2. FUTURE RESEARCH DIRECTION

We have developed a design of a neural network based decoder that reaches good decoding performance and has constant execution time. There are many different designs of neural network based decoders being developed at the moment by many research groups, which provide new insight on the challenges of such decoders.

As mentioned, an implementation in hardware should be attempted since it will i) showcase bottlenecks such as the data movement, the implementation of the linear function, etc, ii) quantify the execution time in real hardware and iii) showcase limitations on the size of the neural network(s) that can be implemented. Furthermore, we have been testing our decoder with theoretical error models, therefore a more exhaustive testing with more realistic error models is required. The next step would be to obtain data directly from the quantum chip instead of relying to a simulation based on realistic error models, which will determine the decoding performance in the real hardware. Also, the scalability of neural network based decoders is still an important challenge, although a hardware implementation might mitigate that issue due to the fast sampling and training rate.

Quantum error correction requires many qubits in order to increase protection, but at the moment only chips with an extremely small number of qubits are available. Thus, for such kind of quantum systems, error correction might be detrimental. For that reason, new schemes that require little or no error correction are proposed for Noisy Intermediate-Scale Quantum (NISQ) systems. In the case that error correction is included, neural network based decoders seem to be well suited for such systems. In a NISQ system, the relevant information for evaluating the decoder will be the encoding of the quantum system, the definition of the error model and definition of the error syndrome measurement process. Then, the decoder has all the information required to perform the decoding process.

As quantum technology advances and error rates become smaller, the limitations that current decoders are faced with nowadays will be mitigated. Therefore, we need to start thinking about decoding with error rates many orders of magnitude smaller than current ones ($\sim 10^{-3}$) and envision how decoders will be designed for that scenario.

In the future, we will have quantum chips with many qubits, allowing us to run quantum algorithms with error correction. In the quantum algorithms, we will be using logical qubits and perform logical operations. Decoding such operations, for example logical CNOT via lattice surgery, might have to change the way decoding is assumed for the quantum memory, which is the case of our research so far.

Finally, hardware implemented decoders of any type need to be pursued, in order to get a realistic estimation of the execution time of the decoder. There is a lot of speculation about the execution time of each decoder and the available time budget for decoding provided by various quantum technologies, however a calculation in hardware has not yet been performed.

REFERENCES

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, *Topological quantum memory,* Journal of Mathematical Physics **43**, 4452 (2002), http://dx.doi.org/10.1063/1.1499754 .

[2] A. G. Fowler, A. M. Stephens, and P. Groszkowski, *High-threshold universal quantum computation on the surface code,* Physical Review A **80**, 052312 (2009).

[3] Y. Tomita and K. M. Svore, *Low-distance surface codes under realistic quantum noise,* Phys. Rev. A **90**, 062320 (2014).

[4] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, *An experimental microarchitecture for a superconducting quantum processor,* in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)* (IEEE/ACM, 2017) pp. 813–825.

[5] R. P. Feynman, *Simulating physics with computers,* International Journal of Theoretical Physics **21**, 467 (1982).

[6] P. Benioff, *Quantum mechanical models of turing machines that dissipate no energy,* Phys. Rev. Lett. **48**, 1581 (1982).

[7] D. Deutsch and R. Penrose, *Quantum theory, the church–turing principle and the universal quantum computer,* Proceedings of the Royal Society of London. A. Mathematical and Physical and Engineering Sciences (1985), http://doi.org/10.1098/rspa.1985.0070.

[8] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,* SIAM Journal on Computing **26**, 1484 (1997), http://dx.doi.org/10.1137/S0097539795293172 .

[9] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, *Efficient networks for quantum factoring,* Phys. Rev. A **54**, 1034 (1996).

[10] L. K. Grover, *Quantum mechanics helps in searching for a needle in a haystack,* Phys. Rev. Lett. **79**, 325 (1997).

[11] P. W. Shor, *Scheme for reducing decoherence in quantum computer memory,* Physical review A **52**, R2493 (1995).

[12] A. Steane, *Multiple-particle interference and quantum error correction,* Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452** (1996), 10.1098/rspa.1996.0136.

[13] A. M. Steane, *Error correcting codes in quantum theory,* Phys. Rev. Lett. **77**, 793 (1996).

[14] A. M. Steane, *Simple quantum error-correcting codes,* Phys. Rev. A **54**, 4741 (1996).

[15] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, *Demonstration of a fundamental quantum logic gate,* Phys. Rev. Lett. **75**, 4714 (1995).

[16] D. P. DiVincenzo, *The physical implementation of quantum computation,* arXiv:quant-ph/0002077 (2000).

[17] J. A. Jones and M. Mosca, *Implementation of a quantum algorithm to solve deutsch's problem on a nuclear magnetic resonance quantum computer,* J. Chem. Phys. **109** (1998), 10.1063/1.476739.

