Efficient Methods for Tackling Error in Discrete Quantum Circuits

A thesis submitted for the degree of

Doctor of Philosophy

by

Debasmita Bhoumik

Advisor

Prof. Susmita Sur-Kolay

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203 B. T. Road, Kolkata - 700108

July, 2024

To Shovik and Smitika,
whose unwavering support has strengthened, enriched,
and fulfilled me throughout this long journey.

Acknowledgement

I would like to express my sincere gratitude to all those who have supported me throughout the course of my PhD journey.

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Susmita Sur-Kolay. Her unwavering patience and guidance have been instrumental in navigating the challenges of my PhD journey. Her support, especially during periods of difficulty, has been invaluable in making this experience smoother. I could not have hoped for a better Ph.D supervisor.

My heartfelt thanks go to Ritajit Majumdar, a friend who has become a true mentor, philosopher, and guide. From the beginning of our PhD journey, working on various projects together, his timely support, insightful guidance, and even his constructive criticisms have greatly contributed to the success of this work. I feel incredibly fortunate to have such a supportive friend.

I also want to acknowledge both Ritajit and Kritanta, my labmates, who have been with me since our preparation for the ISI entrance exams. Our shared experiences, including the countless enjoyable moments spent exploring new restaurants and the inevitable weight gain from food deliveries to the lab, have been a cherished part of this journey.

My sincere thanks to the faculty and staff of ACMU and the Dean's Office for their consistent support and assistance whenever I needed it.

I am grateful to Professor Gautam Mahapatra (GM Sir), my first Computer Science teacher at college, for sparking my interest in research and guiding me towards this career path when I was initially considering an IT job. Your influence has been pivotal in shaping my academic journey.

I extend my appreciation to my co-authors, especially Amit Saha (Amit Da), for his enthusiasm and timely contributions. I also thank Dhiraj Madan, Shesha Raghunathan, and Dhinakaran Vinayagamurthy for their regular meetings and support, which were crucial whenever I encountered difficulties. Additionally, I am thankful to Dr. Anupama Ray for her mentorship during my research internship at IBM Quantum India; it was truly a rewarding experience.

I would like to acknowledge all the professors from Bethune College and the University of Calcutta for laying the groundwork for my studies in Computer Science.

To my husband, Shovik, my deepest gratitude for your unwavering support throughout this journey. Our shared path has been as long as my PhD journey, and your support has been extraordinary. Your role as a 'more than equal' parent has been crucial, allowing me to focus on my research while knowing that our daughter is in such capable hands. I also thank my daughter, Smitika, for her patience and cooperation. The sacrifices I made, including sleepless nights away from you, were made easier by your peaceful sleep and understanding.

I am profoundly grateful to my parents for their belief in me, their encouragement towards a better career, and their support throughout my academic journey.

My appreciation extends to my parents-in-law, who have selflessly taken on the role of primary caregiver for my daughter, allowing me to work with focus and dedication. Without their support, this achievement would not have been possible.

Special thanks to my best friend, Priyanka Banerjee. Your patience, understanding, and steadfast support during challenging times have been invaluable. Your non-judgmental nature and constant presence have been a source of strength for me.

Lastly, I would like to thank all those whose names may have inadvertently been omitted. Your support and encouragement have been crucial to the completion of this thesis.

Signature: Debasmita Bhoumik

DEBASMITA BHOUMIK
14.01.2025

Abstract

Quantum computing has emerged as a groundbreaking field with the potential to solve certain complex problems that are currently intractable for classical computers. Leveraging the principles of quantum mechanics, quantum computers offer exponential speedup for specific tasks, making them a revolutionary tool for various domains, including cryptography, material science, and optimization. The unique capabilities of quantum computers, such as superposition and entanglement, enable entirely new computational paradigms with far-reaching implications. However, despite their immense potential, the practical realization of quantum computing faces significant challenges, such as high error rates and limited qubit coherence.

One of the primary obstacles in quantum computing is managing errors that arise from decoherence and imperfect quantum gate operations. These errors severely limit the performance and scalability of quantum circuits. This thesis is dedicated to developing efficient methods for addressing issues related to errors in discrete quantum circuits. By optimizing quantum circuit design and implementing robust error correction strategies, this research aims to significantly enhance the performance of quantum computations. This work is relevant for both the Noisy Intermediate-Scale Quantum (NISQ) era, characterized by quantum devices with a moderate number of qubits prone to noise and errors, and the future era of fault-tolerant quantum computers where error correction can be integral.

The contributions in this thesis comprising of two parts, are novel techniques for circuit optimization and error correction that are tailored to the unique challenges of both NISQ devices and more advanced error-corrected quantum computers of the future. The research encompasses both theoretical advancements and practical implementations, providing a comprehensive framework for improving the fidelity and efficiency of quantum computations. Specifically, the thesis explores innovative methods for circuit design optimizations, strategies to enhance qubit coherence and gate fidelity and error correction.

In the NISQ era, the focus is on developing strategies to reduce noise and errors

to make practical use of the currently available quantum devices. This involves exploring methods as noise-aware circuit design that can operate effectively despite the presence of noise. For the fault-tolerant era, error syndrome detection which is indispensable for error correction, is a computationally hard problem. In this thesis, machine learning based approaches to address this problem have been proposed, implemented and validated.

Overall, this thesis provides a comprehensive exploration of efficient methods for tackling errors in discrete quantum circuits, offering valuable contributions to enhance the performance and reliability of quantum computations. Through a combination of theoretical insights and practical implementations, this research advances the state of the art in quantum error correction and circuit optimization, setting the stage for the future of quantum computing.

Table of Contents

List of Figures viii

List of Tables viii

1 Introduction 2

1.1 Introduction 3

1.2 Basics of Quantum Computing 4

1.2.1 A Quantum Computer and Its Characteristics 5

1.3 Challenges of Quantum Computing 7

1.3.1 Decoherence and Dephasing 8

1.3.2 Gate Errors 8

1.3.3 Measurement Errors 8

1.3.4 Qubit Connectivity and Crosstalk 9

1.3.5 Error Accumulation and Quantum Error Correction 9

1.3.6 Hardware Limitations and Scalability 9

1.4 Motivation and Scope of this Thesis 10

1.4.1 NISQ Era 11

1.4.2 Error Correction Era 11

1.5 Contributions and Organization of the thesis 12

2 Background **16**

2.1 Noise in Quantum Computing 17

2.1.1 Errors in a Quantum Computing System 17

2.1.2 Asymmetric unitary noise channel via twirling 18

2.2 Circuit cutting 21

2.3 Circuit placement and selection of good qubits 23

2.3.1 Intra Device Scheduling 26

2.4 Quantum Error Correction 27

2.4.1 Stabilizer Formulation of Surface Code 30

2.4.2 Heavy Hexagon Code 36

I Error suppression for NISQ devices: Current Era **45**

3 Time and noise optimization for distributed scheduling of quantum circuits **47**

3.1 Introduction 48

3.2 Hardware Schedule for Subcircuits 49

3.3 Proposed framework 51

3.3.1 Selection of appropriate hardware 52

3.3.2 Scoring each hardware as per noise profile 53

3.3.3 Noise and Time Aware Distributed Scheduler (NoTaDS) . . 53

3.4 Experimental Results 57

3.4.1 Criteria for maximum execution time τ 58

3.4.2 Estimation of the execution time of a circuit 59

3.4.3 Results for 6-qubit circuits 60

3.4.4 Results for 10-qubit circuits 62

3.4.5 Variation in fidelity with the number and size of subcircuits 63

3.4.6 Results for a 28-qubit circuit 65

3.4.7 Change in fidelity with and without scheduling 66

3.5 A polynomial time solution for a restricted scenario 69

3.6 Summary 71

4 Resource-aware scheduling of multiple quantum circuits on a hardware device 72

4.1 Introduction 73

4.2 Formulation of intra-device scheduling as an optimization problem . 75

4.2.1 Finding the overlap between two layouts 76

4.2.2 ILP Formulation for our scheduling problem 79

4.3 Polynomial time heuristic algorithm 83

4.3.1 Generation of the compatibility graph 84

4.3.2 Greedy algorithm to find a *maximal* clique in the compatibility graph 87

4.4 Experimental results 89

4.4.1 Selection of ϵ for the layouts 90

4.4.2 Fidelity and hardware utilization in intra-device scheduling . 91

4.5 Conclusion 92

II Error correction in Fault Tolerant Era 97

5 Machine-Learning based Decoding of Surface Code Syndromes in Quantum Error Correction 99

5.1 Introduction 100

5.2 Design methodology of our ML based decoder 102

5.2.1 Mapping surface code onto a square lattice 102

5.2.2 Error injection and syndrome extraction 103

5.2.3 Training our ML model 104

5.3 Experimental Results 106

5.3.1 Noise models 106

5.3.2 Machine Learning Parameters 108

5.3.3 More sophisticated ML models 110

5.3.4 Empirical train-test-ratio for optimal accuracy 112

5.3.5 Performance on Training with Symmetric Noise Models and Testing with Asymmetric Noise Models 118

5.4 Conclusion 119

6 Efficient Syndrome Decoder for Heavy hexagon QECC via Machine Learning 120

6.1 Introduction 121

6.2 Designing ML based decoder for heavy hexagon code 122

6.3 Reducing error classes for heavy hexagon code 123

6.3.1 Reducing bit flip error classes by search based gauge equivalence . . . 126

6.3.2 Reducing phase flip error classes by search based gauge equivalence . . . 131

6.3.3 Reducing bit flip error classes by rank based gauge equivalence . . . 134

6.4 Simulation Results . . . 138

6.4.1 Machine Learning Parameters . . . 139

6.4.2 Estimating Logical Error Rate with our Machine Learning based Syndrome Decoder . . . 140

6.4.3 Comparison of ML-based decoder results with MWPM . . . 141

6.4.4 Scalability of our ML based decoder . . . 148

6.4.5 Machine Learning based Decoding of Heavy hexagon QECC for Asymmetric Quantum Noise . . . 150

6.5 Discussion . . . 153

7 Conclusion and Future directions **155**

7.1 Summary . . . 155

7.2 Chapter-wise Contributions . . . 156

7.3 Future Directions . . . 158

Bibliography **iv**

List of Figures

2.1 Circuit cutting for a 6-qubit RealAmplitudes ansatz, with single repetition and reverse-linear entanglement [KM+17], into two sub-circuits. 23

2.2 The coupling map and error distribution of a 5 qubit IBM Quantum device *Belem*. 24

2.3 An example of *mapomatic* [NK+21] to find the best placement of a circuit on hardware. This figure is obtained from the GitHub repository of mapomatic (https://github.com/Qiskit-Partners/mapomatic). 25

2.4 An example of a 15-qubit circuit assigned to a 27- qubit hardware. The used qubits are shown in purple while the unused qubits are shown in blue. The hardware still has room to accommodate one or more quantum circuit(s) using the free qubits. 26

2.5 Threshold (black dot) and Pseudo-threshold (red dot) of a MWPM decoder for a topological (heavy hexagon Code) QECC, with performance comparison for distance 3 (blue), distance 5 (orange), and distance 7 (green). The cyan straight line $y = x$ is for equal probabilities of physical qubit error and logical error. 29

2.6 (a) Distance 3 surface code, where the numbered circles (0 - 8) are the physical qubits, white plaquettes are X stabilizers (i.e., M-X qubits $AX0$, $AX1$, $AX2$, $AX3$), gray plaquettes are Z stabilizers (i.e., M-Z qubits $AZ0$, $AZ1$, $AZ2$, $AZ3$); (b) the syndromes are defined in a $d \times d$ lattice (d=3), with physical qubits on the vertices and plaquette stabilizers (measure qubits) as faces: (i) pink (purple) plaquettes indicate stabilizers which check the Z (X) parity of qubits on the vertices of the plaquettes as shown in (ii), (iii) green circles indicate errors and red circles violated stabilizers (i.e., syndromes [VCB17]) 31

2.7 Quantum circuit for a single cycle of surface code. (a) circuit for M-Z qubit, (b) circuit for M-X qubit [FWH12] 32

2.8 Example of surface code for d=5, where two errors produce the same syndrome [VCB17]. Pink (Purple) plaquettes indicate stabilizers which check the Z (X) parity of qubits on the vertices of the plaquette. Green circles are used to indicate errors and red circles to indicate violated stabilizers. 33

2.9 (a) No logical error and (b) Logical error due to mis-classification in low level decoding 36

2.10 Distance 3 heavy hexagon code encoding one logical qubit: (a) the hexagonal structure, (b) the circuit illustration of the heavy hexagon code with the CNOT gates. Here yellow, white and black circles represents data, flag and ancilla qubits respectively; black ancilla qubits are for measuring the X (red face or plaquette) and Z (blue face or strip) gauge generators. The product of two Z gauge generators at each white plaquette forms a Z stabilizer. The figure is adapted from the reference [CZ+20]. 38

2.11 Circuits for measuring X and Z gauge generators in the heavy hexagon code where t_i denotes the i^{th} time step. Two flag qubits (white circles) measure a X gauge generator having a weight of 4 and one flag qubit measures a Z gauge generator having a weight of 2 [CZ+20]. 39

2.12 The stabilizers and gauge generators for a distance 5 heavy hexagon code: the weight-4 X gauge generators are in the red plaquettes, while the weight-2 X gauge generators are on the upper and lower boundaries. Each blue strip denotes weight-2 Z gauge generators. A vertical strip of two adjacent columns with X gauge generators form an X stabilizer. The weight-4 Z gauge generators in the white plaquettes, and weight-2 Z gauge generators on the left and right boundaries are themselves Z stabilizers. [CZ+20]. 40

3.1 A flowchart of our proposed noise and time optimized scheduler, including circuit cutting, scoring of (circuit, hardware) pair, noise and time optimized scheduling, and final reconstruction of the entire probability distribution from those of the subcircuits. 52

3.2 Cutting with unequal depth : to understand why area is an important element in the objective function 56

3.3 An example of a 6-qubit RealAmplitudes circuit with the level. . . . 60

3.4 Two subcircuits of 6-qubit Ripple carry adder circuit obtained by using Circuit-knitting-toolbox [LA+23]. 62

3.5 Fidelity obtained by the $NoTaDS$ scheduler with an increasing number of subcircuits for 20-qubit RealAmplitudes and Bernstein Vazirani (BV) circuits. 64

3.6 The increase in classical reconstruction time of the full probability distribution from the subcircuits with increasing number of subcircuits. 65

3.7 Fidelity obtained by the NoTaDS scheduler with increasing size of the circuit where each circuit is partitioned into two subcircuits. . . 66

3.8 Fidelity of a 16-qubit RealAmplitudes circuit, when partitioned into 2 subcircuits, and executed on all possible hardware pair. Since there are only two subcircuits, when the subcircuits are scheduled to two different hardware, the execution time is τ_{min}, and when scheduled to the same hardware the execution time is τ_{max}. 67

3.9 For mapping the 6-qubit ripple carry adder to the available hardware : (a) The bipartite graph where the first (left) set of vertices denotes the hardware, and the second (right) set of vertices denotes the two subcircuits of a 6-qubits ripple carry adder. (b) A Minimum Weight Maximum Matching based solution to the assignment of two subcircuits corresponding to the 6-qubit ripple carry adder. . 69

4.1 An example of a 15-qubit circuit assigned to a 27- qubit hardware. The used qubits are shown in purple while the unused qubits are shown in blue. The hardware still has room to accommodate one or more quantum circuit(s) using the free qubits. 73

4.2 Placement of a 15-qubit and a 8-qubit circuit simultaneously on a 27-qubit hardware with buffer distance (a) $b = 0$, and (b) $b = 2$. The former has a significantly higher probability of crosstalk affecting the quality of the computation. The blue qubits are the unused ones. 76

4.3 The entire workflow of our heuristic algorithm: (a) schematic diagrams of three circuits that are to be placed in the hardware; (b) two possible layouts for each of the three circuits and their corresponding mapomatic scores; (c) the compatibility graph with edge weights, (d) the connected components of the graph and (e) greedy selection of edges in each of the components. 94

4.4 Fidelity (along with the mean and standard deviation) for benchmark circuits (QAOA, Trotterized, Real Amplitude) with and without using our intra-device scheduling executed in (a) Noisy IBMQ simulator with the noise profile and coupling map of 27-qubit IBMQ KOlkata, (b) 127-qubit IBMQ Brisbane hardware. 96

5.1 SC17 to syndrome generation 103

5.2 Outline of the ML based syndrome decoding for surface code 107

5.3 Pseudo-threshold and accuracy — MWPM vs ML-based decoder for distance 3 surface code 109

5.4 Threshold and accuracy — MWPM and ML-based decoder for surface code with $d = 3, 5, 7$ 111

5.5 ML model accuracy vs physical error probability for various ML models in $d = 3$ and 5 surface code 112

5.6 Average accuracy (along with its standard deviation) of our low level decoder vs Test Ratio for different values of p_{phys} in distance 3 surface code 115

5.7 Logical vs physical error probability for various ML models in $d = 3$ and 5 surface code 116

5.8 Training with symmetric noise Models and testing with asymmetric noise models 117

6.1 (a) X gauge equivalence for bit flip error: simultaneous errors on data qubits 4, 7 and 8 is equivalent to an error on data qubit 5, because by applying X gauge operator $G3$ (consisting of X operators on data qubits 4, 5, 7 and 8) on data qubits 4, 7 and 8, data qubit 5 has an error. (b) Z gauge equivalence for phase flip errors: an error on data qubit 7 is equivalent to an error on data qubit 1 because by applying Z gauge operator $g4$ (consisting of Z operators on qubits 4 and 7) followed by Z gauge operator $g1$ (consisting of Z operators on qubits 1 and 4) on data qubit 7, data qubit 1 has an error. This figure is adapted from [CZ+20] 125

6.2 Threshold and pseudo-threshold values for MWPM and ML based decoders in case of distance 3, 5, and 7 heavy hexagon code for bit flip error on data qubit 142

6.3 Threshold and pseudo-threshold values for MWPM and ML based decoders (without and with gauge equivalence) and in case of distance 3, 5, and 7 heavy hexagon code for bit flip error on data qubit along with measurement, and stabilizer error 144

6.4 Comparison of time (in seconds) needed to (a) pre-process and find gauge equivalence based on linear search and on rank for distance 3 (blue), distance 5 (orange) and 7 (green) heavy hexagon code, (b) train the FFNN based decoder for the three cases, namely without gauge equivalence, with search based gauge equivalence and with rank based gauge equivalence 145

6.5 Phase flip error rates for the heavy hexagon code – (a) In MWPM based decoder (b) In FFNN based decoder with gauge equivalence. 147

6.6 In FFNN based decoder for depolarizing noise model (a) Logical X error rates and (b) logical Z error rates for the heavy hexagon code. In MWPM based decoder (c) Logical X error rates and (d) logical Z error rates for the heavy hexagon code. 149

6.7 (a) Using FFNN based decoder, Logical error rates for the heavy hexagon code and (b) Using MWPM based decoder, Logical error rates for the heavy hexagon code, for Amplitude damping noise model, $\gamma_A = 0.01$ 151

6.8 Training in one γ_A value and testing in other γ_A values 153

List of Tables

2.1 *Mapomatic* score for the best layout of the 6-qubit RealAmplitudes circuit (Fig: 2.1) corresponding to each of the available hardware . 25