[18] I. L. Chuang, N. Gershenfeld, and M. Kubinec, *Experimental implementation of fast quantum searching,* Phys. Rev. Lett. **80**, 3408 (1998).

[19] N. A. Gershenfeld and I. L. Chuang, *Bulk spin-resonance quantum computation,* Science **275**, 350 (1997).

[20] D. G. Cory, A. F. Fahmy, and T. F. Havel, *Ensemble quantum computing by NMR spectroscopy,* Proceedings of the National Academy of Sciences **94**, 1634 (1997).

[21] D. Gottesman, *The heisenberg representation of quantum computers,* arXiv preprint quant-ph/9807006 (1998).

[22] I. Kassal, J. D. Whitfield, A. Perdomo-Ortiz, M.-H. Yung, and A. Aspuru-Guzik, *Simulating chemistry using quantum computers,* Annual Review of Physical Chemistry **62**, 185 (2011).

[23] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, *A variational eigenvalue solver on a photonic quantum processor,* Nature Communications **5**, 4213 (2014).

[24] L. K. Grover, *A fast quantum mechanical algorithm for database search,* in *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)* (ACM, 1996) pp. 212–219.

[25] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, *Quantum machine learning,* Nature **549**, 195 (2017).

[26] K. Lu, Y. Zhang, K. Xu, Y. Gao, and R. C. Wilson, *Approximate maximum common sub-graph isomorphism based on discrete-time quantum walk,* in *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR)* (IEEE, 2014) pp. 1413–1418.

[27] K. K. H. Cheung and M. Mosca, *Decomposing finite abelian groups,* Quantum Information & Computation **1**, 26 (2001).

[28] C. Monroe, D. Meekhof, B. King, W. M. Itano, and D. J. Wineland, *Demonstration of a fundamental quantum logic gate,* Physical Review Letters **75**, 4714 (1995).

[29] S. Debnath, N. Linke, C. Figgatt, K. Landsman, K. Wright, and C. Monroe, *Demonstration of a small programmable quantum computer with atomic qubits,* Nature **536**, 63 (2016).

[30] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Y. 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 J. M. Martinis, *State preservation by repetitive error detection in a superconducting quantum circuit,* Nature **519**, 66 (2015).

[31] D. Ristè, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O.-P. Saira, and L. DiCarlo, *Detecting bit-flip errors in a logical qubit using stabilizer measurements,* Nature Communications **6**, 6983 (2015).

[32] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, J. M. Chow, and J. M. Gambetta, *Hardware-efficient quantum optimizer for small molecules and quantum magnets,* arXiv:1704.05018 (2017).

[33] R. Hanson, L. P. Kouwenhoven, J. R. Petta, S. Tarucha, and L. M. K. Vandersypen, *Spins in few-electron quantum dots,* Reviews of Modern Physics **79**, 1217 (2007).

[34] F. A. Zwanenburg, A. S. Dzurak, A. Morello, M. Y. Simmons, L. C. Hollenberg, G. Klimeck, S. Rogge, S. N. Coppersmith, and M. A. Eriksson, *Silicon quantum electronics,* Reviews of Modern Physics **85**, 961 (2013).

[35] T. F. Watson, S. G. J. Philips, E. Kawakami, D. R. Ward, P. Scarlino, M. Veldhorst, D. E. Savage, M. G. Lagally, M. Friesen, S. N. Coppersmith, M. A. Eriksson, and L. M. K. Vandersypen, *A programmable two-qubit quantum processor in silicon,* Nature **555**, 633 (2018).

[36] G. De Lange, Z. Wang, D. Riste, V. Dobrovitski, and R. Hanson, *Universal dynamical decoupling of a single solid-state spin from a spin bath,* Science **330**, 60 (2010).

[37] J. Cramer, N. Kalb, M. A. Rol, B. Hensen, M. S. Blok, M. Markham, D. J. Twitchen, R. Hanson, and T. H. Taminiau, *Repeated quantum error correction on a continuously encoded qubit by real-time feedback,* Nature Communications **7** (2016).

[38] X.-C. Yao, T.-X. Wang, H.-Z. Chen, W.-B. Gao, A. G. Fowler, R. Raussendorf, Z.-B. Chen, N.-L. Liu, C.-Y. Lu, Y.-J. Deng, Y.-A. Chen, and J.-W. Pan, *Experimental demonstration of topological error correction,* Nature **482**, 489 (2012).

[39] V. Mourik, K. Zuo, S. M. Frolov, S. R. Plissard, E. P. A. M. Bakkers, and L. P. Kouwenhoven, *Signatures of majorana fermions in hybrid superconductor-semiconductor nanowire devices,* Science **336**, 1003 (2012), http://science.sciencemag.org/content/336/6084/1003.full.pdf .

[40] A. D. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, *Demonstration of a quantum error detection code using a square lattice of four superconducting qubits,* Nature Communications **6**, 6979 (2015).