2.2 The 20 Z gauge generators $Z_{i,j}Z_{i+1,j}$ with weight 2 41

2.3 The 8 X gauge generators $X_{i,j}X_{i,j+1}X_{i+1,j}X_{i+1,j+1}$ with weight 4 . . 41

2.4 The 4 X gauge generators $X_{1,2m-1}X_{1,2m}$ and $X_{d,2m}X_{d,2m+1}$ with weight 2 41

2.5 The 8 Z stabilizers $Z_{i,j}Z_{i,j+1}Z_{i+1,j}Z_{i+1,j+1}$ with weight 4 42

2.6 The 4 Z stabilizers $Z_{2m-1,d}Z_{2m,d}$ and $Z_{2m,1}Z_{2m+1,1}$ with weight 2 . 42

2.7 The 4 X stabilizers $\Pi_i X_{i,j}X_{i,j+1}$ 43

3.1 Number of qubits and noise profile of the hardware 57

3.2 Fidelity for 6-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM). 61

3.3 Scheduling details of the 6-qubit ripple carry adder circuit and its two subcircuits obtained after cutting. 62

3.4 Fidelity for 10-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM). 63

3.5 Fidelity for 28-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM). 66

4.1 Comparison of values of fidelity for the best score, the worst score and the last of the top 50% in noisy simulator of IBMQ Kolkata for 5-qubit circuits 90

4.2 Fidelity for different sized Real Amplitude circuits with and without using our intra-device scheduling to be run on Noisy IBMQ simulator with the noise profile and coupling map of 27-qubit IBMQ Kolkata 91

4.3 Hardware utilization in intra device scheduling 92

5.1 Pseudo-threshold of the low and high level decoders for distance d = 3, 5 and 7 surface code 113

5.2 Comparison of training times for different ML models 113

5.3 Comparison of threshold of the low and high level decoders 114

6.1 Number of bit flip error class labels without and with gauge equivalence 127

6.2 Number of error class labels without and with gauge equivalence in case of phase flip 130

6.3 Number of nodes in the layers of our FFNN based heavy hexagon QECC syndrome decoder for bit flip and phase flip errors 141

6.4 Comparison of UF, MWPM and FFNN based decoder result in case of bit flip error (assuming ideal stabilizers and measurements) . . . 142

6.5 Comparison of FFNN based decoder, result without and with gauge equivalence in case of bit flip error on data qubit along with measurement and stabilizer errors 146

6.6 Comparison of FFNN based decoder with MWPM decoder for logical X in depolarization noise model 148

6.7 Comparison of FFNN based decoder with MWPM decoder for logical error in various noise model 152

CHAPTER 1

Introduction

Contents

1.1 Introduction **3**

1.2 Basics of Quantum Computing **4**

1.2.1 A Quantum Computer and Its Characteristics 5

1.3 Challenges of Quantum Computing **7**

1.3.1 Decoherence and Dephasing 8

1.3.2 Gate Errors 8

1.3.3 Measurement Errors 8

1.3.4 Qubit Connectivity and Crosstalk 9

1.3.5 Error Accumulation and Quantum Error Correction . . 9

1.3.6 Hardware Limitations and Scalability 9

1.4 Motivation and Scope of this Thesis **10**

1.4.1 NISQ Era 11

1.4.2 Error Correction Era 11

1.5 Contributions and Organization of the thesis **12**

1.1 Introduction

> "The ultimate purpose of quantum mechanics is to describe the universe. The universe is, essentially, a gigantic quantum computer."
>
> –Seth Lloyd

This profound insight underscores the intrinsic connection between quantum mechanics and computational paradigms, illuminating the transformative potential of quantum computing in addressing complex, real-world challenges [Llo00].

Quantum computing has emerged as a revolutionary field with the potential to solve complex problems that are intractable for classical computers [NC10]. Leveraging principles of quantum mechanics, quantum computers offer exponential speedup for specific tasks, making them a topic of intense research and development. The unique capabilities of quantum computers, such as superposition and entanglement, enable new computational paradigms that have far-reaching implications across various domains, including cryptography, material science, and optimization problems. However, the practical realization of quantum computing faces significant hurdles, particularly concerning error rates and qubit coherence [Pre18].

One of the primary challenges in quantum computing is dealing with errors that arise from decoherence and imperfect quantum gate operations [Sho96]. These errors severely limit the performance and scalability of quantum circuits. My research focuses on developing efficient methods for tackling these errors in discrete quantum circuits. By optimizing quantum circuit design and implementing robust error correction strategies, this thesis aims to enhance the reliability and perfor-

mance of quantum computations in both the Noisy Intermediate-Scale Quantum (NISQ) era and the future error correction era [Kit97]. The NISQ era is characterized by quantum devices with a moderate number of qubits that are prone to noise and errors, necessitating innovative approaches to error mitigation and circuit optimization.

This thesis contributes to the field by presenting novel techniques for error correction and circuit optimization that are tailored to the unique challenges of both NISQ devices and the more advanced error-corrected quantum computers of the future. The research encompasses both theoretical advancements and practical implementations, providing a comprehensive framework for improving the fidelity and efficiency of quantum computations. The following sections of this introduction will delve into the basics of quantum computing, outline the specific challenges faced in the field, and detail the motivation and scope of this thesis. Additionally, the contributions and organization of the thesis will be presented, offering a roadmap for understanding the structure and significance of the work.

1.2 Basics of Quantum Computing

Quantum computing is based on the principles of quantum mechanics, which govern the behavior of particles at the atomic and subatomic levels. Unlike classical computers that use bits as the smallest unit of information, quantum computers use qubits. A qubit can exist in a state of 0, 1, or any quantum superposition of these states, enabling it to process a vast amount of information simultaneously [NC10]. This section introduces the fundamental concepts and properties of quantum computing, along with the postulates that form the basis of quantum mechanics.

1.2.1 A Quantum Computer and Its Characteristics

Superposition

A qubit can be in a superposition of the states $|0\rangle$ and $|1\rangle$, represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$ [NC10]. This property allows quantum computers to perform many calculations at once, providing an exponential increase in computational power for certain tasks.

Entanglement

When two qubits become entangled, the state of one qubit is dependent on the state of the other, no matter the distance between them [NC10]. This correlation, which has no classical analog, is a cornerstone of quantum computing, enabling phenomena such as quantum teleportation and superdense coding [BB+93].

Quantum Gates

Quantum gates manipulate the state of qubits, analogous to classical logic gates but operating on quantum states. Examples include the Pauli-X, Pauli-Y, Pauli-Z, Hadamard (H), and controlled-NOT (CNOT) gates [NC10]. These gates are represented by unitary matrices and can be combined to form quantum circuits that execute algorithms [Pre18].

Quantum Circuits

A quantum circuit is a sequence of quantum gates applied to an initial set of qubits to perform a specific computation [NC10]. The design and optimization of quantum circuits are critical for the efficiency of quantum algorithms. Circuits must

be designed to minimize error rates and maximize the fidelity of the computation [Sho96].

Measurement

The process of measurement in quantum mechanics collapses a qubit's superposition state into one of the basis states, $|0\rangle$ or $|1\rangle$, with probabilities determined by the coefficients α and β [NC10]. Measurement is a probabilistic process that provides the final result of a quantum computation [Pre18].

Notable Algorithms

Quantum algorithms leverage the properties of superposition and entanglement to achieve significant speedups over classical algorithms [NC10]. Shor's algorithm, for instance, factors large integers exponentially faster than the best-known classical algorithms, posing a threat to classical cryptographic systems [Sho96]. Grover's algorithm provides a quadratic speedup for unstructured search problems, highlighting the potential for quantum advantage in various applications [Gro96].

Postulates of Quantum Mechanics

- **State Postulate**: The state of a quantum system is described by a vector $|\psi\rangle$ in a complex vector space known as the Hilbert space [NC10]. The state vector contains all the information about the system.
- **Evolution Postulate**: The evolution of a closed quantum system is governed by the Schrödinger equation [Sho96]. The state vector $|\psi\rangle$ evolves over time according to a unitary transformation U, such that $|\psi(t)\rangle = U|\psi(0)\rangle$.
- **Measurement Postulate**: Quantum measurements are described by a set of measurement operators $\{M_i\}$. When a measurement is made, the prob-

ability of obtaining the outcome i is given by $\langle\psi|M_i^\dagger M_i|\psi\rangle$, and the state collapses to $\frac{M_i|\psi\rangle}{\sqrt{\langle\psi|M_i^\dagger M_i|\psi\rangle}}$ [NC10].

- **Composite Systems Postulate**: The state space of a composite quantum system is the tensor product of the state spaces of the individual subsystems. For systems A and B, the combined state is described by $|\psi\rangle_A \otimes |\psi\rangle_B$ [NC10].

These postulates form the foundation of quantum mechanics and underpin the operation of quantum computers. Understanding these principles is essential for grasping the capabilities and limitations of quantum computing.

Quantum computing represents a paradigm shift in computational theory and practice. Its principles challenge our classical understanding of computation, opening new frontiers for solving complex problems. As we navigate the current NISQ era, characterized by noisy and intermediate-scale quantum devices, and look forward to the error correction era, the efficient management of quantum errors remains a pivotal research focus. This thesis addresses these challenges by exploring innovative methods for error correction and circuit optimization, aiming to enhance the performance and reliability of quantum computations across different eras of quantum technology [Pre18].

1.3 Challenges of Quantum Computing

Despite the significant potential of quantum computers, several formidable challenges must be addressed to realize their full capabilities. Among these challenges, noise and error rates are particularly critical, as they directly impact the reliability and scalability of quantum computations. This section explores the various sources of noise and errors in quantum systems and their implications for quantum computing.

1.3.1 Decoherence and Dephasing

Decoherence occurs when a qubit loses its quantum coherence through interaction with its environment, causing it to transition from a pure state to a mixed state [Zur03]. This interaction results in a loss of information stored in the quantum state, severely limiting the time available for quantum computations. Dephasing, a specific type of decoherence, affects the phase relationship between quantum states, further degrading the accuracy of quantum operations. Both decoherence and dephasing are influenced by factors such as temperature, electromagnetic interference, and material impurities [Pre18].

1.3.2 Gate Errors

Quantum gates, which are the building blocks of quantum circuits, are prone to errors due to imperfections in their implementation [NC10]. These errors can arise from imprecise control pulses, crosstalk between qubits, and limitations in the hardware. Gate errors accumulate over the course of a computation, leading to a significant decrease in the overall fidelity of the quantum circuit [Sho96]. Error rates of quantum gates are a major concern, especially for complex algorithms that require a large number of gate operations.

1.3.3 Measurement Errors

The process of measuring a qubit's state introduces another source of error. Measurement errors occur due to imperfections in the measurement apparatus and noise in the readout process [NC10]. These errors can result in incorrect outcomes, further complicating the task of obtaining reliable results from quantum computations. Accurate and efficient measurement techniques are essential for extracting meaningful information from quantum systems.

1.3.4 Qubit Connectivity and Crosstalk

The physical layout and connectivity of qubits on a quantum chip influence the efficiency and accuracy of quantum computations [Pre18]. Limited qubit connectivity can necessitate additional gate operations, such as SWAP gates, to move qubits into the required positions, thereby increasing the overall error rate. Additionally, crosstalk between neighboring qubits can introduce unwanted interactions, leading to further errors in the computation. Optimizing qubit layout and minimizing crosstalk are crucial for improving quantum circuit performance [Zur03].

1.3.5 Error Accumulation and Quantum Error Correction

As quantum computations scale up, the accumulation of errors poses a significant challenge. Quantum error correction (QEC) aims to address this issue by encoding logical qubits into multiple physical qubits, allowing for the detection and correction of errors [Sho96]. However, implementing QEC requires additional qubits and resources, which are currently limited in the NISQ era. Developing efficient error correction codes and fault-tolerant quantum computing techniques is essential for the transition to large-scale, reliable quantum computers [Pre18].

1.3.6 Hardware Limitations and Scalability

Current quantum hardware is limited in terms of the number of qubits, coherence times, and gate fidelities [NC10]. Scaling up quantum computers to handle more complex computations requires overcoming these hardware limitations. Advances in qubit technology, materials science, and fabrication techniques are needed to build larger, more robust quantum systems. Additionally, developing new architectures and error mitigation strategies will be key to achieving scalable quantum computing [Sho96].

Addressing these challenges is critical for advancing the field of quantum computing. My thesis focuses on developing efficient methods for tackling errors in discrete quantum circuits, with an emphasis on both the current NISQ era and the future error correction era. By improving error correction techniques and optimizing quantum circuit design, this research aims to enhance the reliability and performance of quantum computers, paving the way for practical and scalable quantum applications.

1.4 Motivation and Scope of this Thesis

Quantum computing represents a transformative leap in computational capabilities, with the potential to address problems that are currently intractable for classical systems [Pre18, JATK+24]. However, the realization of practical quantum computing is hindered by the challenges of noise and inefficiency inherent in current quantum hardware. These challenges are particularly pronounced in the Noisy Intermediate-Scale Quantum (NISQ) era, where devices are limited by both their inherent noise and the constraints of execution time and resources. To advance the field, it is essential to develop innovative solutions that enhance the fidelity of quantum circuits and optimize the scheduling of quantum operations [NC10].

Addressing these challenges requires a multifaceted approach that combines advancements in error suppression with efficient resource management. The motivation for this thesis lies in the necessity to improve quantum circuit performance and operational efficiency, ultimately contributing to the progression towards fault-tolerant quantum computing [Sho96].

The main objective of this thesis is to explore and develop techniques to enhance the quality of quantum computations in the presence of noise. In the short term, this involves creating methods to mitigate the impact of noise on quantum systems. In the long term, the goal shifts towards the implementation of error correction

and fault tolerance mechanisms. This thesis is structured to address both these facets comprehensively in two distinct parts.

1.4.1 NISQ Era

In the NISQ (Noisy Intermediate-Scale Quantum) era, the focus is on optimizing quantum computations and scheduling in the presence of noise. This section tackles the optimization of quantum circuit scheduling in the NISQ era, where noise and execution time constraints are critical factors. It explores methods for both time and noise optimization in distributed scheduling of quantum circuits and resource-aware scheduling of multiple quantum circuits on a single hardware device [Pre18]. The goal is to enhance the fidelity of quantum operations and improve the overall efficiency of quantum processors. By integrating innovative scheduling techniques with noise-aware strategies, this part addresses the immediate challenges faced by current quantum hardware, offering practical solutions to boost performance.

1.4.2 Error Correction Era

In the era of quantum error correction, the focus shifts to developing robust decoding algorithms to correct errors and maintain quantum coherence. The second part of the thesis focuses on advancing quantum error correction (QEC) methods through machine learning (ML) techniques. It introduces novel ML-based decoders designed to correct errors in surface codes and heavy hexagonal codes [Sho96]. These decoders leverage advanced techniques such as gauge equivalence to improve decoding performance for both symmetric and asymmetric noise models. The exploration of ML models in this context reveals significant improvements over traditional methods, providing a pathway towards more effective error correction in quantum systems. This part also considers future research directions, including the application of these techniques to other noise models, further ad-

vancing the field of quantum error correction.

In summary, this thesis aims to address key challenges in quantum computing by optimizing NISQ device performance and advancing QEC techniques, thereby contributing to the broader goal of developing practical and fault-tolerant quantum computing systems [Pre18].

1.5 Contributions and Organization of the thesis

The chapter-wise contributions of the thesis are listed below.

- Chapter 1 covers the basic principles and challenges of quantum computing, and the scope of the thesis.
- Chapter 2 provides a background of this thesis.
- **Part I**: Error suppression for NISQ devices: Current Era
  - **Time and Noise Optimization for Distributed Scheduling of Quantum Circuits:** In Chapter 3, we introduce a distributed scheduler optimized for noise and execution time, designed to manage subcircuits generated through circuit cutting. This scheduler ensures that fidelity is maximized while adhering to a predefined execution time limit for each hardware unit. By integrating inter-device parallelization with noise-aware scheduling, our approach enhances circuit fidelity. For 10-qubit circuits, our method achieves an average fidelity improvement over uncut circuit by $\sim 12.3\%$ and $\sim 21\%$ respectively with and without measurement error mitigation, even when each hardware was allowed the minimum possible execution time. Furthermore, we introduce a polynomial-time graph-theoretic scheduling method that yields the same results as the ILP scheduler when the number of subcircuits is not greater than the number of available hardware, and each hardware is allocated the minimum execution time allowed. Additionally,

this chapter explores the potential of this method to be particularly advantageous in the near-term, where noisy devices and limited execution time on quantum hardware are prevalent.

- **Resource-aware Scheduling of Multiple Quantum Circuits on a Hardware Device:** Chapter 4 tackles the crucial issue of optimizing quantum circuit scheduling to boost the throughput and efficiency of quantum computing hardware. By drawing parallels to the classical bin packing problem, we establish the NP-Hard nature of our problem, which involves allocating multiple quantum circuits to quantum processing units while accounting for noise and limited qubit connectivity. Our proposed solution leverages integer linear programming (ILP) and a greedy heuristic approach based on compatibility graphs and maximal cliques, effectively managing the trade-off between noise reduction and resource utilization. Our method provides 2× and 3× better utilization for 27-qubit and 127-qubit hardware devices respectively in terms of qubits and time. The chapter presents experimental results demonstrating substantial gains in throughput and efficiency, validating the approach as a practical solution for real-world quantum computing challenges.

- **Part II** Error correction in Fault Tolerant Era

  - **Machine-Learning based Decoding of Surface Code Syndromes in Quantum Error Correction:** In Chapter 5, we introduced a machine learning (ML) decoder designed to correct both symmetric and asymmetric depolarizing noise on surface codes. Our decoder operates in two stages: a low-level stage that aims to predict qubit errors accurately, followed by a high-level stage that identifies any logical errors introduced by the low-level decoder. This approach utilizes neural networks (both feedforward neural networks and convolutional neural networks) for surface codes with distances of 3, 5, and 7. The results show that our proposed decoding method achieves 10 and 2 higher values of pseudo-threshold and threshold respectively, than for those with MWPM. Moreover, ML based decoders can do faster decoding than

MWPM (detailed calculation in section 6.4.4). Furthermore, the decoder performs effectively with asymmetric errors, which are more representative of real-world quantum device conditions. The chapter also investigates the performance of various ML models with differing levels of complexity, revealing that while more sophisticated models require longer decoding times, they do not necessarily offer improved performance.

– **Efficient Syndrome Decoder for Heavy Hexagonal QECC via Machine Learning for Symmetric and Asymmetric Noise Models:**

Chapter 6 introduces a machine learning (ML) decoder for the heavy hexagonal code, employing an innovative technique based on gauge equivalence to enhance decoding performance. We demonstrated that even a basic ML decoder achieves ~ 5 times higher values of threshold than that by MWPM. The application of gauge equivalence further boosts the decoder's performance, indicating its potential to advance quantum devices towards fault tolerance. For symmetric noise, we explored search-based and rank-based methods to determine gauge equivalence, finding that the search-based method excels with phase flip errors while the rank-based method performs better with bit flip errors. We obtain a quadratic reduction in the number of error classes for both bit flip and phase flip errors, thus achieving a further improvement of $\sim 14\%$ in the threshold over the basic ML decoder. In the case of asymmetric noise, the rank-based gauge equivalence method significantly enhances the ML decoder's performance compared to MWPM decoders. The chapter also outlines future research directions, including the exploration of gauge equivalence and ML-based decoders for other noise models, such as Pauli, amplitude damping, and amplitude-phase damping noise.

- Chapter 7 summarizes the contributions of this thesis and discusses potential directions for future research.

In the following chapter, we delve into some necessary background concepts that is necessary to read this thesis.

CHAPTER 2

Background

Contents

2.1 Noise in Quantum Computing 17

2.1.1 Errors in a Quantum Computing System 17

2.1.2 Asymmetric unitary noise channel via twirling 18

2.2 Circuit cutting 21

2.3 Circuit placement and selection of good qubits 23

2.3.1 Intra Device Scheduling 26

2.4 Quantum Error Correction 27

2.4.1 Stabilizer Formulation of Surface Code 30

2.4.2 Heavy Hexagon Code 36

2.1 Noise in Quantum Computing

In the realm of quantum computing, achieving reliable computations without error correction is an ongoing challenge. Circuit cutting has emerged as a promising technique to mitigate the impact of noise by leveraging classical post-processing, albeit at the expense of additional computational resources. Despite these advancements, the ultimate goal remains error correction and fault tolerance, essential for enabling arbitrarily long quantum computations.

This chapter begins by providing a comprehensive overview of various types of noise that commonly affect quantum systems. Following this, the concept of circuit cutting is introduced, highlighting its role in reducing noise. The chapter then delves into general principles of intra-device scheduling, discussing strategies to optimize the execution of quantum circuits. Finally, an examination of error correction techniques is presented, with a focus on stabilizer quantum error-correcting codes (QECC), surface codes, and heavy-hex codes.

2.1.1 Errors in a Quantum Computing System

The physical errors in the qubits can be of different types. In this subsection we discuss briefly the noise models that are used for this work.

Bit flip error

The action of a bit flip error [NC10] on a quantum state ρ is denoted as

$$\rho \rightarrow (1 - p_x)\rho + p_x X \rho X^\dagger,$$

p_x being the probability that an unwanted Pauli X error occurs.

Phase flip error

The evolution of the state in a phase flip [NC10] is given as

$$\rho \rightarrow (1 - p_z)\rho + p_z Z\rho Z^\dagger$$

where p_z is the probability that an unwanted Pauli Z error occurs.

Depolarization noise

The evolution of a quantum state ρ under depolarization noise [NC10] is given as

$$\rho \rightarrow (1 - p)\rho + \frac{p}{3}X\rho X^\dagger + \frac{p}{3}Y\rho Y^\dagger + \frac{p}{3}Z\rho Z^\dagger$$

where p is the probability of error. It is basically a depolarizing channel with error rate p [LM+19].

2.1.2 Asymmetric unitary noise channel via twirling

The evolution of a quantum system ρ in the presence of noise is governed by the Kraus operator represented as [Pre98]

$$\rho \rightarrow \sum_i K_i \rho K_i^\dagger$$

where K_i are the Kraus operators and $\sum_i K_i^\dagger K_i = I$. For a unitary noise model such as depolarization noise [NC10], we have $K_i^\dagger K_i = K_i K_i^\dagger = I$ for all i. The evolution of an n-qubit quantum state under this model is given as

$$\rho \rightarrow (1 - p)\rho + p\frac{I}{2^n}$$

where p is the probability of error. This noise model keeps the state intact with probability $1-p$, and completely destroys the information in it with probability p. Mathematically, this is equivalent to a system being affected by the Pauli X (bit flip), Z (phase flip), and Y (bit-phase flip) errors with equal probability [NC10], i.e., $p_x = p_z = p_y$. However, for non-unitary noise models, *twirling* [WE16] is used to create an average channel which behaves like unitary noise. By twirling, each 2-qubit gate U is replaced by $G_i U G_j^\dagger$ such that these are functionally equivalent. Here G_i is sampled randomly from a group such as Clifford group or Pauli group. $G_j^\dagger$ depends on the choice of G_i to ensure functional equivalence. This is termed as *Clifford twirling* or *Pauli twirling* depending on the group from which G_i, G_j are sampled. The average channel formed by many such samples behave like a unitary channel.

Clifford twirling leads to an average channel which is depolarizing in nature. Therefore, the ML decoder from [BM+24] suffices in that scenario. However, Clifford group consists of CNOT gates, which are significantly more noisy ($\sim 100\times$) than single qubit gates. This makes Clifford twirling impractical since it will increase the noise in the system by adding more CNOT gates via sampling [MW22]. Hence, we resort to Pauli twirling, where all G_i, G_j are single qubit gates. Now, Pauli twirling results in an *asymmetric* unitary noise channel of the form

$$\rho \rightarrow (1 - p_x - p_y - p_z)\rho + p_x X\rho X^\dagger + p_y Y\rho Y^\dagger + p_Z Z\rho Z^\dagger$$

where p_x, p_y and p_z are unequal. In this study, we consider the asymmetric noise model derived from Pauli twirling of amplitude damping, and amplitude-phase damping noise models. Note that twirling of phase damping noise model alone leads to a channel with $p_x = p_y = 0$, and $p_z = \frac{\gamma}{2}$.

Twirling amplitude damping channel

A qubit, initially prepared in the excited state, tends to spontaneously emit energy and return to its ground state if left undisturbed. This phenomenon is known as

amplitude damping and is quantified by the parameter T_1, which is equivalent to the half-life of decay. The Kraus operators for this channel are [Pre98]

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma_A} \end{pmatrix} \quad K_1 = \begin{pmatrix} 0 & \sqrt{\gamma_A} \\ 0 & 0 \end{pmatrix}$$

where $\gamma_A = 1 - exp(-t/T_1)$, t being the time of computation or idle time. In other words, the probability that the state remains unchanged after time t is γ_A. Pauli twirling produces a unitary channel with $p_x = \gamma_A/4$, $p_y = \gamma_A/4$, $p_z = (1 - \sqrt{1-\gamma_A})^2/4$.

Twirling amplitude-phase damping channel

The different energy states of a qubit in superposition tend to accumulate different phase changes, leading to a phase difference between the energy states. This phenomenon is known as phase damping, and is characterized by the parameter T_2. A qubit in superposition is thus affected both by amplitude and phase damping noise. The Kraus operators for this combined channel are [Pre98]

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{(1-\gamma_A)(1-\gamma_P)} \end{pmatrix} \quad K_1 = \begin{pmatrix} 0 & \sqrt{\gamma_A} \\ 0 & 0 \end{pmatrix}$$
$$K_2 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{(1-\gamma_A)(1-\gamma_P)} \end{pmatrix}$$

where $\gamma_P = 1 - exp(-t/T_2)$, t being the time. Pauli twirling produces a unitary channel with $p_x = \gamma_A/4$, $p_y = \gamma_A/4$, $p_z = (1 - \gamma_A - \gamma_P + \gamma_A\gamma_P)/2$.

2.2 Circuit cutting

Circuit cutting stands out as an effective method to reduce system noise. This technique involves partitioning a quantum circuit into smaller subcircuits, computing each subcircuit independently, and then using classical post-processing to reconstruct the original output distribution [PH+20]. Initially proposed as a way to execute larger circuits on smaller devices, circuit cutting has since been shown to reduce noise because each subcircuit contains fewer qubits and gates [ST+21, BS+22, MW22]. For instance, in one study, circuit cutting was used to obtain a more accurate estimation of the ground state energy of the nearest neighbor Hamiltonian by computing each subcircuit on the least-busy device, although the noise profile of the subcircuits was not considered [KM+23]. Another study introduced a scheduling framework to assign circuits to multiple devices, aiming to minimize overall execution time without considering the noise profile of the hardware [CD+22].

How to cut a quantum circuit

Due to limitations in the size of current hardware, methods to partition a circuit into multiple smaller subcircuits have been studied extensively. These methods include splitting the problem itself to execute multiple smaller subcircuits (e.g. entanglement forging [EM+22]), cutting the circuit between two gates to create multiple tomographic instances of smaller subcircuits (called wire cutting [PH+20]) or replacing two-qubit gates by multiple instances of single qubit operation and feedforward classical communication (called gate cutting [MF21]). In this manuscript, we shall stick to wire cutting only, and use the term *circuit cutting* to imply wire cutting.

Given a circuit Φ, let us denote the expectation value of some observable A as $\Phi(A)$. Note that, for any observable A, it is possible to write [TT+21]

$$A = \frac{Tr\{A.I\}I + Tr\{A.X\}X + Tr\{A.Y\}Y + Tr\{A.Z\}Z}{2}$$

where I, X, Y, Z are the Pauli operators [NC10].
In other words, $\Phi(A) = \frac{1}{2}\sum_{P\in\{I,X,Y,Z\}} c_P \Phi_P(A)$, where $\Phi_P(A) = Tr\{AP\}\rho_P$. Here ρ_P denotes the eigenstates of the Pauli operator P and c_P denotes the eigenvalue. Note that the mathematical expression $Tr\{AP\}\rho_P$ takes instances of both subcircuits into account where the former is measured in basis P and the latter is prepared in the state ρ_P. Since there are two eigenstates corresponding to each Pauli operator, this method results in four subcircuit instances for measurement basis and eight for preparation state. The uncut expectation value (or probability distribution) is obtained via classical postprocessing.

In [TT+21] the authors showed that the previous representation of the observable A is tomographically over-complete; It is possible to have a more succinct representation of $\Phi(A) = \sum_i Tr\{AO_i\}\rho_i$, where $O_i \in \{X, Y, Z\}$ and $\rho_i \in \{|0\rangle, |1\rangle, |+\rangle, |+i\rangle\}$. These two sets O_i and ρ_i are tomographically complete and hence denote the minimum number of subcircuits necessary. Here, there are three subcircuit instances for measurement basis and four for preparation state. A general drawback of cutting is that the classical postprocessing time scales exponentially in the number of cuts when the full probability distribution needs to be reconstructed. Therefore, this method is suitable only for circuits that can be split into disjoint subcircuits using a small (ideally constant) number of cuts only.

Let us consider a RealAmplitudes [KM+17] circuit with linear reverse entanglement with a single repetition. An n-qubit RealAmplitudes circuit consists of $n-1$ CNOT gates and two layers of R_y gates, resulting in $2n$ parameters. Fig. 2.1 shows circuit-cutting of a 6-qubit RealAmplitudes circuit resulting in two subcircuits. The cut is denoted by the dotted red line. Here ρ_i and O_i have similar meaning as discussed above. Therefore, there are three variants of the first subcircuit for $O_i = X, Y, Z$, and four of the second for $\rho_i = |0\rangle, |1\rangle, |+\rangle, |+i\rangle$.

Since each subcircuit has a lower number of qubits and/or gates, the noise on each subcircuit is expected to be lower. Hence circuit cutting is often used as a method

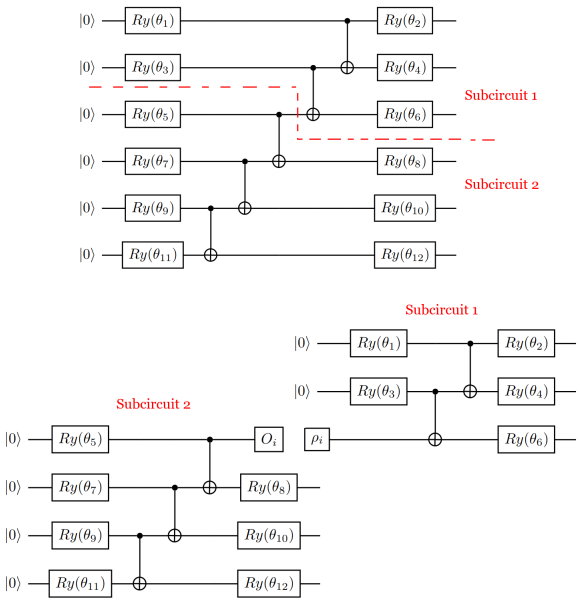


Figure 2.1: Circuit cutting for a 6-qubit RealAmplitudes ansatz, with single repetition and reverse-linear entanglement [KM+17], into two subcircuits.

to lower the noise in the system [TT+21, BS+22, MW22, KM+23]. In other words, the motivation for circuit cutting is not only the ability to run bigger circuits on smaller hardware but also to lower the noise in the system at the cost of some classical post-processing.

2.3 Circuit placement and selection of good qubits

In current quantum devices, a two-qubit operation is possible only between nearest neighbours. For example, Fig. 2.2 shows the coupling map of a 5-qubit IBM Quantum device. Here a two-qubit operation is possible between qubits 0 and 1, but not between 0 and 2 since the latter are not neighbours. In order to perform a two-qubit operation between qubits 0 and 2, they must be adjacent to each other using SWAP gates. A general requirement of placement and scheduling algorithms [ZD+23, BC20, LZF20, SD+20, AA+19, ZW17, KH+19, CS+19, MB+19]

is to minimize the number of SWAP gates.

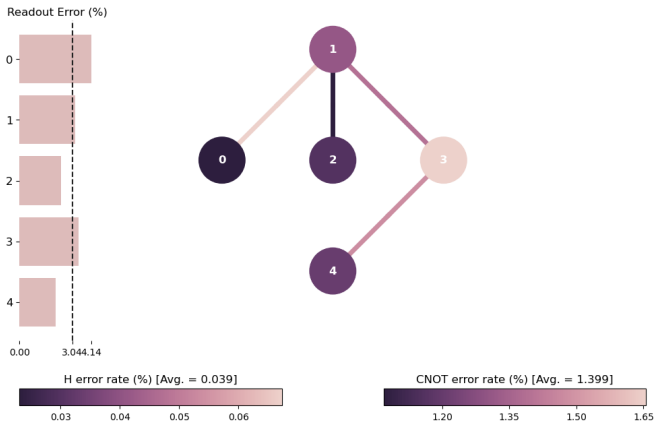


Figure 2.2: The coupling map and error distribution of a 5 qubit IBM Quantum device *Belem*.

Although the aim of the placement is to minimize the number of SWAP gates, Fig. 2.2 clearly shows that the noise profile of all the qubits is not the same. Therefore, it is important to try to involve the less noisy, or *good*, qubits from the hardware for placement. However, selecting *good* qubits for placement may lead to increased SWAP gates if the *good* qubits are not adjacent. Therefore, minimization of SWAP gates and selection of *good* qubits can often be contradictory requirements in placement.

In [NT23], the authors proposed a two-step solution for this. In the first step, also known as *transpilation*, the placement algorithm focuses on minimizing the number of SWAP gates without considering the noise profile of the hardware. As a second step, a list of isomorphisms of the transpiled circuit graph on the hardware graph is generated (refer to Fig. 2.3). Each of these isomorphisms is also called *layout*. Finally, the noise profile of each layout is calculated from the calibration data of the hardware to assign a score Q which is an indicator of the quality of the layout. The layout having the lowest score, which corresponds to the best quality, is selected. This entire process has been named *mapomatic* by the authors. We use *mapomatic* for the selection of the best qubit placement for a given circuit.

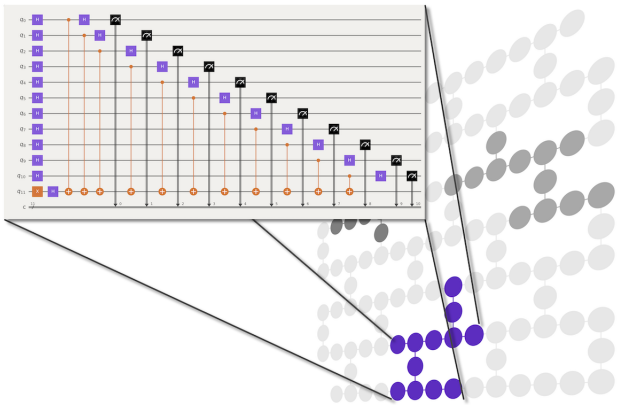


Figure 2.3: An example of *mapomatic* [NK+21] to find the best placement of a circuit on hardware. This figure is obtained from the GitHub repository of mapomatic (https://github.com/Qiskit-Partners/mapomatic).

For a set of hardware, Table 2.1 shows the least mapomatic score and the corresponding layout for placement of a 6-qubit RealAmplitudes circuit (Fig. 2.1). In other words, each layout shown in the table implies that both the number of SWAP gates and the noise will be minimized if the circuit is placed on those qubits of the hardware. A layout is generally represented as an array l, where $l[k]$ denotes the qubit of the hardware on which the $k-th$ qubit of the circuit is mapped. For example, from Table 2.1, in IBMQ Hanoi, the qubits 0, 1, 2, 3, 4, and 5 of the 6-qubit Real Amplitudes circuit are respectively mapped to physical qubits 0, 1, 2, 4, 7 and 6. From this table, we get that IBMQ Kolkata is the best hardware with layout [22, 25, 26, 24, 23, 21] to execute the 6-qubit Real Amplitudes circuit.

Table 2.1: *Mapomatic* score for the best layout of the 6-qubit RealAmplitudes circuit (Fig: 2.1) corresponding to each of the available hardware

Backend	# Qubits	Corresponding layout	Mapomatic score
IBMQ Hanoi	27	[0, 1, 2, 4, 7, 6]	0.099
IBMQ Mumbai	27	[6, 7, 4, 10, 12, 13]	0.183
IBMQ Cairo	27	[13, 12, 10, 15, 18, 17]	0.105
IBMQ Kolkata	27	[22, 25, 26, 24, 23, 21]	0.084
IBMQ Guadalupe	16	[15, 12, 13, 10, 7, 6]	0.142
IBMQ Lagos	7	[0, 1, 2, 3, 5, 6]	0.093
IBMQ Nairobi	7	[0, 1, 2, 3, 5, 4]	0.193

2.3.1 Intra Device Scheduling

As the demand for quantum computing is rising, the users often face long queuing time as their jobs need to wait till the execution of all the previously submitted jobs have been completed. Consequently, there is a pressing need to enhance the efficiency and throughput of quantum computers to improve user experience. In this work, we study the benefits and challenges of executing more than one quantum circuit, i.e., more than one job simultaneously on the same hardware to improve the throughput, as well as the hardware utilization, without yielding on the quality of outcome.

Let us motivate the problem with an example. Consider a 15-qubit circuit which is to be executed on a 27 qubit device as shown in Fig 4.1. Thus 12 qubits of the device remain unused, which could have been utilized to execute simultaneously some other circuit(s) requiring ≤ 12 qubits – thus improving the throughput and the hardware utilization.

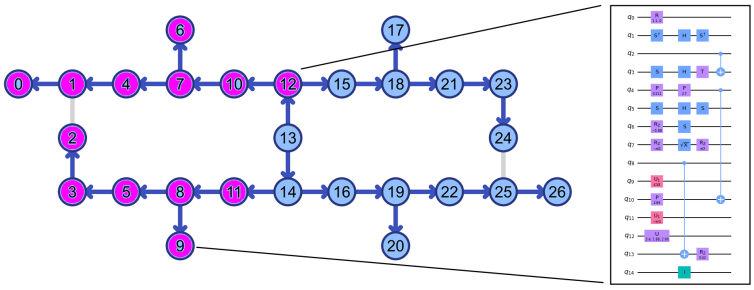


Figure 2.4: An example of a 15-qubit circuit assigned to a 27- qubit hardware. The used qubits are shown in purple while the unused qubits are shown in blue. The hardware still has room to accommodate one or more quantum circuit(s) using the free qubits.

Simultaneous execution of multiple circuits on a single hardware is not without challenge. Previous studies [CBSG17] do not consider the effect of noise arising due to simultaneous execution of circuits. First, when a circuit is mapped to a hardware, the requirement is to reduce the number of SWAP gates, as well as to

use a layout with minimal noise profile [T+22]. However, when multiple circuits are placed simultaneously, it is likely that all of them cannot be placed on their corresponding best layout, leading to degradation in the quality of the outcome of the computation. Furthermore, if two circuits are computed on neighbouring qubits, then there is a possibility of crosstalk affecting the quality of the computation for both of them.