[41] R. Li, L. Petit, D. P. Franke, J. P. Dehollain, J. Helsen, M. Steudtner, N. K. Thomas, Z. R. Yoscovits, K. J. Singh, S. Wehner, L. M. K. Vandersypen, J. S. Clarke, and M. Veldhorst, *A crossbar network for silicon quantum dot qubits,* Science Advances **4** (2018), 10.1126/sciadv.aar3960, http://advances.sciencemag.org/content/4/7/eaar3960.full.pdf .

[42] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information* (Cambridge University Press, 2000).

[43] V. Bužek and M. Hillery, *Quantum copying: Beyond the no-cloning theorem,* Phys. Rev. A **54**, 1844 (1996).

[44] S. J. Devitt, W. J. Munro, and K. Nemoto, *Quantum error correction for beginners,* Reports on Progress in Physics **76**, 076001 (2013).

[45] D. A. Lidar and T. A. Brun, *Quantum Error Correction* (Cambridge University Press, 2013).

[46] E. Knill, *Quantum computing with realistically noisy devices,* Nature **434**, 39 (2005).

[47] J. Ghosh, A. G. Fowler, and M. R. Geller, *Surface code with decoherence: An analysis of three superconducting architectures,* Phys. Rev. A **86**, 062318 (2012).

[48] J. J. Wallman and J. Emerson, *Noise tailoring for scalable quantum computation via randomized compiling,* Phys. Rev. A **94**, 052325 (2016).

[49] Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, *High-level synthesis: productivity, performance, and software constraints,* Journal of Electrical and Computer Engineering **2012**, 1 (2012).

[50] D. Bacon, *Lecture notes in quantum computing, cse599d,* (2018), [Online; accessed 11-02-2019].

[51] Wikipedia, *Adiabatic quantum computation — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 11-02-2019].

[52] Wikipedia, *Topological quantum computer — Wikipedia, the free encyclopedia,* (2018), [Online; accessed 11-02-2019].

[53] A. Paetznick, *Resource optimization for fault-tolerant quantum computing,* arXiv:1410.5124 (2014).

[54] A. R. Calderbank and P. W. Shor, *Good quantum error-correcting codes exist,* Phys. Rev. A **54**, 1098 (1996).

[55] J.-P. Tillich and G. Zémor, *Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength,* IEEE Transactions on Information Theory **60**, 1193 (2014).

[56] D. Bacon, *Operator quantum error-correcting subsystems for self-correcting quantum memories,* Phys. Rev. A **73**, 012340 (2006).

[57] H. Bombin and M. A. Martin-Delgado, *Topological quantum distillation,* Physical review letters **97**, 180501 (2006).

[58] H. Bombin and M. A. Martin-Delgado, *Topological computation without braiding,* Phys. Rev. Lett. **98**, 160502 (2007).

[59] A. Yu. Kitaev, *Quantum error correction with imperfect gates,* Quantum Communication, Computing, and Measurement (1997).

[60] A. Y. Kitaev, *Fault-tolerant quantum computation by anyons,* Annals of Physics **303**, 2 (2003).

[61] S. Bravyi, G. Duclos-Cianci, D. Poulin, and M. Suchara, *Subsystem surface codes with three-qubit check operators,* Quantum Info. Comput. **13**, 963 (2013).

[62] C. Jones, P. Brooks, and J. Harrington, *Gauge color codes in two dimensions,* Phys. Rev. A **93**, 052332 (2016).

[63] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, *Surface codes: Towards practical large-scale quantum computation,* Physical Review A **86**, 032324 (2012).

[64] R. Raussendorf and J. Harrington, *Fault-tolerant quantum computation with high threshold in two dimensions,* Phys. Rev. Lett. **98**, 190504 (2007).

[65] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg, *Surface code quantum computing with error rates over 1%,* Phys. Rev. A **83**, 020302 (2011).

[66] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, *Towards practical classical processing for the surface code,* Phys. Rev. Lett. **108**, 180501 (2012).

[67] H. Bombin and M. A. Martin-Delgado, *Quantum measurements and gates by code deformation,* Journal of Physics A: Mathematical and Theoretical **42**, 095302 (2009).

[68] H. Bombin, *Clifford gates by code deformation,* New Journal of Physics **13**, 043005 (2011).

[69] S. B. Bravyi and A. Y. Kitaev, *Quantum codes on a lattice with boundary,* arXiv preprint quant-ph/9811052 (1998).

[70] M. H. Freedman and D. A. Meyer, *Projective plane and planar quantum codes,* Foundations of Computational Mathematics **1**, 325 (2001).

[71] 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 (2017).

[72] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, *Surface code quantum computing by lattice surgery,* New Journal of Physics **14**, 123011 (2012).

[73] B. M. Terhal, *Quantum error correction for quantum memories,* Reviews of Modern Physics **87**, 307 (2015).