Minimizing the degradation in quality due to worse layout selection and crosstalk, while maximizing the throughput of the hardware presents a multifaceted challenge since these two objectives conflict with one another.

2.4 Quantum Error Correction

Quantum states are, however, very prone to errors. Being vectors in Hilbert space, even the slightest unwanted rotation occurring due to interaction with the environment introduces error in the quantum system. It was shown by Shor [Sho95] that any unitary quantum error can be expressed as a linear combination of the Pauli matrices (I, X, Y, Z) [1]. Hence, if a quantum error correcting code (QECC) can correct the Pauli errors, then it can also correct any unitary error. The 9-qubit code [Sho95], 7-qubit code [Ste96] and 5-qubit code [LMPZ96] are some early QECCs. The 5-qubit code is optimum in the number of qubits .

The circuit realization of the above-mentioned QECCs comprise multiple operations involving qubits which are not adjacent to each other. Operation on two non-adjacent qubits is both slow and error-prone, due to the multiple swap operations required. Surface code was introduced to overcome this drawback, known as the Nearest Neighbour (NN) problem, by placing the qubits in a 2D grid-like structure [BK98], and the operations for error correction are performed only between adjacent qubits. Protocols are formulated for error recovery, and the efficacy of

[1] An operator U is unitary operator if $U^\dagger U = UU^\dagger = I$, where I is the identity element and $U^\dagger$ is the adjoint of U. An operator is hermitian if $U = U^\dagger$. Note that, hermitian operator is a subset of unitary operator. Pauli matrices are hermitian.

these protocols were studied in [DKLP02] which is reviewed in a simplied manner by [FSG09]. An improved decoding algorithm for the surface code is formulated in [WFH11].

A QECC encodes $n > 1$ physical qubits into $m < n$ logical qubits, where the latter are expected to be more secure under noise. A decoder (which is a classical process), on the other hand, detects the error present in the logical qubit. Decoding is followed by another step where the correction is applied physically and classically (or in some cases noted logically only [RFV+17]). A distance d QECC can correct $\lfloor \frac{d}{2} \rfloor$ errors on the physical qubits, keeping the logical qubit error free. However, the logical qubit can become erroneous as well if more errors occur. This is termed as *logical error*. The errors may occur due to interaction with the environment, or faulty decoding. While the first issue may be tackled with a QECC having a larger d, the latter can pose a serious threat towards building error corrected qubits. The performance of a decoder for a QECC is assessed by two parameters [FWH12], namely: (i) *pseudo-threshold*, which is the probability of physical error below which error-correction leads to a lower logical error probability, and (ii) *threshold*, which is the probability of physical error beyond which increasing d leads to higher logical error probability.

We have illustrated these two parameters in Fig. 2.5. A decoder with higher pseudo-threshold and threshold is desirable.

Apart from the accuracy of decoding, the time required is also important. In a fault-tolerant quantum computer, the qubits are encoded only once at the beginning of the computation, whereas they are decoded several times during the computation. Therefore, decoding time is critical [NC10, MB+16]. The most popular decoding algorithm for surface codes is Blossom Decoder [Edm65a] based on $\mathcal{O}(N^4)))$ time Minimum Weight Perfect Matching (MWPM) algorithm, N being the number of qubits. Recently machine learning (ML) has been used for decoding in linear time [VBA19]. A baseline decoding algorithms complemented by different kinds of deep neural decoders was introduced by [CR18] and applied to analyze the common fault-tolerant error correction protocols such as the surface code. The

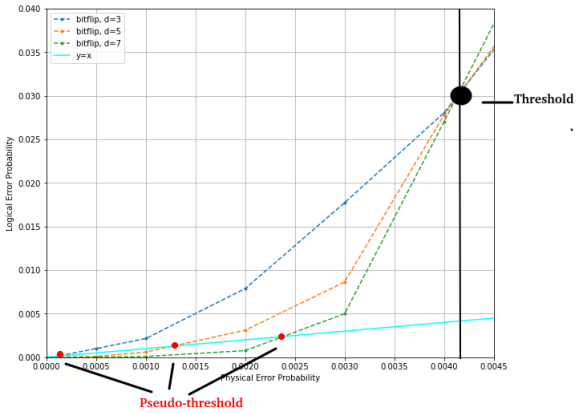


Figure 2.5: Threshold (black dot) and Pseudo-threshold (red dot) of a MWPM decoder for a topological (heavy hexagon Code) QECC, with performance comparison for distance 3 (blue), distance 5 (orange), and distance 7 (green). The cyan straight line $y = x$ is for equal probabilities of physical qubit error and logical error.

decoding problem is reduced to a classification problem that a feedforward neural network can solve, in [VCB17], for small code distances. Reinforcement Learning based decoders for Fault-Tolerant Quantum Computation were proposed in [S+18]. It has been observed that the MWPM based decoder performs satisfactorily when the error probability of the system is low, as it always tries to find the minimum number of errors that can generate the observed syndrome. But occurrence of error(s) in the system during decoding is ignored, which the ML based approaches do consider. Therefore, ML based decoders are expected to perform at least as well as MWPM based method and in less time.

ML based decoders can tackle errors incorporated due to faulty decoding upto some extent. This is achieved by introducing two-level decoding, where the low level is a traditional decoder (need not be an ML decoder [VCB17]), and the high level (necessarily ML decoder) predicts any logical errors that may have resulted during decoding. In [VBA19] the authors have used ML for both low and high-level decoders. However, it is unclear whether their noise model considers errors in a single or multiple steps in the error correction cycle of surface code (see Fig: 2.7). Moreover, the performance of ML decoders for asymmetric noise (which is a more

realistic noise model [IM07]), and whether the usage of more sophisticated ML models can significantly enhance the performance of the decoder, remains largely unanswered.

2.4.1 Stabilizer Formulation of Surface Code

Gottesman [Got97] proposed the stabilizer formulation for error correction. A set of mutually commuting operators $M_1, \ldots, M_r$, where each $M_i \in \{I, X, Z, Y\}^{\otimes n}$, is said to stabilize an n-qubit quantum state $|\psi\rangle$ if $M_i |\psi\rangle = |\psi\rangle$, $\forall\ i$ [Got97]. An error E is said to correctable by a QECC, if there exist stabilizers $M_e \subseteq \{M_1, \ldots, M_r\}$, such that $M_e\ (E |\psi\rangle) = -E |\psi\rangle$.

A QECC is called degenerate if there exist errors $e_1 \neq e_2$ such that $e_1 |\psi\rangle = e_2 |\psi\rangle$ where $|\psi\rangle$ is the codeword. It is not possible to distinguish between such errors in a degenerate code. Surface code is a degenerate stabilizer code. Surface code is implemented on a two-dimensional array of physical qubits. The data qubits (in which the quantum information is stored) are placed on the vertices, and the faces are the stabilizers (refer Fig. 2.6). The qubits associated with the stabilizers are also called measure qubits. These are of two types : *Measure-Z* (M-Z) and *Measure-X* (M-X). Each data qubit interacts with four measure qubits — two M-Z and two M-X, and each measure qubit, in its turn, interacts with four data qubits (Fig. 2.6). An M-Z (M-X) qubit forces its neighboring data qubits a, b, c and d into an eigenstate of the operator product $Z_a Z_b Z_c Z_d$ ($X_a X_b X_c X_d$), where Z_i (X_i) implies Z (X) measurement on qubit i. Pauli-X and Pauli-Z errors are detected by the Z- and X- stabilizers respectively (Fig. 2.6). An X (Z) logical operator is any continuous string of X (Z) errors that connect the top (left) and bottom (right) boundaries of the 2D array. The number of measure qubits,

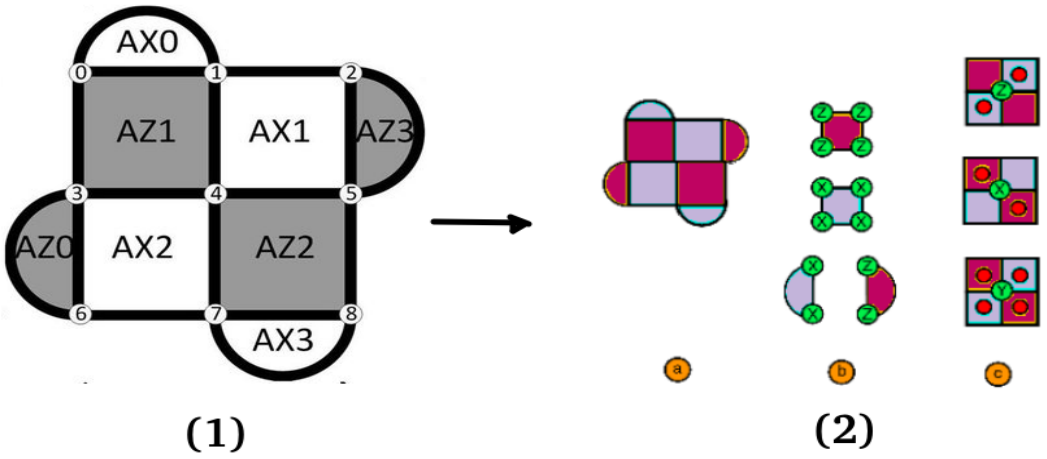


Figure 2.6: (a) Distance 3 surface code, where the numbered circles (0 - 8) are the physical qubits, white plaquettes are X stabilizers (i.e., M-X qubits $AX0$, $AX1$, $AX2$, $AX3$), gray plaquettes are Z stabilizers (i.e., M-Z qubits $AZ0$, $AZ1$, $AZ2$, $AZ3$); (b) the syndromes are defined in a $d \times d$ lattice (d=3), with physical qubits on the vertices and plaquette stabilizers (measure qubits) as faces: (i) pink (purple) plaquettes indicate stabilizers which check the Z (X) parity of qubits on the vertices of the plaquettes as shown in (ii), (iii) green circles indicate errors and red circles violated stabilizers (i.e., syndromes [VCB17])

and hence the number of stabilizers, is one less than the number of data qubits when encoding a single logical qubit of information. An error-correcting code can correct up to t errors if its distance $d \geq 2t+1$. A distance 3 surface code consists of 9 data qubits and 8 measure qubits (Fig. 2.6). Thus a total of 17 qubits encode a single logical qubit, and hence the distance 3 surface code is also called SC17.

The circuit representations of the decoding corresponding to a single M-Z qubit and an M-X qubit are shown in Fig. 2.7. Since the same measure-qubit is shared by multiple data qubits, different errors can lead to the same syndrome in surface code. Hence the mapping from syndrome to error is not one-to-one, as illustrated in Fig. 2.8. This often leads to poor decoding performance by decoders. In fact,

if a decoder misjudges an error e_1 for some other error e_2, it can so happen that $e_1 \oplus e_2$ leads to a logical error. Therefore, not only the presence of physical errors, but also incorrect decoding can lead to uncorrectable logical errors as well. The goal of designing a decoder, thus, is to reduce the probability of logical error for some physical error probability.

The performance of a decoder is measured in terms of pseudo-threshold and threshold. With increasing code distance, the pseudo-threshold for a particular decoder also increases, which supports the intuition that using larger distance gives better protection from noise. On the other hand, the threshold does not change with respect to the distance because a decoder for a particular surface code yields a fixed threshold. The higher are the values of these parameters, the better is the performance of the decoder. Of these two parameters, the pseudo threshold is lower than the threshold for a decoder. The reason is that error correction is effective below the pseudo-threshold point, and coding theory asserts [Hil86] that in this region, increasing the distance of the code leads to higher suppression of logical errors. Therefore, if the threshold point is below the pseudo-threshold point it violates coding theory. Hence, it is more important for a decoder to have a higher pseudo-threshold than a higher threshold, since, beyond this error probability, QECC no longer provides any improvement in suppression of errors.

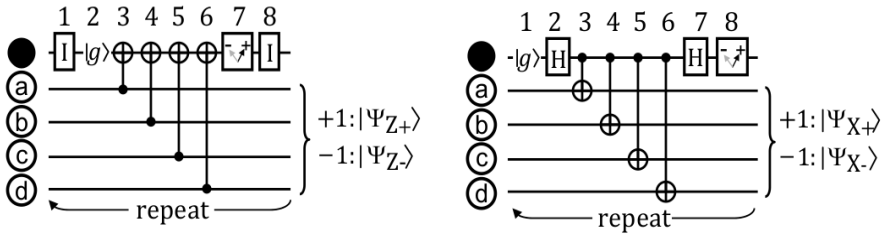


Figure 2.7: Quantum circuit for a single cycle of surface code. (a) circuit for M-Z qubit, (b) circuit for M-X qubit [FWH12]

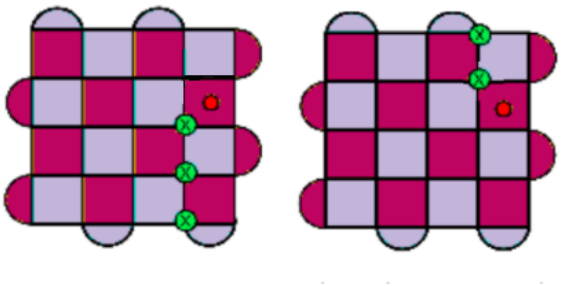


Figure 2.8: Example of surface code for d=5, where two errors produce the same syndrome [VCB17]. Pink (Purple) plaquettes indicate stabilizers which check the Z (X) parity of qubits on the vertices of the plaquette. Green circles are used to indicate errors and red circles to indicate violated stabilizers.

Let there be n physical qubits in a logical qubit (eg. $n{=}d^2$ for surface code). A logical error can occur only when at least d of the n physical qubits are erroneous. Nevertheless, the presence of d or more physical errors does not necessarily imply the presence of a logical error. If p and p_L are respectively the probability of physical and logical error, then

$$p_L \leq \Sigma_i p^i, \text{ for } d \leq i \leq n$$

Moreover, incorrect decoding itself can lead to logical errors. This can happen when the decoder fails to detect the actual physical errors and thus incorporates more errors during correction. Once again, not every incorrect decoding leads to a logical error. Therefore, if p_d is the probability of failure of the decoder, then

$$p_L \leq \Sigma_i p^i + f(p_d), \text{ for } d \leq i \leq n$$

where $f(p_d)$ is a function of the probability of failure of the decoder. The function $f(p_d)$ may vary with the decoder, hence the logical error probability may differ, resulting in different values of pseudo-threshold and threshold.

Machine learning based syndrome decoding for surface code

Machine Learning is a branch of artificial intelligence where a machine learns without being explicitly programmed. Depending on the type of training data (labeled /unlabeled /combined), the ML algorithm can vary (supervised /unsupervised /semi-supervised). It has a plethora of applications domains such as soil properties prediction [KG22], human pose estimation [SB22], object recognition [JW22], video tracking [PR20], prediction of the efficacy of online sale [SM21] etc. Here, we employ machine learning to decode error syndrome(s) for quantum error correction.

Advantages of Machine learning based syndrome decoder

Classical algorithms for decoding, such as Minimum Weight Perfect Matching (MWPM), may perform poorly in certain cases. For example, MWPM tries to find a minimum number of errors that can recreate the error syndrome obtained without considering the probability of error.

If $|\psi\rangle_{1,2,\ldots,n}$ is an n-qubit codeword, and we consider the error generation on this codeword is a stochastic map $S(p_1, p_2, \ldots, p_n)$, where p_i is the probability of error on qubit i (we can further write p_i in terms of the probability of Pauli errors), then the error state $|\psi\rangle_e = S(p_1, p_2, \ldots, p_n)|\psi\rangle_{1,2,\ldots,n}$. Now, for a distance d surface code with t types of errors (t=4 for depolarization, 2 for bit/phase flip), there are t^{d^2} possible errors and t^{d^2-1} possible syndromes. Therefore, multiple errors $\mathcal{E}=\{e_1, e_2, \ldots, e_l\}$ lead to the same syndrome, and detecting a syndrome cannot uniquely specify the type of error causing it. Since Pauli errors are hermitian, correction is simply applying the same error once more. If the choice of error is not perfect, then the system ends up with (probably) more error than before after the correction step.

MWPM, being a deterministic algorithm, does not consider the Stochastic map. It assumes that the error probability is low, and always finds the minimum weight error $e_{min} \in \mathcal{E}$ that creates the observed syndrome. On the other hand, an ML

decoder learns the probabilities $p_1, p_2, \ldots, p_n$ from the training phase. Thus, this decoder finds most likely error $e_{ml} \in \mathcal{E}$ that can cause observed syndrome depending on the Stochastic Map.

Furthermore, the time complexity of MWPM grows as $\mathcal{O}(N^4))$ where N is the number of qubits. Lookup Tables have been used for decoding as well [V+18]. While Lookup Table Decoder is sometimes better than MWPM in performance, its complexity scales as $\mathcal{O}(4^N)$ which becomes infeasible even for moderate values of N. To overcome such drawbacks, ML techniques have been applied to learn the probability of error in the system and propose the best possible correction accordingly with comparatively lower time complexity. For example, [VCB17] reduced the decoding problem to a classification problem that a feed-forward neural network can solve, for small code distances. A deep neural network based decoder is proposed by [KJ17] for Stabilizer Codes. Therefore, supervised learning techniques, such as Feed-forward neural network (FFNN), Recurrent Neural Network (RNN) show that these are capable of outperforming the traditional decoding techniques.

As discussed earlier, surface code is degenerate, i.e., there exist errors $e_1 \neq e_2$ such that $e_1 |\psi\rangle = e_2 |\psi\rangle$, where $|\psi\rangle$ is the codeword. This leads to any decoder failing to distinguish between some errors e_1 and e_2. Nevertheless, that does not always lead to a logical error. For example, bit-flip error in bit 1 and bit 2 are indistinguishable. But error in decoding these two will not lead to a logical error (Refer Fig. 2.9 (a)). On the other hand, it is possible that $e_1 \oplus e_2$ leads to a logical error, i.e., the decoder may itself incorporate logical errors while correcting physical errors. For example, bit-flip error on qubit 4 is indistinguishable from those on qubits 1 and 7 together. But failure to distinguish between these two bit-flip errors leads to logical error (refer Fig. 2.9 (b)).

In general, *usually* the decoder incorporates logical errors when it fails to distinguish between $\lfloor$(d-1)/2$\rfloor$ and $\lceil$(d+1)/2$\rceil$ errors. Broadly speaking, ML can learn the probability of error and predict which of those two are more likely. This makes ML-decoder outperform other traditional decoders.