[74] R. Raussendorf, J. Harrington, and K. Goyal, *Topological fault-tolerance in cluster state quantum computation,* New Journal of Physics **9**, 199 (2007).

[75] S. Bravyi, M. Suchara, and A. Vargo, *Efficient algorithms for maximum likelihood decoding in the surface code,* Phys. Rev. A **90**, 032326 (2014).

[76] J. Edmonds, *Paths, trees, and flowers,* Canadian Journal of Mathematics **17**, 449 (1965).

[77] V. Kolmogorov, *Blossom V: a new implementation of a minimum cost perfect matching algorithm,* Mathematical Programming Computation **1**, 43 (2009).

[78] A. G. Fowler, *Minimum weight perfect matching of fault-tolerant topological quantum error correction in average* $o(1)$ *parallel time,* Quantum Information & Computation **15**, 145 (2015).

[79] A. Hutter, J. R. Wootton, and D. Loss, *Efficient markov chain monte carlo algorithm for the surface code,* Phys. Rev. A **89**, 022326 (2014).

[80] J. R. Wootton and D. Loss, *High threshold error correction for the surface code,* Phys. Rev. Lett. **109**, 160503 (2012).

[81] G. Duclos-Cianci and D. Poulin, *A renormalization group decoding algorithm for topological quantum codes,* Information Theory Workshop (ITW), IEEE , 1 (2010).

[82] G. Duclos-Cianci and D. Poulin, *Fast decoders for topological quantum codes,* Phys. Rev. Lett. **104**, 050504 (2010).

[83] A. G. Fowler, A. C. Whiteside, A. L. McInnes, and A. Rabbani, *Topological code autotune,* PHYSICAL REVIEW X **2**, 041003 (2012).

[84] A. G. Fowler, *Optimal complexity correction of correlated errors in the surface code,* arXiv:1310.0863 (2013).

[85] G. Kuczera and E. Parent, *Monte carlo assessment of parameter uncertainty in conceptual catchment models: the metropolis algorithm,* Journal of Hydrology **211**, 69 (1998).

[86] G. Torlai and R. G. Melko, *Neural decoder for topological codes,* Phys. Rev. Lett. **119**, 030501 (2017).

[87] S. Krastanov and L. Jiang, *Deep neural network probabilistic decoder for stabilizer codes,* Scientific Reports **7** (2017), 10.1038/s41598-017-11266-1.

[88] S. Varsamopoulos, B. Criger, and K. Bertels, *Decoding small surface codes with feedforward neural networks,* Quantum Science and Technology **3**, 015004 (2018).

[89] P. Baireuther, T. E. O'Brien, B. Tarasinski, and C. W. J. Beenakker, *Machine-learning-assisted correction of correlated qubit errors in a topological code,* Quantum **2**, 48 (2018).

[90] C. Chamberland and P. Ronagh, *Deep neural decoders for near term fault-tolerant experiments,* Quantum Science and Technology **3**, 044002 (2018).

[91] X. Ni, *Neural network decoders for large-distance 2d toric codes,* arXiv:1809.06640 (2018).

[92] S. Varsamopoulos, K. Bertels, and C. G. Almudever, *Designing neural network based decoders for surface codes,* arXiv:1811.12456 (2018).

[93] A. Davaasuren, Y. Suzuki, K. Fujii, and M. Koashi, *General framework for constructing fast and near-optimal machine-learning-based,* arXiv:1801.04377 (2018).

[94] M. Maskara, A. Kubica, and T. Jochym-O'Connor, *Advantages of versatile neural-network decoding for topological codes,* arXiv:1802.08680 (2018).

[95] A. S. Darmawan and D. Poulin, *Linear-time general decoding algorithm for the surface code,* Phys. Rev. E **97**, 051302 (2018).

[96] R. Sweke, M. S. Kesselring, E. P. L. van Nieuwenburg, and J. Eisert, *Reinforcement learning decoders for fault-tolerant quantum computation,* arXiv:1810.07207 (2018).

[97] D. A. Lidar, *Review of decoherence-free subspaces, noiseless subsystems, and dynamical decoupling,* arXiv:1208.5791 (2014).

[98] G. H. Low, T. J. Yoder, and I. L. Chuang, *Optimal arbitrarily accurate composite pulse sequences,* Phys. Rev. A **89**, 022341 (2014).

[99] Y. Li and S. C. Benjamin, *Efficient variational quantum simulator incorporating active error minimization,* Phys. Rev. X **7**, 021050 (2017).

[100] K. Temme, S. Bravyi, and J. M. Gambetta, *Error mitigation for short-depth quantum circuits,* Phys. Rev. Lett. **119**, 180509 (2017).

[101] I. Changhau, *Lstm and gru – formula summary,* (2017).

[102] X. Fu, L. Riesebos, L. Lao, C. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, *A heterogeneous quantum computer architecture,* in *CF* (ACM, 2016) pp. 323–330.