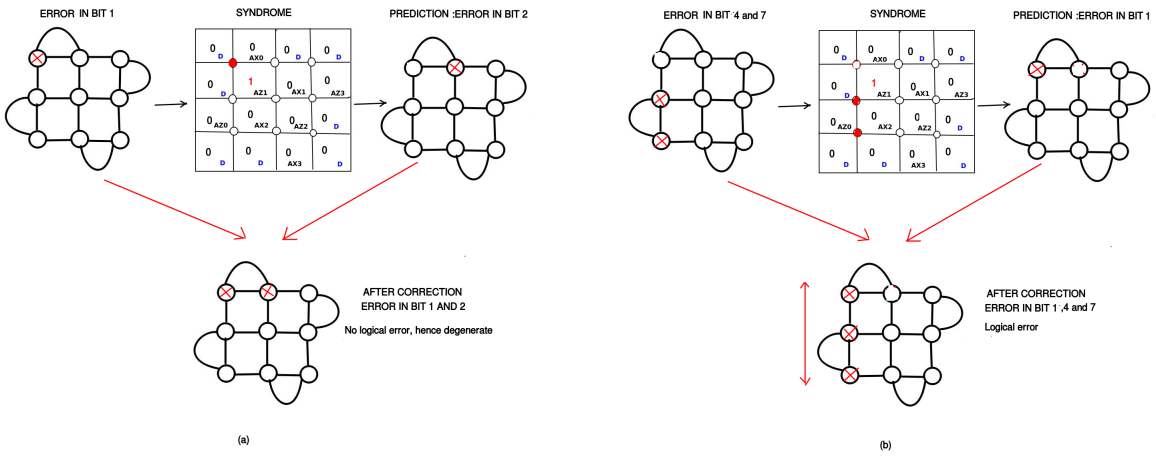


Figure 2.9: (a) No logical error and (b) Logical error due to mis-classification in low level decoding

Since a decoder itself can incorporate logical errors, two stages of decoders, namely low level followed by high level decoder, have been applied where

- Low level decoders search for exact position of errors at the physical level.
- High level decoders attempt to correct any logical error incorporated by the correction mechanism of low level decoders.

2.4.2 Heavy Hexagon Code

Recently, industry research labs have been shifting towards the hexagonal architecture for their quantum computers. This architecture has the advantage of reducing the number of distinct frequencies, and thus crosstalk. The surface code [BK98] structure has been modified to a topological code with a heavy hexagonal structure [CZ+20] in order to become more suitable for these architectures. The heavy hexagon code [CZ+20] uses a combination of degree-two and degree-three vertices in the topology, and can be considered as a hybrid of a surface code and a Bacon-Shor code [Bac06]. This QECC reduces the distinct number of frequencies required

in their realization by introducing more ancilla qubits (termed as flag qubits) for entanglement in the syndrome measurement [CZ+20].

The authors of [CZ+20] have proposed the heavy hexagon code for QECC and have used an MWPM [Edm65a, CZ+20] decoder to evaluate the code. The asymptotic threshold for logical bit flip or X errors is 0.0045. Since phase flip or Z errors are corrected using Bacon-Shor type stabilizers, no threshold for Z errors can be defined as such. To the best of our knowledge, there is no ML based decoder for the heavy hexagon code, so the threshold 0.0045 is considered the state-of-the-art.

For a distance d QECC, if more than $\lfloor \frac{d-1}{2} \rfloor$ errors occur, then the QECC fails to correct those errors, leading to an incorrect logical state called a *logical error*. A logical error can occur due to incorrect decoding as well. Logical errors pose a serious threat towards building error-corrected qubits since these remain undetected, and are retained in the logical state of the system. Given a QECC, the goal therefore is to design a decoder which reduces the probability of logical error.

Let us now describe the heavy hexagon code structure, the noise model used in this work, and the motivation for using machine learning based decoder.

Structure of Heavy Hexagon Code

This QECC encodes a logical qubit over a hexagonal lattice. As qubits are present on both the vertices and edges of the lattice, the term heavy is used. This is a combination of degree-2 and degree-3 qubits hence there is a huge improvement in terms of average qubit degree in comparison with surface code structure which has qubits of degree-4 [CZ+20]. Fig. 2.10 shows the lattice for a distance-3 heavy hexagon code encoding one logical qubit.

The heavy hexagon code is a combination of surface code and subsystem code (Bacon Shor code) [CZ+20]. A subsystem code is defined by G, a set of gauge operators where $\forall\ g \in G$, $|\psi\rangle \equiv g\,|\psi\rangle$ [Bac06]. A gauge operator takes a codeword to an equivalent subsystem. In other words, a codespace in a subsystem code consists

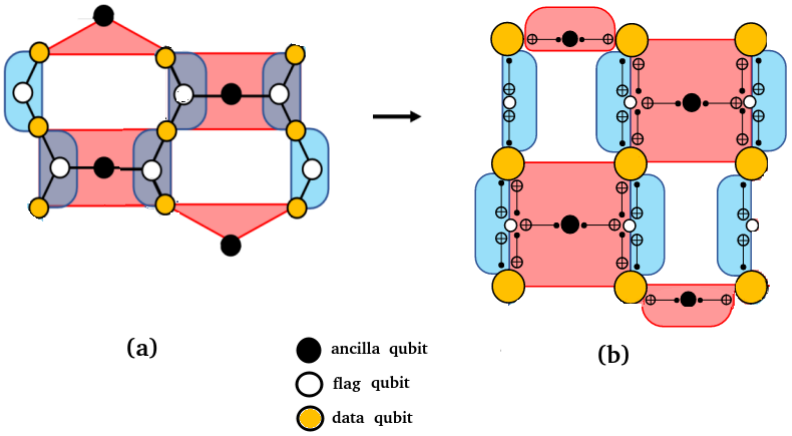


Figure 2.10: Distance 3 heavy hexagon code encoding one logical qubit: (a) the hexagonal structure, (b) the circuit illustration of the heavy hexagon code with the CNOT gates. Here yellow, white and black circles represents data, flag and ancilla qubits respectively; black ancilla qubits are for measuring the X (red face or plaquette) and Z (blue face or strip) gauge generators. The product of two Z gauge generators at each white plaquette forms a Z stabilizer. The figure is adapted from the reference [CZ+20].

of multiple equivalent subsystems. It is to be noted that the gauge operators are not necessarily commutative. The product of two or more gauge operators forms a stabilizer, which keeps the codeword unchanged.

Gauge generators

For the heavy hexagon code, its gauge generators which form the gauge group are defined in terms of Pauli operators as

$$\langle Z_{i,j}Z_{i+1,j},\ X_{i,j}X_{i,j+1}X_{i+1,j}X_{i+1,j+1},\ X_{1,2m-1}X_{1,2m},\ X_{d,2m}X_{d,2m+1}\rangle,$$

where $i \in \{1, 2, \ldots, d-1\}$, $j \in \{1, 2, \ldots, d\}$ and $m \in \{1, 2, \ldots, (d-1)/2)\}$. Further, for the second type of the gauge generators $X_{i,j}X_{i,j+1}X_{i+1,j}X_{i+1,j+1}$, $(i+j)$ has to be odd. A gauge generator $g_{i,j}$ is a gauge operator acting on the $(i,j)^{th}$ data qubit in the lattice. The product of one or more gauge generators can form a *gauge operator*.

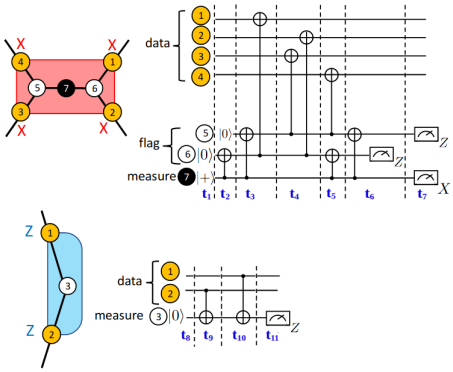


Figure 2.11: Circuits for measuring X and Z gauge generators in the heavy hexagon code where t_i denotes the i^{th} time step. Two flag qubits (white circles) measure a X gauge generator having a weight of 4 and one flag qubit measures a Z gauge generator having a weight of 2 [CZ+20].

Fig. 2.11 shows the X and Z gauge generators along with the circuits [CZ+20] for measuring these. A single error correction cycle requires 11 time-steps (7 for X and 4 for Z). The weight of a gauge operator g is defined as the number of non-identity Pauli operators in g. In Fig. 2.12, the Z and X gauge generators, indicated in blue and red, can correct bit flip and phase flip errors respectively. This is explained with detailed examples in Appendix.

For a distance d, the number of gauge generators is $(d^2 - 1)/2$, so the number of possible gauge operators can be exponential in d, thereby syndrome decoding poses a computational challenge.

Stabilizers

The stabilizer group of the heavy hexagon code [CZ+20] is defined as

$$\langle Z_{i,j}Z_{i,j+1}Z_{i+1,j}Z_{i+1,j+1},\ Z_{2m-1,d}Z_{2m,d},\ Z_{2m,1}Z_{2m+1,1},\ \Pi_i X_{i,j}X_{i,j+1}\rangle$$

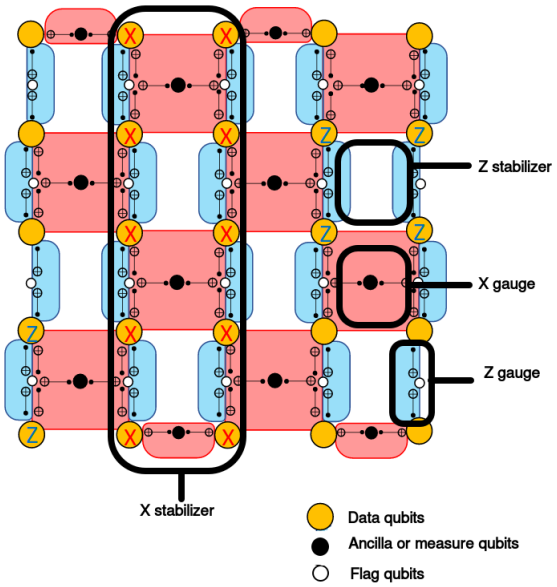


Figure 2.12: The stabilizers and gauge generators for a distance 5 heavy hexagon code: the weight-4 X gauge generators are in the red plaquettes, while the weight-2 X gauge generators are on the upper and lower boundaries. Each blue strip denotes weight-2 Z gauge generators. A vertical strip of two adjacent columns with X gauge generators form an X stabilizer. The weight-4 Z gauge generators in the white plaquettes, and weight-2 Z gauge generators on the left and right boundaries are themselves Z stabilizers. [CZ+20].

where $i \in \{1, 2, \ldots, d-1\}$, $j \in \{1, 2, \ldots, d\}$ and $m \in \{1, 2, \ldots, (d-1)/2)\}$. Further, for the first type of stabilizers $Z_{i,j}Z_{i,j+1}Z_{i+1,j}Z_{i+1,j+1}$ having weight 4, $(i+j)$ has to be even. The measurement outcome of one such stabilizer is the product of the measured eigenvalues of the two weight-two gauge generators $Z_{i,j}Z_{i+1,j}$ and $Z_{i,j+1}Z_{i+1,j+1}$, $i+j$ being even.

Enumerating Gauge Generators of Heavy Hexagonal QECC for $d = 5$

The 20 Z gauge generators of the type $Z_{i,j}Z_{i+1,j}$ in Fig. 2.12 are given in Table 2.2. The 8 X gauge generators of the type $X_{i,j}X_{i,j+1}X_{i+1,j}X_{i+1,j+1}$ (represented by red squares in Fig. 2.12) are listed in Table 2.3. The 2 X gauge generators of the type $X_{1,2m-1}X_{1,2m}$ and 2 X gauge generators of the type $X_{d,2m}X_{d,2m+1}$ (represented by

red semicircles in Fig. 2.12) are shown in Table 2.4. Hence, there are a total of 12 X gauge generators in Fig. 2.12. In this appendix, we list all the gauge generators for a distance heavy hexagonal code. Here, $i \in \{1,\ 2,\ 3,\ 4\}$, $j \in \{1,\ 2,\ 3,\ 4,\ 5\}$ and $m \in \{1,\ 2\}$.

Table 2.2: The 20 Z gauge generators $Z_{i,j}Z_{i+1,j}$ with weight 2

Gauge Generator	Position in lattice (blue semicircle)
$Z_{1,1}Z_{2,1}$	first row first column (top left)
$Z_{2,1}Z_{3,1}$	second row first column
...	
...	
$Z_{4,5}Z_{5,5}$	last row last column

Table 2.3: The 8 X gauge generators $X_{i,j}X_{i,j+1}X_{i+1,j}X_{i+1,j+1}$ with weight 4

Gauge Generator	Position in lattice (red square)
$X_{1,2}X_{1,3}X_{2,2}X_{2,3}$	first row second column
$X_{1,4}X_{1,5}X_{2,4}X_{2,5}$	first row fourth column
$X_{2,1}X_{2,2}X_{3,1}X_{3,2}$	second row second column
$X_{2,3}X_{2,4}X_{3,3}X_{3,4}$	second row fourth column
$X_{3,2}X_{3,3}X_{4,2}X_{4,3}$	third row second column
$X_{3,4}X_{3,5}X_{4,4}X_{4,5}$	third row fourth column
$X_{4,1}X_{4,2}X_{5,1}X_{5,2}$	fourth row second column
$X_{4,3}X_{4,4}X_{5,3}X_{5,4}$	fourth row fourth column

Table 2.4: The 4 X gauge generators $X_{1,2m-1}X_{1,2m}$ and $X_{d,2m}X_{d,2m+1}$ with weight 2

Gauge Generator	Position in lattice (red semicircle)
$X_{1,1}X_{1,2}$	first row first column (top left)
$X_{1,3}X_{1,4}$	first row third column
$X_{5,2}X_{5,3}$	fifth row second column
$X_{5,4}X_{5,5}$	fifth row fifth column

Enumerating Stabilizers of Heavy Hexagonal QECC for $d = 5$

The 8 Z stabilizers of the type $Z_{i,j}Z_{i,j+1}Z_{i+1,j}Z_{i+1,j+1}$ in Fig. 2.12 are shown in Table 2.5. The 2 Z stabilizers of the type $Z_{2m-1,d}Z_{2m,d}$ and 2 Z stabilizers of the type $Z_{2m,1}Z_{2m+1,1}$ (represented by blue semicircles in Fig. 2.12) are given in Table 2.6. Hence, there are a total of 12 Z stabilizers in Fig. 2.12. Next, the 4 X stabilizers of the type $\Pi_i X_{i,j}X_{i,j+1}$ in Fig. 2.12 are shown in Table 2.7.

Table 2.5: The 8 Z stabilizers $Z_{i,j}Z_{i,j+1}Z_{i+1,j}Z_{i+1,j+1}$ with weight 4

Stabilizer	Position in lattice (white square)
$Z_{1,1}Z_{1,2}Z_{2,1}Z_{2,2}$	first row first column
$Z_{1,3}Z_{1,4}Z_{2,3}Z_{2,4}$	first row third column
$Z_{2,2}Z_{2,3}Z_{3,2}Z_{3,3}$	second row second column
$Z_{2,4}Z_{2,5}Z_{3,4}Z_{3,5}$	second row fourth column
$Z_{3,1}Z_{3,2}Z_{4,1}Z_{4,2}$	third row first column
$Z_{3,3}Z_{3,4}Z_{4,3}Z_{4,4}$	third row third column
$Z_{4,2}Z_{4,3}Z_{5,2}Z_{5,3}$	fourth row second column
$Z_{4,4}Z_{4,5}Z_{5,4}Z_{5,5}$	fourth row fourth column

Table 2.6: The 4 Z stabilizers $Z_{2m-1,d}Z_{2m,d}$ and $Z_{2m,1}Z_{2m+1,1}$ with weight 2

Stabilizer	Position in lattice (blue semicircle)
$Z_{1,5}Z_{2,5}$	first row fifth column
$Z_{3,5}Z_{4,5}$	third row fifth column
$Z_{2,1}Z_{3,1}$	second row first column
$Z_{4,1}Z_{5,1}$	fourth row first column

For QECCs which are not subsystem codes, such as [Sho97, Ste96], an n-qubit QECC with $n - k$ stabilizers can correct upto k errors. On the other hand,

Table 2.7: The 4 X stabilizers $\Pi_i X_{i,j} X_{i,j+1}$

Stabilizer	Position in lattice (vertical strip)
$X_{1,1}X_{1,2}X_{2,1}X_{2,2}X_{3,1}X_{3,2}X_{4,1}X_{4,2}X_{5,1}X_{5,2}$	the vertical strip of the column 1 and 2
$X_{1,2}X_{1,3}X_{2,2}X_{2,3}X_{3,2}X_{3,3}X_{4,2}X_{4,3}X_{5,2}X_{5,3}$	vertical strip of the column 2 and 3
$X_{1,3}X_{1,4}X_{2,3}X_{2,4}X_{3,3}X_{3,4}X_{4,3}X_{4,4}X_{5,3}X_{5,4}$	vertical strip of the column 3 and 4
$X_{1,4}X_{1,5}X_{2,4}X_{2,5}X_{3,4}X_{3,5}X_{4,4}X_{4,5}X_{5,4}X_{5,5}$	vertical strip of the column 4 and 5

subsystem codes need fewer stabilizers [Bac06]. Errors and the corresponding syndromes do not form a one-to-one mapping. This has motivated us to design an ML based syndrome decoder, along with two efficient algorithms to identify equivalence classes for all possible errors to enhance the efficiency of the decoder.

Noise model considered for Heavy Hexagon QECC

Apart from bit flip, phase flip, depolarization error amplitude damping and amplitude-phase damping noise on data qubits, the following noise are also considered.

Measurement error

After measuring the ancilla, we get the syndrome. Depending on the measurement error probability, bit flip error is incorporated in that syndrome.

Stabilizer error

Stabilizer error is basically erroneous measure qubits. Hence bit flip errors are applied on ancilla qubits.

Moreover, a single error correction cycle in heavy hexagon code consists of eleven steps (Fig. 2.11. An error can occur on $0 \leq k \leq d^2$ data qubits in each of the eleven steps, d being the distance of the code. Therefore, the overall probability of error for depolarizing error for each error correction cycle is $1 - (1 - p)^{11}$.

Building on this foundation, we now explore error suppression strategies specifically tailored for NISQ devices in the current era.

Part I

Error suppression for NISQ devices: Current Era

CHAPTER 3

Time and noise optimization for distributed scheduling of quantum circuits

Contents

3.1 Introduction 48

3.2 Hardware Schedule for Subcircuits 49

3.3 Proposed framework 51

3.3.1 Selection of appropriate hardware 52

3.3.2 Scoring each hardware as per noise profile 53

3.3.3 Noise and Time Aware Distributed Scheduler (NoTaDS) 53

3.4 Experimental Results 57

3.4.1 Criteria for maximum execution time τ 58

3.4.2 Estimation of the execution time of a circuit 59

3.4.3 Results for 6-qubit circuits 60

3.4.4 Results for 10-qubit circuits 62

3.4.5 Variation in fidelity with the number and size of subcircuits 63

3.4.6 Results for a 28-qubit circuit 65

3.4.7 Change in fidelity with and without scheduling 66
3.5 A polynomial time solution for a restricted scenario . 69
3.6 Summary 71

3.1 Introduction

Circuit cutting is an approach that has been proven to reduce the noise in the system. This method of partitioning a circuit into multiple subcircuits, independently computing each of these and then using classical postprocessing to retrieve the uncut output distribution, was proposed primarily as a method to compute larger circuits on smaller devices [PH+20]. However, since then, multiple studies [ST+21, BS+22, MW22] establish their capability to reduce the noise in the system since each of the subcircuits involves fewer qubits and/or gates. In [KM+23], the authors obtained a more accurate estimation of ground state energy of the nearest neighbour Hamiltonian by leveraging circuit cutting and computing each subcircuit on the least-busy device. However, they did not consider the noise profile of the subcircuit. In [CD+22], the authors provided the framework of a scheduler to assign circuits to multiple hardware to minimize the overall execution time without considering the noise profile of the hardware. In this study, we propose a noise and time-aware scheduler that leverages circuit cutting and then schedules the subcircuits to multiple hardware to maximize the overall fidelity while restricting the execution time on each hardware below a predefined value τ.

Ideally, a user would want to (i) reduce the effect of noise on the quantum circuit, and (ii) execute their quantum circuit on the least noisy hardware. These two requirements are not independent, but we leverage circuit cutting and perform noise-aware scheduling respectively. Circuit cutting is known to lower the effective noise on the system, but it also leads to an increased number of subcircuit instances to be executed. A user may not have the desired execution time available on the hardware of their choice due to limited access, or due to the access time being

shared among multiple users. In our approach, we tackle the challenge of enhancing fidelity while reducing execution time by combining circuit cutting and selecting the best available hardware for each subcircuit.

Achieving the optimal trade-off between noise and execution time is a complex task since these two requirements are often orthogonal to each other. In order to minimize the noise, we would want to execute all the subcircuits on the least noisy device available, leading to an increased execution time. Conversely, to minimize the execution time, we can opt to distribute the subcircuits across all available devices without considering their individual noise profiles, leading to low fidelity. In order to address this optimization challenge, we design an integer linear program (ILP) that seeks to maximize the fidelity while conforming to a fixed maximum allowable execution time τ for each hardware. The uncut probability distribution is obtained through classical postprocessing over the outcomes of the individual subcircuits. The results obtained through our Noise and Time Aware Distributed Scheduler (NoTaDS) demonstrate significantly better fidelity for different benchmark circuits, compared to the scenario where the uncut circuit was executed on the least noisy device. Moreover, if the number of subcircuits is not more than the number of hardware, then, when some restrictions on the maximum execution time for each hardware are maintained, we provide a polynomial time graph theoretical scheduler which provides results equivalent to the ILP scheduler. Our method represents an initial step towards noise and time-minimized distributed quantum computing, showcasing promising outcomes in improving the performance of quantum computing in real-world applications. For this study, we have not considered the queuing delay of the hardware since there is currently no known relationship between queue time and hardware noise.

3.2 Hardware Schedule for Subcircuits

In this work, we study the problem of scheduling jobs on different hardware with a focus on maximizing the fidelity and minimizing the execution time. First, we want

to emphasize that in this work we consider circuit-cutting primarily as a method to suppress the effect of noise. It has been shown in multiple studies that circuit cutting itself can lower noise in the system [BS+22, ST+21, TAR+21, MW22]. Therefore, we shall resolve to circuit cutting even if the circuit is small enough to be executed on the hardware. This method allows us to improve fidelity as well as use parallel scheduling of the subcircuits obtained after cutting to multiple hardware, thus lowering the execution time [CD+22].

Consider a list of hardware H and a list of circuits (or subcircuits) C. For a (sub) circuit $i \in C$, let l_{ij} be the optimum (least noisy) layout on a hardware $j \in H$. Ideally, each (sub) circuit can be assigned to the best layout corresponding to it, in terms of noise. However, cutting the circuit increases the number of executable circuits by creating multiple instances for each subcircuit (refer to Fig. 2.1). Therefore, the trade-off for error suppression using circuit cutting is the increased execution time to execute all the subcircuit instances.

If the number of hardware available is at least as many as the number of subcircuit instances, then a polynomial time algorithm for finding a minimum weight maximum matching in the bipartite graph having an edge between a subcircuit and a hardware with a weight (say, the *mapomatic* score) can provide the required assignment. This also comes with an inherent assumption that the maximum execution time for each hardware can accommodate no more than one subcircuit instance. However, if the number of subcircuit instances are more than that of available hardware, or the allowed execution time for each hardware can accommodate more than one subcircuit instance, then job scheduling has to be performed.

Of these multiple subcircuits, most of them may conform to the least noisy layout on the same hardware. Therefore, the best assignment may lead to *sequential* execution of a large number of subcircuits, leading to a large execution time. A user often has limitations on the execution time on a particular hardware, barring this sequential approach. On the other hand, the execution time can be minimized if we opt for as much parallelization as possible, i.e., equally distribute the (sub) circuits to all the available hardware without considering the noise profile.

In this study, we delve into finding the optimum scheduling of the (sub) circuits on the available hardware, when an upper limit on the execution time for each hardware is imposed, such that the overall fidelity is maximized. The problem statement can be formally stated as follows: Given a list of circuits C, a list of hardware H and the corresponding execution time limit τ_j for $j \in H$, find an assignment $X_{ij}\ \forall\ i \in C$ such that the fidelity of the circuits are maximized and the execution time $t_j \leq \tau_j\ \forall\ j \in H$.

This problem, in general, is not known to have a polynomial time solution. However, if (i) the number of subcircuits is at most as many as the number of hardware, and (ii) the maximum allowable time τ_j for each $j \in H$ can accommodate only one subcircuit. We first propose an integer linear programming (ILP) approach to the scheduling problem in general in Sec. 3.3, elaborated in Fig. 3.1. Later, in Sec. 3.5 we show a polynomial time solution to this problem under the above mentioned restrictions.

3.3 Proposed framework

We start with the premise where a list of circuits C and a list of hardware H are provided. The list of hardware can either be provided by the user or can be determined from their credential for a particular vendor. For each $c \in C$, we first fragment it into k subcircuits via circuit cutting. Note that as stated before, we resort to cutting all the circuits, irrespective of whether these can be executed on a single hardware, in order to reduce the noise, and thus improve the fidelity. Henceforth, C denotes the set of all subcircuits obtained via circuit-cutting, and $i \in C$ implies a subcircuit.

The steps in the flowchart of Fig. 3.1 are described in the following subsections.

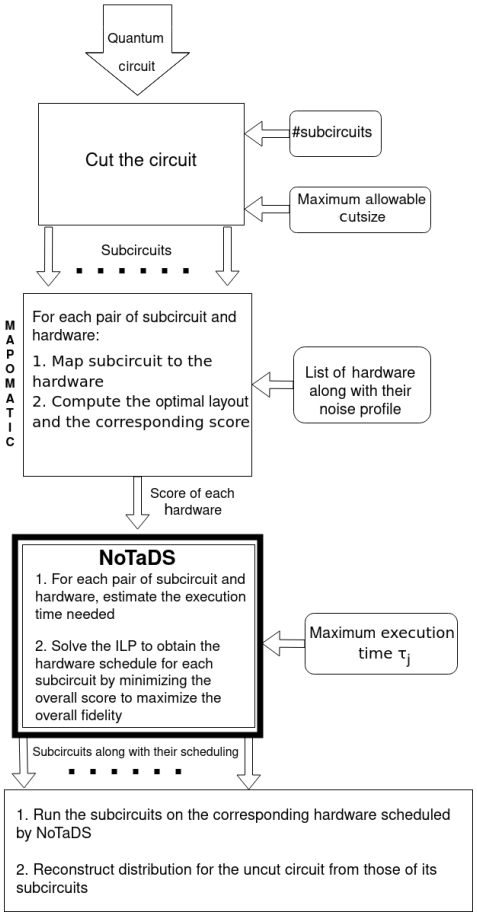


Figure 3.1: A flowchart of our proposed noise and time optimized scheduler, including circuit cutting, scoring of (circuit, hardware) pair, noise and time optimized scheduling, and final reconstruction of the entire probability distribution from those of the subcircuits.

3.3.1 Selection of appropriate hardware

As stated before, let C be the set of all subcircuits. Naturally, tagging is required to keep track of which subcircuit corresponds to which circuit for classical recombination over the cuts which follows later. First, for each subcircuit $i \in C$, the set of hardware $H_i \in H$ is determined such that for all $j \in H_i$ the number of qubits in j is at least as big as the number of qubits in the subcircuit i. If $H_i = \{\}$ for any subcircuit i, then i needs to be partitioned again such that at least one hardware

can accommodate each subcircuit.

At the end of this step, we obtain a list of feasible hardware H_i for each subcircuit i.

3.3.2 Scoring each hardware as per noise profile

Next, we use mapomatic [T+22] for each $i \in C$ and $j \in H_i$. The action of mapomatic here can be considered as a function

$$\mathcal{F} : \{j, i\} \rightarrow \{l_{ij}, Q_{ij}\}$$

where l_{ij} is the optimum layout and Q_{ij} is the mapomatic score for this set of circuits and layout. In other words, given a hardware j and a circuit i, mapomatic returns a list of physical qubits l, which is the initial layout for the placement of i on j, and a score Q_{ij}, which is an indicator of the noise. For each (subcircuit-hardware) pair (i, j), we obtain a set of such score Q_{ij}. Therefore, for each circuit i, this step produces a list of hardware $j \in H_i$ ordered by the score Q_{ij}. In the usual scoring technique of mapomatic, a lower score implies a lower noise profile. Therefore, a hardware h_1 is better than h_2 for a circuit i if $Q_{ih_1} < Q_{ih_2}$. However, one may define their own custom scoring technique which may imply the opposite. At the end of this step, we obtain an ordering among the list of feasible hardware for each subcircuit in terms of their noise profile.

3.3.3 Noise and Time Aware Distributed Scheduler (NoTaDS)

Now, we propose an integer linear program (ILP) to schedule each subcircuit from C to quantum hardware such that the fidelity is maximized while conforming to the upper bound on the execution time for each hardware. Note that the list of

feasible hardware and their ordering as per mapomatic score may vary from circuit to circuit. Therefore, the optimization needs to take into account this variation for each subcircuit.

After cutting, each subcircuit corresponds to multiple subcircuit instances. The number of instances of a subcircuit i depends on the number of preparation qubits ρ_i and the number of measurement qubits O_i. We associate a value η_i with each subcircuit i such that

$$\eta_i = \begin{cases} 1 & \text{if all instances are scheduled individually} \\ \nu(\rho_i, O_i) & \text{otherwise} \end{cases}$$

where $\nu(\rho_i, O_i)$ denotes the total number of subcircuit instances for subcircuit i. In Sec. 3.4 we discuss their advantages and disadvantages.

Next, we define the variables, constraints, and the objective function for the ILP.

1. Variables: We associate a variable X_{ij} for each subcircuit $i \in C$ and hardware $j \in H_i$ such that

$$X_{ij} = \begin{cases} 1 & \text{if subcircuit } i \text{ is scheduled to hardware } j \\ 0 & \text{otherwise.} \end{cases}$$

   In other words, X_{ij} acts as a decision variable for the scheduling. Moreover, a score variable Q_{ij} is associated with each X_{ij} which indicates the mapomatic score when subcircuit i is placed on hardware j.

2. Constraints: Next, we fix the constraints for the ILP.

   (a) The first requirement is that every subcircuit i is assigned to some hardware. Formally, this constraint can be represented as

$$\sum_{j \in H_i} X_{ij} = 1. \tag{3.1}$$

Note that this constraint should hold for all subcircuits $i \in C$, and therefore, Constraint 4.2 essentially results in $|C|$ constraints.

(b) Now, as discussed before, there is some time restriction for each hardware for the user. We associate a maximum execution time τ_j for each $j \in H$. The value of τ_j can be provided by the user or can be determined from the user's access plan. Let t_i denote the execution time for each subcircuit $i \in C$. Then, the total execution time of all the subcircuits scheduled to a particular hardware j should not exceed τ_j. Formally, this is represented as

$$\sum_{i \in C} \eta_i \cdot t_i \cdot X_{ij} \leq \tau_j. \tag{3.2}$$

Note that while the summation of this constraint goes over the entire set of subcircuits, the indicator variable X_{ij} ensures that the time for a particular subcircuit is added to the execution time only if it is scheduled to that hardware. This constraint holds for each hardware, and therefore Constraint 4.1 essentially results in $|H|$ constraints.

3. Objective Function: The objective of this optimization problem is to maximize the overall fidelity, which translates to minimizing the overall score Q. Therefore, the objective function for this is defined as

$$\min \sum_{i \in C, j \in H} X_{ij} \cdot Q_{ij} \cdot A_i \tag{3.3}$$

Here, $0 < A_i \leq 1$ is the area of circuit i normalized over all available circuits. In this study, we define the area of a circuit as the product of the number of qubits and the 2-qubit depth. In other words, higher the area of a circuit, more amenable it is to noise. The inclusion of this term thus ensures that circuits with a higher area are placed in layouts with lower mapomatic scores to ensure the quality of the outcome. For example, Fig. 3.2 shows a possible cutting of a 5-qubit circuit into two subcircuits. Both the subcircuits consist of 3 qubits, but the 2-qubit depth of the first subcircuit is 7 whereas that of the second subcircuit is 2. The first subcircuit which has more depth is more prone to noise, and hence should be placed in a better layout.

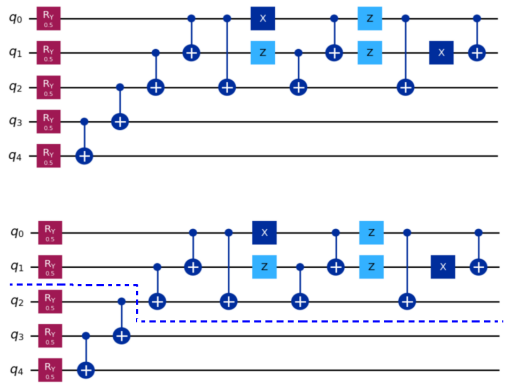


Figure 3.2: Cutting with unequal depth : to understand why area is an important element in the objective function

The final ILP formulation, thus, is

$$\begin{aligned} \min \quad & \sum_{i \in C, j \in H} X_{ij} \cdot Q_{ij} \cdot A_i \\ \text{subject to} \quad & \text{Constraints 4.2-4.1} \\ & X_{ij} \in \{0, 1\}. \end{aligned}$$

Note on linear objective function The objective function of Eq. 3.3 is linear in Q_{ij}. A question may arise whether it is sufficient to have a linear objective function when minimizing over multiple subcircuits and their multiple instances. As we mentioned earlier, the score computed by *mapomatic* is an indicator of the hardware noise profile. However, the quality of the result for a shallow subcircuit running on a noisy layout may exceed that of a very deep circuit running on a less noisy layout. As long as the subcircuits are roughly equal in the number of qubits and the depth, the ordering of the hardware layouts according to its *mapomatic* score primarily depends on the hardware noise profile. In such a scenario, a linear objective function that minimizes the overall score over all the subcircuits is sufficient. However, if the subcircuits are largely imbalanced in width and/or depth, then both the size of the circuit and the noise profile of the layout affect the *mapomatic* score. In that case, a non-linear objective function may be needed to ensure good scheduling for *all* the subcircuits. However, previous results on cir-

cuit cutting show optimal performance when the cutting is more or less balanced [BS+22]. Therefore, in this study, we primarily stick to cuts that lead to roughly balanced subcircuits, hence linear objective function suffices for the main goal.

3.4 Experimental Results

In this section, we show the experimental results of our NoTaDS scheduler for different types of circuits. We have used the Circuit-knitting-toolbox [LA+23] for circuit cutting and reconstruction, and CPLEX Optimization Studio [cpl] to solve the ILP from Sec. 3.3. Table 3.1 shows the set of real IBM quantum hardware used for our experiments and their noise profile. Some of the parameters for the noise profile include the probability of faulty gates and measurement, and the rate of spontaneous decay of a qubit, characterized by T_1 and T_2. The noise profile of the hardware varies with time. The values for each type of error in the table are the average over all the qubits in that hardware. Moreover, the readout error probability for each qubit is the average of $p(0|1)$ and $p(1|0)$ where $p(i|j)$ denotes the probability of measuring i when the outcome was originally j, $i, j \in \{0, 1\}$.

Table 3.1: Number of qubits and noise profile of the hardware

Name of real IBM hardware	# Qubits	2-qubit gate error probability	1-qubit gate error probability	T_1 (μs)	T_2 (μs)	Readout error probability
IBMQ Hanoi	27	8.3×10^{-3}	2.1×10^{-4}	156.69	137.7	10^{-2}
IBMQ Mumbai	27	7.5×10^{-3}	2.5×10^{-4}	118.01	161.97	1.8×10^{-2}
IBMQ Cairo	27	9.4×10^{-3}	2.2×10^{-4}	94.62	116.42	1.3×10^{-2}
IBMQ Kolkata	27	8.7×10^{-3}	2×10^{-4}	117.42	92.97	1.2×10^{-2}
IBMQ Guadalupe	16	9.74×10^{-3}	2.64×10^{-4}	86.72	118.73	1.64×10^{-2}
IBMQ Lagos	7	7.2×10^{-3}	2×10^{-4}	112.51	84.42	1.4×10^{-2}
IBMQ Nairobi	7	8.7×10^{-3}	3.5×10^{-4}	114.75	71.42	2.7×10^{-2}
IBMQ Jakarta	7	7.3×10^{-3}	1.03×10^{-4}	136.95	38.99	2.09×10^{-2}
IBMQ Manila	5	7.7×10^{-3}	2.46×10^{-4}	141.15	56.53	2.2×10^{-2}
IBMQ Lima	5	9.58×10^{-3}	3.76×10^{-4}	98.68	115.32	2.41×10^{-2}
IBMQ Belem	5	8.89×10^{-3}	3.88×10^{-4}	101.42	98.85	2.39×10^{-2}
IBMQ Quito	5	7.9×10^{-3}	2.88×10^{-4}	96.83	104.39	4.15×10^{-2}

In the following subsections, first, we discuss the selection of the maximum execution time τ for each hardware, and then we discuss our method for estimating the execution time of a subcircuit. Finally show the fidelity obtained by our scheduling method for a range of quantum circuits.

3.4.1 Criteria for maximum execution time τ

In Sec. 2, we defined the maximum execution time for a hardware $j \in H$ as τ_j. There are η_i instances for each subcircuit $i \in C$. Let $t^{(i)}$ be the execution time for subcircuit i. The maximum execution time required for a particular hardware, τ_{max}, is when all the instances of all the subcircuits are executed sequentially on one particular hardware. Then

$$\tau_{max} = \sum_{i \in C} \eta_i \cdot t^{(i)}.$$

Note that allowing any excess time to τ_{max} does not change the scheduling and execution time. Therefore, we stick to equality instead of $\geq$.

The minimum time τ_{min} that each hardware should allow should be such that at least one subcircuit can be executed. If there are any hardware which does not conform to this requirement, that one can simply be removed from the list of all available hardware. Here, we want to mention once more that one subcircuit $i \in C$ consists of η_i subcircuit *instances*. One can choose to schedule each instance or each subcircuit. We tested the former, which resulted in a drop of fidelity by $\sim 9\%$ over the latter. This is obvious since the different instances are equivalent to a tomography of the subcircuit [PS+21, MW22]. Therefore, running each instance of the subcircuit on different hardware (i.e., different noise models) leads to an inaccurate tomography, which makes the reconstruction fallible. Therefore, for this study, we stick to the scheduling of subcircuits, and not the instances. There may be scenarios where scheduling the instances in an intelligent way may lead to a lower decrease in the fidelity – we postpone that for future studies.

For our experimental settings, the number of hardware is always chosen to be greater than the number of subcircuits. Therefore,

$$\tau_{min} = \max_{i \in C} \eta_i \cdot t^{(i)}.$$

In all our experiments, we fix $\tau_j = \tau_{min}\ \forall\ j \in H$. Later in Sec. 3.4.7 we show the change in fidelity and execution time if we allow $\tau_j > \tau_{min}$.

Calculating the values of τ_{max} and τ_{min} require the execution time $t^{(i)}$ for all the subcircuits $i \in C$. In the following subsection, we discuss the method we used to fix the value of $t^{(i)}\ \forall\ i \in C$ in our experiment.

3.4.2 Estimation of the execution time of a circuit

There are sophisticated methods for estimating the run-time of a quantum circuit [JG+23]. However, for our experiment, we stick to a simple method of calculating the time of each level of a circuit. We define *level* of a circuit as the timestamp where some gates are executed in parallel. In Fig. 3.3 we separate the different levels of the circuit by red lines.