[103] M. Takita, A. Córcoles, E. Magesan, B. Abdo, M. Brink, A. Cross, J. M. Chow, and J. M. Gambetta, *Demonstration of weight-four parity measurements in the surface code architecture,* arXiv preprint arXiv:1605.01351 (2016).

[104] E. Knill, *Scalable quantum computing in the presence of large detected-error rates,* Physical Review A **71**, 042322 (2005).

[105] D. P. DiVincenzo and P. Aliferis, *Effective fault-tolerant quantum computation with slow measurements,* Physical review letters **98**, 020501 (2007).

[106] P. Aliferis and J. Preskill, *Fault-tolerant quantum computation against biased noise,* Physical Review A **78**, 052331 (2008).

[107] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, *Layered architecture for quantum computing,* Physical Review X **2**, 031007 (2012).

[108] S. Balensiefer, L. Kregor-Stickles, and M. Oskin, *An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures,* in *ACM SIGARCH Computer Architecture News*, Vol. 33 (IEEE Computer Society, 2005) pp. 186–196.

[109] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, *A layered software architecture for quantum computing design tools,* IEEE Computer **39**, 74 (2006).

[110] D. Wecker and K. M. Svore, *Liqui|>: A software design architecture and domain-specific language for quantum computing,* arXiv preprint arXiv:1402.4467 (2014).

[111] K. Svore, A. Cross, A. Aho, I. Chuang, and I. Markov, *Toward a software architecture for quantum computing design tools,* in *QPL* (2004) pp. 145–162.

[112] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, *Scaffcc: A framework for compilation and analysis of quantum computing programs,* in *CF* (ACM, 2014) p. 1.

[113] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels, *QX: A high-performance quantum computer simulation platform,* in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)* (IEEE, 2017) pp. 464–469.

[114] S. Aaronson and D. Gottesman, *Improved simulation of stabilizer circuits,* Physical Review A **70**, 052328 (2004).

[115] I. Georgescu, S. Ashhab, and F. Nori, *Quantum simulation,* Reviews of Modern Physics **86**, 153 (2014).

[116] T. Monz, D. Nigg, E. A. Martinez, M. F. Brandl, P. Schindler, R. Rines, S. X. Wang, I. L. Chuang, and R. Blatt, *Realization of a scalable Shor algorithm,* Science **351**, 1068 (2016).

[117] S. Jordan, *Quantum algorithms,* (2016).

[118] Z. Wu, J. Li, W. Zheng, J. Luo, M. Feng, and X. Peng, *Experimental demonstration of the Deutsch-Jozsa algorithm in homonuclear multispin systems,* Phys. Rev. A **84**, 042312 (2011).

[119] Y. Liu and F. Zhang, *First experimental demonstration of an exact quantum search algorithm in nuclear magnetic resonance system,* Science China Physics, Mechanics & Astronomy **58**, 1 (2015).

[120] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, *A variational eigenvalue solver on a photonic quantum processor,* Nature Communications **5** (2014), 10.1038/ncomms5213, http://dx.doi.org/10.1038/ncomms5213 .

[121] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer, A. Blais, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, *Demonstration of two-qubit algorithms with a superconducting quantum processor,* Nature **460**, 240 (2009), http://dx.doi.org/10.1038/nature08121 .

[122] C. Chamberland, P. Iyer, and D. Poulin, *Fault-Tolerant Quantum Computing in the Pauli or Clifford Frame with Slow Error Diagnostics,* ArXiv e-prints (2017), arXiv:1704.06662 [quant-ph] .

[123] T. O'Brien, B. Tarasinski, and L. DiCarlo, *Density-matrix simulation of small surface codes under current and projected experimental noise,* arXiv preprint arXiv:1703.04136 (2017).

[124] E. Nachmani, Y. Be'ery, and D. Burshtein, *Learning to decode linear codes using deep learning,* in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (IEEE, 2016) pp. 341–346.

[125] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, *RNN Decoding of Linear Block Codes,* arXiv preprint arXiv:1702.07560 (2017).

[126] D. Poulin, *Optimal and efficient decoding of concatenated quantum block codes,* Phys. Rev. A **74**, 052333 (2006).

[127] D. J. MacKay, *Information theory, inference and learning algorithms* (Cambridge university press, 2003).

[128] S. Kullback and R. A. Leibler, *On information and sufficiency,* Ann. Math. Statist. **22**, 79 (1951).

[129] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems,* (2015), software available from tensorflow.org.

[130] *Virtex UltraScale FPGAs Datasheet : DC and AC switching characteristics* (2017).

[131] A. J. Landahl, J. T. Anderson, and P. R. Rice, *Fault-tolerant quantum computing with color codes,* arXiv:1108.5738 (2011).

[132] H. Bombin, G. Duclos-Cianci, and D. Poulin, *Universal topological phase of two-dimensional stabilizer codes,* New Journal of Physics **14**, 073048 (2012).

[133] A. Couvreur, N. Delfosse, and G. Zémor, *A construction of quantum LDPC codes from Cayley graphs,* IEEE Transactions on Information Theory **59**, 6087 (2013).

[134] D. Bacon, S. T. Flammia, A. W. Harrow, and J. Shi, *Sparse quantum codes from quantum circuits,* IEEE Transactions on Information Theory **63**, 2464 (2017).

[135] D. Gottesman, *Stabilizer Codes and Quantum Error Correction* (Caltech Ph.D. Thesis, 1997).

[136] M. Herold, E. T. Campbell, J. Eisert, and M. J. Kastoryano, *Cellular-automaton decoders for topological quantum memories,* Npj Quantum Information **1** (2015), 10.1038/npjqi.2015.10.

SUMMARY

Quantum error correction (QEC) is key to have reliable quantum computation and storage, due to the fragility of qubits in current quantum technology and the imperfect application of quantum operations. In order to have efficient quantum computation and storage, active QEC is required. QEC consists of an encoding and a decoding process. The way that encoding protects quantum information is through grouping many unreliable physical qubits into one more reliable logical qubit. Then, computation occurs based on the logical qubits, however, errors still occur on the physical qubits. Decoding is the process of identifying the location and type of errors occurring on the physical qubits. The decoder proposes corrections against the errors that have been identified. In this thesis, we are exploring novel ways to design decoders for QEC codes, focusing on the surface code.

We began our investigation by implementing a rule-based decoder for the smallest surface code, which consists of 17 qubits. We incorporated this decoder to a platform that we created, called Quantum Platform Development Framework (QPDO), in order to study the working principles of a Pauli frame and to quantify its potential effect on the decoding performance. The Pauli frame unit keeps track of errors on physical qubits without the need to apply corrections constantly. We quantified through simulation the benefits in terms of the decoding performance and the execution schedule of QEC, minimizing the idle time. Minimizing the execution time is critical, due to the limited time budget of quantum error correction, thus requiring a high speed decoder capable of still reaching high decoding performance. We show that when the decoding time is equal to the time required to run a surface code cycle, the decoder reaches its maximum performance. However, such a rule-based decoder cannot easily scale to larger quantum systems, therefore other decoding approaches should be considered.

Most of the classical decoders that have been developed so far, do not have a good balance between short execution time and high decoding performance. Therefore, we proposed decoders that incorporate neural networks to keep the execution time small, while keeping the decoding performance high. We designed a two-module decoder, which included a classical module and a neural network. We named this configuration neural network based decoder (NNbD). We compare different designs of NNbDs with classical decoders and prove that NNbDs can reach similar or better decoding performance compared to classical decoders while having constant execution time. Furthermore, we quantified the execution time of a NNbD and argued about the speed that can be achieved in a hardware chip like a Field Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC). Both the classical module and the neural network are highly parallelizable and fast modules by construction, leading to constant execution time for a given code distance. We proved that neural network based decoders can adapt to any noise model, since the neural network functionality is based on creating a map between the input and output data, requiring no knowledge about the underlying error model.

Following that, a comparison between different NNbD design approaches was performed. We show that it is advantageous to start with a classical decoding module and improve on its decoding performance with a neural network rather than having a neural network perform the decoding on its own. Also, in the latter case, the execution time of

such a decoder is non-constant and on average larger than the decoder containing a classical module and a neural network. Moreover, we show that for the design containing a classical module and a neural network, the execution time is increasing linearly as the code distance increased, which was mainly attributed to the increase of the size of the neural network.

However, there is a fundamental difference between NNbDs and classical decoders in that NNbDs require sampling and training based on data obtained from the problem, unlike classical decoders. As the code distance increases, the amount of data required to be gathered and trained are exponentially increasing, imposing a limit to the size of the quantum system that can be efficiently decoded. We proposed as a solution to have a distributed decoding approach that divides the code into small regions and then decodes each region locally. We show that using such a distributed decoding approach for small code distances does not lead to significant loss in decoding performance, while simultaneously providing a way to decode large code distances.

Thus, we were able to create a decoder that can achieve high decoding performance with constant execution time. However, there are still some issues to keep in mind with such kind of decoders. The main challenge of NNbDs is that they are a dedicated decoder for a given problem. Every time that some aspect of the problem changes (quantum error correcting code, code distance, error model), sampling, training and evaluating the decoder needs to be repeated. Moreover, there is a large number of neural network parameters that need to be specifically tuned when the problem changes. A careful study of the design choices is required to maximize the performance of the decoder.

We envision that when sampling and training are performed in hardware, the time required for these processes will be decreased compared to the time required in software. Finally, if the hardware resources allow us to include multiple neural networks, then this can potentially increase the decoding performance. As we presented, dividing the task of decoding to smaller tasks that are distributed to many neural networks can be beneficial.