The time duration of each level is determined by the longest gate in that level. Naturally, 2-qubit gates have a much larger execution time than 1-qubit gates. Therefore, if a level contains a 2-qubit gate, then the time duration of that level is t_2, which is the execution time of a single 2-qubit gate. Note that it doesn't matter if the level contains multiple 2-qubit gates since they are operated parallelly. On the other hand, if a level contains only single qubit gates then the time duration of that level is t_1, which is the execution time of a single 1-qubit gate. Therefore, if a circuit contains κ_1 levels where only 1-qubit gates are present and κ_2 levels where 2-qubit gates are also present, then the overall runtime is $\kappa_1 \cdot t_1 + \kappa_2 \cdot t_2$. In Fig. 3.3, in the whole circuit $\kappa_1 = 2$ and $\kappa_2 = 5$. In the first subcircuit, $\kappa_1 = 2$ and $\kappa_2 = 2$, and in the second subcircuit $\kappa_1 = 2$ and $\kappa_2 = 3$.

In current IBM Quantum devices, the execution time of a CNOT gate is $\sim 10\times$ that of single-qubit gates. For this study, we assume $t_1 = 1$, making $t_2 = 10$. From the abstraction, the execution time of the circuit in Fig. 3.3 is $2 \cdot t_1 + 5 \cdot t_2$. This abstract calculation of the execution time keeps the method simple. Since the values of τ_{min} and τ_{max} depend on the execution time, if some other method for

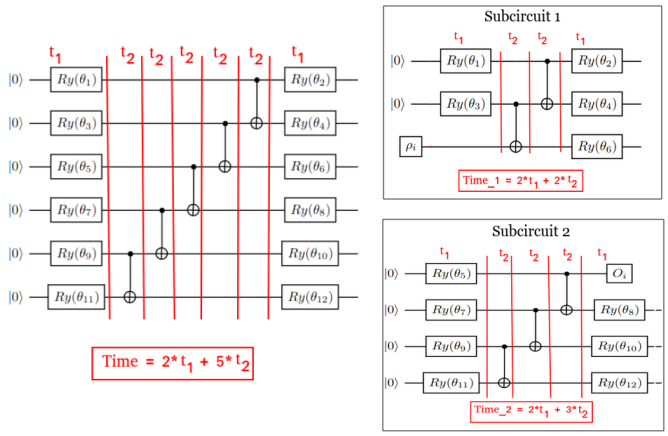


Figure 3.3: An example of a 6-qubit RealAmplitudes circuit with the level.

determining the execution time is used, or if absolute execution times are selected, then the values of τ will change accordingly without hampering the assignment of the subcircuits on the hardware. Note that if absolute values are used instead of t_1, t_2, then one should also account for the fact that the absolute values for the execution time of gates are not always the same on different hardware.

3.4.3 Results for 6-qubit circuits

In Table 3.2 we consider four benchmark circuits having 6 qubits each. These circuits are small enough to be executed on any hardware with ≥ 7 qubits, and hence distributed scheduling using circuit cutting may not be deemed necessary here. However, Table 3.2 shows that distributed scheduling using circuit cutting still helps in the improvement of fidelity. For each of the circuits, we provide its fidelity with the ideal simulation both without any error mitigation and with measurement error mitigation (MEM). For MEM, we have used the default *MThree* mitigation [NK+21] provided in *Qiskit Runtime* by setting the *resilience level* option to 1.

Naturally, MEM improves the fidelity over no-mitigation. However, we observe that distributed scheduling with circuit cutting without any error mitigation outperforms the fidelity of the uncut circuit with MEM. In this experiment, we par-

Table 3.2: Fidelity for 6-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM).

Benchmark circuit	# qubits	Cut size	# subcircuits	Fidelity			
				Uncut		Cut	
				NoMit	MEM	NoMit	MEM
Ripple carry adder [SC+04]	6	2	2	0.759	0.787	0.792	0.843
RealAmplitudes [KM+17]	6	1	2	0.959	0.983	0.987	0.997
Trotterized [MW22]	6	2	2	0.922	0.951	0.965	0.974
Bernstein Vazirani [AA+19]	6	1	2	0.81	0.869	0.882	0.944

titioned the circuit into two subcircuits. We want to emphasize here that (i) as discussed before, τ for all hardware was fixed to τ_{min}, and (ii) the uncut circuit was always executed on the best hardware and its corresponding layout as per mapomatic. We obtain an average percentage improvement in fidelity for distributed scheduling using circuit cutting over no cutting by ~ 5.2 when no mitigation was used, and by ~ 4.89 when MEM was used. The average is taken over the four circuits in Table 3.2.

Next, we dive deeper into the exact details of the experiment for the 6-qubit Ripple carry adder circuit [SC+04]. This is meant to provide an overall idea for recreating the experimental steps for the circuit in Table 3.2 and also those in later subsections. The steps follow from the flowchart provided in Fig. 3.1.

Experiment details for the 6-qubit Ripple carry adder circuit

Fig. 3.4 shows the circuit for the 6-qubit ripple carry adder and its two subcircuits obtained using the Circuit-knitting-toolbox [LA+23]. After obtaining the subcircuits, we found the hardware big enough to accommodate each of them. In this particular scenario, all the hardware from Table 3.1 can accommodate each subcircuit.

Next, we use mapomatic to find the score for each subcircuit against each hardware and its layout. Then we use the optimization in Sec. 3.3 to schedule the subcircuits to the hardware. Finally, we use mapomatic to find the best hardware and its

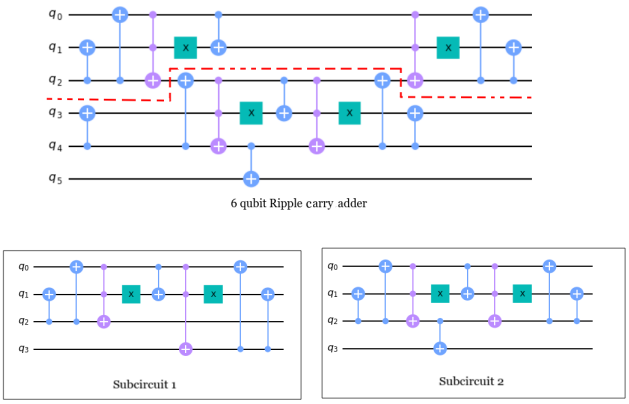


Figure 3.4: Two subcircuits of 6-qubit Ripple carry adder circuit obtained by using Circuit-knitting-toolbox [LA+23].

layout for the uncut circuit as well. In Table 3.3 we show the layout, backend, and the mapomatic score for the uncut circuit and the two subcircuits.

Table 3.3: Scheduling details of the 6-qubit ripple carry adder circuit and its two subcircuits obtained after cutting.

Circuit	# qubits	Layout	Scheduled backend	Mapomatic score
Uncut	6	[26, 25, 24, 23, 21, 18]	IBMQ Kolkata	0.28
Subcircuit 1	4	[26, 25, 22, 19]	IBMQ Mumbai	0.13
Subcircuit 2	4	[4, 1, 2, 3]	IBMQ Hanoi	0.11

Note that the cut-size to partition the 6-qubit adder circuit into two subcircuits is 2. Therefore, each subcircuit has 4 qubits. We note from Table 3.3 that our scheduler has scheduled the two subcircuits on two different hardware, each of which has a mapomatic score lower than the one where the original circuit has been scheduled. This provides an explanation as to why the fidelity obtained via cutting exceeds the uncut circuit.

3.4.4 Results for 10-qubit circuits

Next, in Table 3.4 we take a few 10-qubit circuits. These circuits are too big to be executed on 5 or 7-qubit devices but can be executed without cutting on 16 or 27-qubit devices. However, as before, we show that the fidelity can be improved by

using our NoTaDS scheduler. Table 3.4 shows the fidelity of four 10-qubit circuits with and without measurement error mitigation, where the value of τ is set to τ_{min} for all hardware.

Table 3.4: Fidelity for 10-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM).

Benchmark circuit	# qubits	Cut size	# subcircuits	Fidelity			
				Uncut		Cut	
				NoMit	MEM	NoMit	MEM
Ripple carry adder [SC+04]	10	2	2	0.315	0.325	0.375	0.5138
Bernstein Vazirani [AA+19]	10	1	2	0.702	0.714	0.728	0.749
RealAmplitudes [KM+17]	10	1	2	0.806	0.876	0.977	0.994
Trotterized [MW22]	10	2	2	0.878	0.891	0.927	0.960

Once more we observe that the fidelity of the noisy and the ideal circuit obtained without any error mitigation via our scheduling method outperforms (sometimes significantly, e.g., see RealAmplitudes and Trotterized circuits) the fidelity of the uncut circuit with MEM. We obtain an average improvement in fidelity for distributed scheduling using circuit cutting over no cutting by $\sim 12.38\%$ when no mitigation was applied, and by $\sim 21\%$ when MEM was used. The average is taken over the four circuits in Table 3.4.

In the following subsection, we take a deeper dive into the improvement in fidelity with the variation in the number and size of the subcircuits.

3.4.5 Variation in fidelity with the number and size of subcircuits

In Fig. 3.5 we plot the fidelity of 20-qubit RealAmplitudes and Bernstein-Vazirani circuits as the number of subcircuits is increased linearly from 2 to 6. In each case, each subcircuit is scheduled using the NoTaDS scheduler, and the fidelity is compared with the ideal outcome. We notice that the fidelity increases linearly with an increasing number of subcircuits.

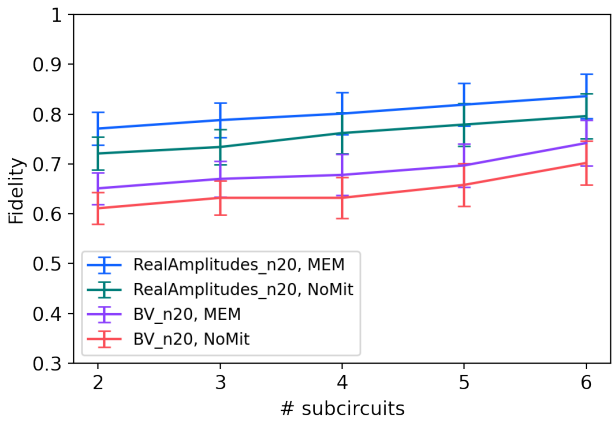


Figure 3.5: Fidelity obtained by the *NoTaDS* scheduler with an increasing number of subcircuits for 20-qubit RealAmplitudes and Bernstein Vazirani (BV) circuits.

Here we consider cutting 20-qubit RealAmplitudes and Bernestein-Vazirani circuits, where the number of subcircuits varies from 2 to 6. The subcircuits are then scheduled with our proposed NoTaDS scheduler. The result is bootstrapped over 10 trials. We notice a linear improvement in fidelity with the increase in the number of subcircuits. As the number of subcircuits increases, each subcircuit becomes smaller, and hence less contagious noise. Therefore, the fidelity is increased. However, with an increase in the number of subcircuits, the cut-size also increases leading to an exponential increment in the classical postprocessing time for reconstruction of the full probability distribution from the subcircuits [PH+20, TT+21]. We verify this in Fig. 3.6. Therefore, the number of cuts cannot be increased beyond a certain point to keep the classical postprocessing time in check.

We show a complimentary result in Fig. 3.7 where we increase the size of the circuit and partition each of them into two subcircuits. The two subcircuits were then scheduled by our proposed NoTaDS scheduler. We notice that the fidelity decreases with an increase in the size of the circuit. The result is bootstrapped over 10 trials.

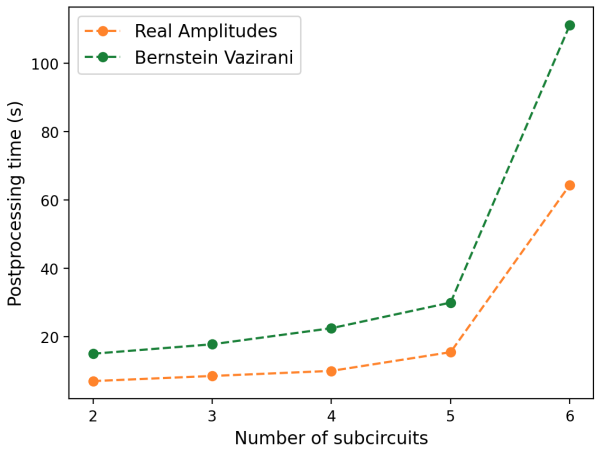


Figure 3.6: The increase in classical reconstruction time of the full probability distribution from the subcircuits with increasing number of subcircuits.

3.4.6 Results for a 28-qubit circuit

In our chosen set of hardware (Table 3.1), the largest hardware contains 27 qubits. Hence, here we consider one circuit that is too big to be executed on any of the hardware. However, via circuit cutting and NoTaDS scheduling, we can still execute such a circuit. In Table 3.5 we show the fidelity obtained with and without MEM for a 28-qubit RealAmplitudes circuit. We do not have any fidelity value for uncut since it is too big to be executed on our set of devices. We observe that the fidelity is poor without error mitigation, but improves significantly in the presence of MEM. This is obvious since the measurement is the most dominant noise in current quantum devices (see Table 3.1 for the probabilities of different types of noise). Therefore, the larger the circuit, the stronger the effect of measurement error, leading to poor fidelity.

We have selected hardware devices up to 27-qubit devices for our experiments. Currently, IBM has hardware with 433 qubits, and our proposed method is independent of the size of the hardware.

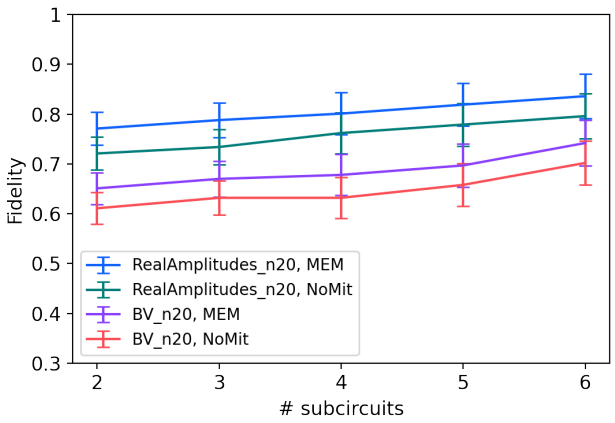


Figure 3.7: Fidelity obtained by the NoTaDS scheduler with increasing size of the circuit where each circuit is partitioned into two subcircuits.

Table 3.5: Fidelity for 28-qubit circuits by scheduling over the hardware in Table 3.1 with and without circuit cutting for no error mitigation (NoMit) and measurement error mitigation (MEM).

Benchmark circuit	# qubits	Cut size	# subcircuits	Fidelity			
				Uncut		Cut	
				NoMit	MEM	NoMit	MEM
RealAmplitudes	28	1	2	-	-	0.31	0.7

3.4.7 Change in fidelity with and without scheduling

As stated before, till now in all our experiments we have fixed $\tau_j = \tau_{min}\ \forall\ j \in H$. Naturally, this makes the scheduling restrictive. It may be possible to execute all the subcircuits on the best device to obtain the best fidelity at the cost of execution time. Our restriction over the maximum allowable execution time τ prevented NoTaDS from doing so.

In Fig. 3.8 we consider a 16-qubit RealAmplitudes circuit which is partitioned into two balanced subcircuits (i.e., the number of qubits and gate count are roughly equal for both). We show the fidelity obtained when the two subcircuits are executed in all possible hardware pairs (j, k), $j, k \in H$. Note that since there are only two subcircuits,

$$execution\ time = \begin{cases} \tau_{max} & \text{for } j = k \\ \tau_{min} & \text{otherwise.} \end{cases}$$

The partition being balanced, we have $\tau_{max} \simeq 2 \cdot \tau_{min}$.

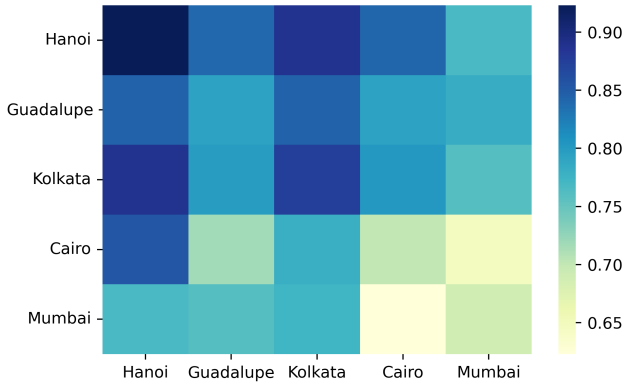


Figure 3.8: Fidelity of a 16-qubit RealAmplitudes circuit, when partitioned into 2 subcircuits, and executed on all possible hardware pair. Since there are only two subcircuits, when the subcircuits are scheduled to two different hardware, the execution time is τ_{min}, and when scheduled to the same hardware the execution time is τ_{max}.

The maximum fidelity obtained in Fig. 3.8 is when both the subcircuits are executed on *ibm_hanoi*, whereas if the two subcircuits are executed on *ibm_hanoi* and *ibmq_kolkata*, the fidelity is slightly lower but the execution time is reduced to half. The reduction in fidelity is $\sim 1\%$ only.

On the other hand, the result here indicates that if it is not possible to execute both the circuits on *ibm_hanoi* due to restrictions in execution time, it is rather more useful to schedule the subcircuits to two different hardware using NoTaDS than to execute both of them together on any other hardware. This holds true even if there is some hardware whose maximum execution time can accommodate both subcircuits. For example, if we keep *ibm_hanoi* out of the story, it is better to distribute the two subcircuits to, say *ibmq_kolkata* and *ibm_cairo* than to execute

both of them on the later, even if it can accommodate both. This is because *ibmq_kolkata* has a lower noise profile than *ibm_cairo*. Therefore, NoTaDS will find this distributed scheduling, and improve the final fidelity of the circuit.

In Fig. 3.8, the number of subcircuits is 2, so for τ_{max} both of them can be executed on the best hardware and for τ_{min} distinct devices need to be assigned. In this scenario, allowing an execution time of $\tau_{min} < \tau < \tau_{max}$ to one or more hardware cannot change the scheduling, and hence the fidelity. However, if the number of subcircuits is more than 2, then $NoTaDS$ may be able to find even better schedules for maximum execution time $\tau_{min} < \tau < \tau_{max}$ so that the difference in fidelity obtained from the scheduling with that when all the subcircuits are executed on the best device is less than even 1%.

Naturally answers to the questions such as (i) what is the best schedule, (ii) is it better to schedule all the subcircuits to the same hardware – changes with time (since noise varies with time), the list of available hardware, and the circuits. NoTaDS automates this process by finding the optimum scheduling based on the hardware noise profile and the upper bound on the execution time of each hardware.

To summarize, we make the subcircuits smaller by performing cutting. This implies that each subcircuit experiences lower noise compared to the original circuit. If we run these smaller subcircuits on the two best devices, we are likely to achieve the best possible result as shown by the heatmap, but it will take more time. The point is that by distributing the subcircuits, we are reducing execution time while choosing the two least noisy devices and showing that, in less time, we are achieving results better than the uncut circuit (possibly not as good as all the cut subcircuits running on the best device).