SAMENVATTING

Quantumfoutcorrectie (QEC) is essentieel voor betrouwbare quantumberekeningen en -opslag vanwege de fragiliteit van qubits en de imperfecte toepassing van quantumoperaties in de huidige quantumtechnologie. Voor efficiënte quantumberekeningen en -opslag is actieve QEC vereist. QEC bestaat uit een coderings- en een decoderingsproces. Codering beschermt quantuminformatie door veel onbetrouwbare, fysieke qubits te groeperen tot een meer betrouwbare, logische qubit. De berekening vindt vervolgens plaats op basis van de logische qubits maar er treden nog steeds fouten op in de fysieke qubits. Decoderen is het identificatieproces van de plek en soort van de optredende fouten in de fysieke qubits. De decoder stelt correcties voor op de gevonden fouten. In dit proefschrift onderzoeken we nieuwe manieren om decoders te ontwerpen voor QEC-codes, in het bijzonder voor surface-code.

We zijn ons onderzoek begonnen met de implementatie van een op regels gebaseerde decoder voor de kleinste surface-code die uit 17 qubits bestaat. We hebben deze decoder geïntegreerd in een platform, Quantum Platform Development Framework (QPDO) genaamd, dat we hebben gecreëerd om de werkingsprincipes van een Pauli-frame te bestuderen en om het mogelijke effect ervan op de decodeerkwaliteit te kwantificeren. Een Pauli-frame houdt bij welk soort fouten in fysieke qubits optraden zonder dat voortdurend correcties nodig zijn. We hebben door middel van simulatie de voordelen voor de decodeerkwaliteit van QEC en hoe vaak QEC moet draaien gekwantificeerd, en daarmee de leeglooptijd geminimaliseerd. Het minimaliseren van de uitvoeringstijd is van cruciaal belang vanwege het beperkte tijdsbudget voor quantumfoutcorrectie waardoor een snelle decoder nodig is die nog steeds tot hoge decoderingsprestaties in staat is. We laten zien dat wanneer de decoderingstijd gelijk is aan de tijd die nodig is om een surface-code cyclus uit te voeren, de decoder het meest optimaal werkt. Een dergelijke op regels gebaseerde decoder kan echter niet eenvoudig worden opgeschaald naar grotere quantumsystemen en daarom moeten andere manieren van decoderen worden overwogen.

De meeste klassieke decoders die tot nu toe zijn ontwikkeld, hebben geen goede balans tussen korte uitvoeringstijd en hoge decodeerkwaliteit. Daarom hebben we decoders voorgesteld die neurale netwerken bevatten om de uitvoeringstijd klein te houden, terwijl de decodeerkwaliteit hoog blijven. We hebben een tweecomponentdecoder ontworpen met een klassieke deel en een neuraal netwerk. We noemden deze combinatie neurale-netwerkgebaseerde decoders (NNbD). We vergelijken verschillende NNbD-ontwerpen met klassieke decoders en bewijzen dat NNbD's vergelijkbaar of beter kunnen presteren met decoderen in vergelijking tot klassieke decoders met constant uitvoeringstijd. Verder hebben we de uitvoeringstijd van een NNbD gekwantificeerd en beargumenteerd over de snelheid die kan worden bereikt in een hardware-chip zoals een Field Programmable Gate Array (FPGA) of een applicatie-specifieke geïntegreerde schakeling (ASIC). Zowel de klassieke module als het neurale netwerk zijn zeer parallelliseerbaar en snelle modules door constructie, wat leidt tot een constante uitvoeringstijd voor een gegeven codeafstand. We hebben bewezen dat neurale-netwerkgebaseerde decoders kunnen worden aangepast aan elk ruismodel, omdat de neurale-netwerkfunctionaliteit is gebaseerd op het maken van een afbeelding tussen de invoer- en uitvoergegevens, waardoor er geen kennis nodig is van het onderliggende foutmodel.

Daarna werd een vergelijking tussen verschillende NNbD ontwerpbenaderingen uitgevoerd. We laten zien dat het voordelig is om te beginnen met een klassieke decoderingsmodule en de decodeerkwaliteit met een neuraal netwerk te verbeteren in plaats van dat een neuraal netwerk het decoderen alleen uitvoert. Ook is in het laatste geval de uitvoeringstijd van een dergelijke decoder niet constant en gemiddeld groter dan die van een decoder die een klassieke module en een neuraal netwerk bevat. Bovendien laten we zien dat in het ontwerp met een klassieke module en een neuraal netwerk, de uitvoeringstijd lineair toeneemt naarmate de codeafstand toeneemt, wat voornamelijk kan worden toegeschreven aan de toename van de grootte van het neurale netwerk.

Er is echter een fundamenteel verschil tussen NNbD's en klassieke decoders omdat NNbD's vergaring van informatie en training vereisen op basis van gegevens die uit het probleem zijn verkregen, in tegenstelling tot klassieke decoders. Naarmate de codeafstand toeneemt, neemt de hoeveelheid gegevens die moet worden verzameld en waarmee moet worden getraind exponentieel toe, waardoor een limiet wordt gesteld aan de omvang van het quantumsysteem dat efficiënt kan worden gedecodeerd. We hebben een oplossing voorgesteld voor een gedistribueerde decodeermethode die de code verdeelt in kleine regio's en vervolgens elke regio lokaal decodeert. We laten zien dat het gebruik van een dergelijke gedistribueerde decodeermethode voor kleine codeafstanden niet leidt tot aanzienlijk verlies van de decodeerkwaliteit, terwijl tegelijkertijd een manier wordt geboden om grote codeafstanden te decoderen.

We waren dus in staat om een decoder te creëren die hoge decodeerkwaliteit met een constante uitvoeringstijd kan halen. Er zijn echter nog steeds enkele problemen waarmee men rekening moet houden bij dergelijke decoders. De grootste uitdaging van NNbD's is dat ze specifiek decoderen voor een gegeven probleem. Telkens wanneer een bepaald aspect van het probleem verandert (quantumfoutcorrectiecodering, codeafstand, foutmodel), moet het vergaren van informatie, het trainen en het evalueren van de decoder herhaald worden. Bovendien is er een grote hoeveelheid neurale-netwerkparameters die specifiek moeten worden ingeregeld wanneer het probleem verandert. Een zorgvuldige afweging van de ontwerpkeuzes is vereist om de prestaties van de decoder te maximaliseren.

We voorzien dat wanneer vergaring van informatie en training worden uitgevoerd in hardware, de tijd die nodig is voor deze processen wordt verminderd in vergelijking met de tijd die nodig is in de software. Tenslotte, als de hoeveelheid ons ter beschikking staande hardware ons in staat stelt meerdere neurale netwerken op te nemen, kan dit de decodeerkwaliteit mogelijk verhogen. Zoals we hebben uiteengezet, kan het van voordeel zijn om de decoderingstaak op te splitsen en te verdelen over meerdere neurale netwerken.

CURRICULUM VITÆ

Savvas VARSAMOPOULOS

Savvas Varsamopoulos was born on October 27th, 1988 in Thessaloniki, Greece. In 2006, he started his undergraduate studies at Aristotle University of Thessaloniki, Greece. As an undergraduate student in the Physics department, he spent the first three years studying various topics in the field of Physics. The final year of his undergraduate studies, he took specialization courses in Electronics and Telecommunications. He obtained his Bachelor degree in Physics in 2011.

During the same year (2011), he enrolled in the Master program of the same university to continue his studies in Electronics and Telecommunications. The Master program consisted of courses about microelectronics, digital system design and programming. His Master thesis involved the implementation of the Simplex algorithm into a Field Programmable Gate Array (FPGA). He received his Master of Science degree in 2014.

A couple of months after finishing the Master of Science program in 2014, he accepted a position as a Ph.D. Candidate in the Quantum and Computer Engineering lab of Delft University of Technology, to work on quantum error correction under the guidance of Prof. dr. Koen Bertels. He was part of the initial group that began investigating quantum computing in the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS/EWI) of Delft University of Technology.

His Ph.D. studies focus on developing high decoding performance and high speed decoders for the surface code and other quantum error correcting codes. The results of his work are presented in the current dissertation.

LIST OF PUBLICATIONS

Conference Proceedings

1. L. Riesebos, X. Fu, **S. Varsamopoulos**, C.G. Almudever, K. Bertels, *Pauli frames for quantum computer architectures*, Proceedings of the 54th Annual Design Automation Conference (DAC), ACM, 76, (2017).
2. C.G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, **S. Varsamopoulos**, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, K. Bertels, *The engineering challenges in quantum computing*, Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp.836-846, (2017).
3. L. Riesebos, X. Fu, A. A. Moueddenne, L. Lao, **S. Varsamopoulos**, I. Ashraf, J. van Someren, N. Khammassi, C.G. Almudever, K. Bertels, *Quantum Accelerated Computer Architectures*, International Symposium on Circuits and Systems, IEEE, (2019) (Accepted).

Journal Papers

1. **S. Varsamopoulos**, B. Criger, K. Bertels, *Decoding small surface codes with feedforward neural networks*, Quantum Science and Technology, vol. 3, IOP Publishing, (2017).

arXiv

1. **S. Varsamopoulos**, K. Bertels, C.G. Almudever, *Designing neural network based decoders for surface codes*, arXiv:1811.12456, 2018. (Submitted to IEEE Transactions on Computers)
2. **S. Varsamopoulos**, K. Bertels, C.G. Almudever, *Decoding surface code with a distributed neural network based decoder*, arXiv:1901.10847, 2019. (Submitted to New Journal of Physics)