3.5 A polynomial time solution for a restricted scenario

In Sec 3.3 we provided an ILP solution for the scheduling problem. However, ILP is an NP-Hard problem. Therefore, in this section, we propose a graph theoretic approach for the scheduling when (i) the number of subcircuits is at most as many as the number of hardware, and (ii) the maximum execution time $\tau_j = \tau_{min}$, $\forall$ $j \in H$. Under these two restrictions, the scheduling problem essentially becomes an assignment and therefore can be solved in polynomial time using a graph theoretic approach. To differentiate this restricted scenario from the general scheduling problem, we shall call this *circuit assignment*.

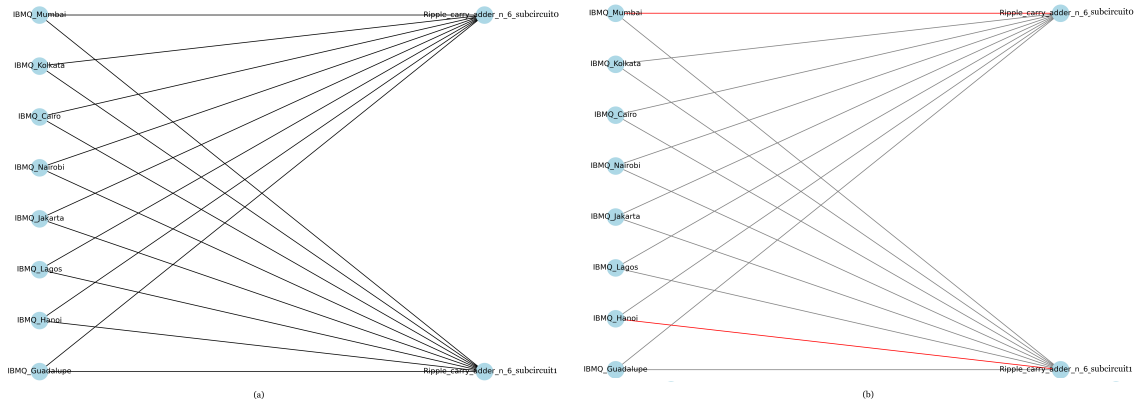


Figure 3.9: For mapping the 6-qubit ripple carry adder to the available hardware : (a) The bipartite graph where the first (left) set of vertices denotes the hardware, and the second (right) set of vertices denotes the two subcircuits of a 6-qubits ripple carry adder. (b) A Minimum Weight Maximum Matching based solution to the assignment of two subcircuits corresponding to the 6-qubit ripple carry adder.

Let, C and H be the set containing the subcircuits and hardware. We convert this *circuit assignment* problem into a complete bipartite graph $G = (C, H, E)$ such that for every two vertices $c \in C$ and $h \in H$, $e = (c, h) \in E$. Such a complete bipartite graph with partitions of size $|C| = m$ and $|H| = n$ is denoted by $K_{m,n}$. As discussed in Sec 2.3, we shall find the best layout and the *mapomatic* score $Q_{c,h}$

for each (circuit, hardware) pair (c, h). We assign a weight of $Q_{c,h}$ to each edge $e = (c, h) \in E$.

In Fig. 3.9 (a) we show an example of mapping the assignment problem to a complete bipartite graph, where the first (left) set of vertices denotes the hardware, and the second (right) set of vertices denotes the two subcircuits of a 6-qubits ripple carry adder. Each edge is associated with the *mapomatic* score for the corresponding (circuit-hardware) pair. However, we have not shown the weights of the edges in the figure to keep it tidy.

In a bipartite graph, matching refers to a collection of edges selected in a manner where none of the chosen edges have a common endpoint. A maximum matching, denotes a matching that has the largest possible size, indicating the highest number of edges that can be included. It becomes a perfect matching when $|C|$ and $|H|$ are equal which is not necessarily true for our case. In our problem, we want to assign each subcircuit to hardware such that the mapomatic score is minimized. Therefore, finding the optimal noise-aware assignment of the subcircuits to the hardware is the same as finding the minimum weight maximum matching (MWMM) [Edm65b].

Fig. 3.9 (b) shows the assignment of the two subcircuits to the hardware, denoted by the red edges. Note that, the ILP solution for the scheduling for the 6-qubit ripple carry adder, shown in Table 3.3, is also the same. Therefore, the fidelity obtained by MWMM is the same as that of the ILP scheduler.

Note that if the number of subcircuits is more than the number of hardware, or $\tau_j > \tau_{min}$ for $j \in H$, then more than one subcircuit may be scheduled to the same hardware. This is no longer a matching, and hence cannot be solved using the MWMM method. Therefore, this polynomial time graph theoretical approach is applicable only to a restricted scenario, as discussed at the beginning of this section.

3.6 Summary

In this work, we propose a noise and time optimized distributed scheduler that schedules the subcircuits obtained after circuit cutting to hardware such that the fidelity is maximized, and yet the execution time on each hardware is restricted by a pre-specified limit.

Note that this same scheduler can be used to schedule a set of circuits to hardware even without circuit cutting. However, we show that our method outperforms the fidelity of the uncut circuit which has been executed on the least noisy device, and yet requires significantly low execution time on a quantum processor. This method combines inter-device parallelization with noise-aware scheduling to optimize the fidelity of the circuit. This method is expected to be particularly useful in the near-term when the devices are noisy, and the execution time available to a user on a quantum device is limited. The study of scheduling circuits where balanced partitioning may be too costly may be explored in the future.

With a focus on distributed scheduling, we proceed to examine resource-aware scheduling methods for optimizing quantum circuit execution on hardware devices in the next chapter.

Code availability

The code to find the optimal scheduling using our proposed NoTaDS scheduler is available in https://github.com/debasmita2102/NoTODS.

CHAPTER 4

Resource-aware scheduling of multiple quantum circuits on a hardware device

Contents

4.1 Introduction 73

4.2 Formulation of intra-device scheduling as an optimization problem 75

4.2.1 Finding the overlap between two layouts 76

4.2.2 ILP Formulation for our scheduling problem 79

4.3 Polynomial time heuristic algorithm 83

4.3.1 Generation of the compatibility graph 84

4.3.2 Greedy algorithm to find a *maximal* clique in the compatibility graph 87

4.4 Experimental results 89

4.4.1 Selection of ϵ for the layouts 90

4.4.2 Fidelity and hardware utilization in intra-device scheduling 91

4.5 Conclusion 92

4.1 Introduction

Currently multiple providers offer free or paid access to their quantum hardware devices. The majority of these access are via cloud. As the demand for quantum computing is rising, the users often face long queuing time with their jobs having to wait till the execution of all the previously submitted jobs have been completed. Consequently, there is a pressing need to enhance the efficiency and throughput of quantum computers to improve user experience. In this paper, we study the benefits and challenges of executing more than one quantum circuit simultaneously on the same hardware to improve the throughput, as well as the hardware utilization, without sacrificing on the quality of outcome.

Let us motivate the problem with an example. Consider a 15-qubit circuit which is to be executed on a 27 qubit device as shown in Fig 4.1. Thus 12 qubits of the device remain unused, which could have been utilized to execute simultaneously some other circuit(s) requiring ≤ 12 qubits – thus improving the throughput and the hardware utilization.

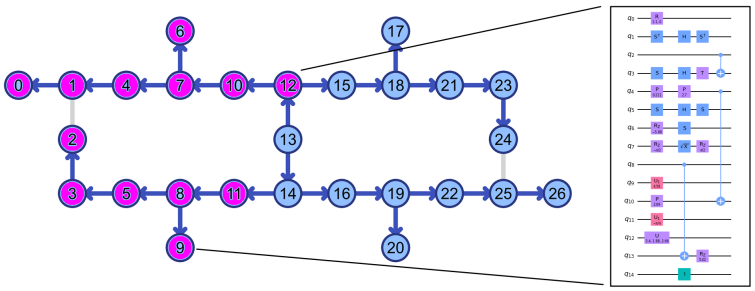


Figure 4.1: An example of a 15-qubit circuit assigned to a 27- qubit hardware. The used qubits are shown in purple while the unused qubits are shown in blue. The hardware still has room to accommodate one or more quantum circuit(s) using the free qubits.

Simultaneous execution of multiple circuits on a single hardware is not without challenge. Previous studies [CBSG17, GP17] do not consider the effect of noise

arising due to simultaneous execution of circuits. First, when a circuit is mapped to a hardware, the requirement is to reduce the number of SWAP gates, as well as to use a layout with minimal noise profile [T+22]. However, when multiple circuits are placed simultaneously, it is likely that all of them cannot be placed on their corresponding best layout, leading to degradation in the quality of the outcome of the computation. Furthermore, if two circuits are computed on neighbouring qubits, then there is a possibility of crosstalk affecting the quality of the computation for both of them.

In [DT+19], the authors take a primarily empirical approach to solving reliability challenges in multi-programming quantum computers. They rely on experimental observations and runtime adaptations through three solutions: Fair and Reliable Partitioning, Delayed Instruction Scheduling (DIS), and Adaptive Multi-Programming (AMP). While achieving a 2x improvement on a 16-qubit system, the approach has several limitations. The partitioning based on noise can lead to sets of qubits in one partition resulting in higher SWAP counts, whereas we avoid such partitioning to ensure optimal SWAP counts for each circuit. Their DIS algorithm, designed for running programs of varying lengths, can be simplified using ALAP (As Late As Possible) scheduling to reduce transpilation overhead. Their AMP technique calculates reliability using entropy functions derived from shared (S) and individual (I) shots, but this undermines the efficiency of multi-programming by requiring circuits to run twice. In contrast, we utilize layout scores to select and combine circuits that fit the top $k\%$ of the layouts. Additionally, the paper does not address the concept of batch execution, which we propose to handle dynamic scenarios while preventing job starvation. Furthermore, their work does not account for critical factors such as qubit quality, coupling map, and layout scores, and it lacks formal mathematical proofs or theoretical foundations.

Minimizing the degradation in quality due to worse layout selection and crosstalk, while maximizing the throughput of the hardware lead to conflicting objectives. In this paper, we study this optimization problem to find the optimal *intra-device scheduling* of $N > 1$ jobs on a m-qubit hardware with little to no compromise on the quality of computation. Given a hardware H, and a set C of N circuits

$\{C_1, C_2, ..., C_N\}$, such that the number of qubits of each circuit is $\leq$ the number of qubits in the hardware, we partition C into batches $B = \{B_1, B_2, ..., B_k\}$ such that each batch consists of circuits which can be executed simultaneously. Therefore, for each i, we must have $1 \leq |B_i| \leq N$ where $|B_i|$ indicates the number of circuits in that batch and the total number of qubits in each B_i is no greater than m. Maximizing intra-device parallelization is thus equivalent to minimizing the number of batches.

4.2 Formulation of intra-device scheduling as an optimization problem

Consider a hardware H with m qubits and a set $C = \{C_1, C_2, ..., C_N\}$ of quantum circuits. The goal of this study is to place batches of $k \leq N$ circuits simultaneously on the hardware such that (i) the total number of qubits for each batch of circuits is no greater than m, (ii) there is no overlap of qubits between different circuits, and (iii) the noise profile of the layout associated with each circuit is within ϵ of the optimal layout for that circuit. In other words, if s_1 be the noise profile of the optimal layout for a circuit, then for that circuit we consider only those layouts l for which $s_l - s_1 < \epsilon$, for a pre-specified ϵ.

Note that the constraint of no overlap between qubits may seem trivial at first. However, it is to be noted that if two neighbouring qubits are associated with different circuits, then there is a possibility of crosstalk affecting the quality of outcome of both the circuits. Therefore, a buffer distance b, i.e., a distance of b, must be maintained between any two qubits associated with two different circuits. In Fig. 4.2, we show examples of two circuits placed with $b = 0$ and $b = 2$ respectively. The former has a significantly higher possibility of crosstalk affecting the quality of outcome [MB+19].

Let l_1 and l_2 be two layouts for two circuits C_1 and C_2. Let the distance between two qubits $q_a \in l_1$ and $q_b \in l_2$ on the same hardware be denoted by $d(q_a, q_b)$.

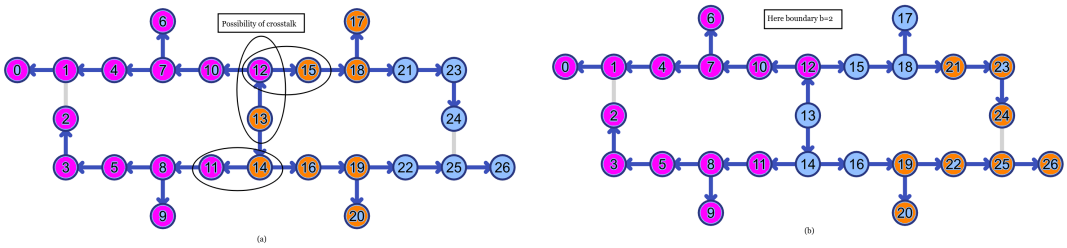


Figure 4.2: Placement of a 15-qubit and a 8-qubit circuit simultaneously on a 27-qubit hardware with buffer distance (a) $b = 0$, and (b) $b = 2$. The former has a significantly higher probability of crosstalk affecting the quality of the computation. The blue qubits are the unused ones.

Henceforth, for a given buffer distance b we say that l_1 and l_2 have b-overlap if $\exists q_a \in l_1$ and $\exists q_b \in l_2$ such that $d(q_a, q_b) < b$, else if $\forall\ q_a \in l_1$ and $q_b \in l_2$, $\min d(q_a, q_b) \geq b$, then l_1 and l_2 are b-non-overlapping. In the next subsection, we propose an efficient algorithm to determine the overlap between two given layouts.

4.2.1 Finding the overlap between two layouts

Given a hardware coupling map, and two layouts l_1 and l_2, the trivial method to check for overlaps is to calculate $d(q_a, q_b)$, $\forall\ q_a \in l_1$ and $q_b \in l_2$. However, if the number of qubits in l_1 and l_2 are n_1 and n_2 respectively, then the time complexity of this method is $\mathcal{O}(n_1.n_2)$. But checking for overlaps only between *boundary qubits* of the two layouts can be done faster.

A qubit in a layout l is said to be a *boundary qubit* if it is adjacent to at least one qubit $q \notin l$. For example, in Fig. 4.2 (b), qubits 11, 12, 21 and 19 are the boundary qubits.

Lemma 4.1

The overlap between the two layouts l_1 and l_2 can be determined by calculating the distance between the boundary qubits of these two layouts only.

Proof. By definition, if $b \in l$ is a boundary qubit in layout l, then $\exists\ q \notin l$ such that $q \in neighbour(b)$, and the overlap between two layouts is the minimum distance between any two qubits from the two layouts. Let $B_l \subseteq l$ be the set of all boundary qubits in layout l. Therefore, a shortest path between some $q_l \in l$, but $\notin B_l$ and $q \notin l$ must contain some $b \in B_l$. Hence, $d(b, q) < d(q_l, q)$. Therefore, the overlap between two layouts can be determined by calculating the distance between the boundary qubits of these two layouts only.

□

In Fig. 4.2 (b), there are 13 and 8 qubits on the two layouts. Therefore, the naive method would have required $13 \cdot 8 = 104$ comparisons to determine the overlap. However, since there are only 2 boundary qubits for each layout, according to Lemma 4.1 only 4 comparisons are sufficient.

Next we provide the algorithms for determining the boundary qubits of a layout, and calculating the overlap between two given layouts.

Algorithm 1 Determine boundary qubits of a layout

Input: Layout L_i and the coupling map of the hardware
Output: A list of the boundary qubits of L_i

```
boundary ← []
for each qubit q ∈ L_i do
    Determine the neighbours Nb of q from the coupling map
    for each qubit q' ∈ Nb do
        if q' ∉ L_i then
            Add q' to boundary
        end if
    end for
end for
return boundary
```

Lemma 4.2

Algorithm 1 finds the boundary qubits of a layout l with n_l qubits in $\mathcal{O}(n_l)$.

Proof. Algorithm 1 iterates through each qubit $q \in l$ exactly once to determine

whether $q \in B_l$ where B_l denotes the set of boundary qubits of layout l. For this, the algorithm checks whether $n_q \in l \; \forall \; n_q \in neighbour(q)$. The time required for this is $\mathcal{O}(d_q)$ where d_q denotes the degree of q. Majority of the current quantum devices conform to a planar graph architecture. Therefore, the degree of the qubits are bounded. Since d_q does not depend on the length of the layout, the overall time requirement to determine the boundary qubits of l is $\mathcal{O}(|l|)$. □

Algorithm 2 Check b-overlap between two layouts

Input: Layouts l_i, l_j, coupling map M of the hardware, desired buffer distance b
Output: True if *overlap*, False otherwise

```
1: overlap ← False
2: if l_i ∩_0 l_j ≠ φ then
3:    overlap ← True
4:    return overlap
5: end if
6: boundary_1 ← find_boundary(l_i, M)
7: boundary_2 ← find_boundary(l_j, M)
8: for each qubit q_i in boundary_1 do
9:    for each qubit q_j in boundary_2 do
10:      d ← distance(q_i, q_j)
11:      if d ≤ b then
12:         overlap ← True
13:         break
14:      end if
15:   end for
16: end for
17: return overlap
```

Lemma 4.3

Given two layouts l_i and l_j, with n_i and n_j qubits of which k_i and k_j denote respectively the number of boundary qubits of the two layouts, then Algorithm 2 finds the overlap between the two layouts in time $\mathcal{O}(\max\{n_i, n_j\} + k_i \cdot k_j)$.

Proof. The algorithm 2 first checks for 0-overlap between l_i and l_j. This overlap can be determined in $\mathcal{O}(\min\{|l_i|, |l_j|\})$. When the two qubits are 0-overlapping, the

algorithm determines the boundary qubits for both the layouts in $\mathcal{O}(\max\{|l_i|, |l_j|\})$ if done in parallel. Let $B_i \subseteq l_i$ and $B_j \subseteq l_j$ denote the set of boundary qubits of l_i and l_j respectively, where $|B_i| = k_i$ and $|B_j| = k_j$. Now, the algorithm checks for the distance between every $b_i \in B_i$ and $b_j \in B_j$ to determine the buffer b in $\mathcal{O}(k_i \times k_j)$.

Thus, the overall time complexity of the algorithm is $\mathcal{O}(\max\{|l_i|, |l_j|\} + k_i \times k_j)$, where $\max\{|l_i|, |l_j|\}$ accounts for the time complexity of finding the boundaries of l_i and l_j, followed by the time $k_i \times k_j$ required for the pairwise distance checking between boundary qubits of the two layouts.

□

Given a circuit i and a hardware H, let L be the list of all isomorphic layouts obtained from mapomatic. Let $l \in L$ be the best layout with the lowest mapomatic score. We define a subset $L_\epsilon \subseteq L$ such that for each $l' \in L_\epsilon$, $Q_{il'} - Q_{il} \leq \epsilon$. Recall that by definition, the lower the mapomatic score, the better is the layout.

The problem at hand, therefore, is to schedule a set of circuits C to a set of layouts L_ϵ for each circuit such that no two layouts corresponding to two different circuits are b-overlapping. The theoretical formulation and the solution of this problem does not depend on a specific value of ϵ. Therefore, for the rest of the paper, we exclude any explicit mention of ϵ. However, whenever a list of layouts L is mentioned, it implies L_ϵ for a pre-specified ϵ. In Sec. 4.4 we shall discuss the choice of ϵ for our experimental results.

4.2.2 ILP Formulation for our scheduling problem

Let us formulate an integer linear program (ILP) for the noise-aware intra device scheduling (NIDS). Consider a list of circuits $C = \{C_1, C_2, ..., C_N\}$, and a list of hardware layouts $L_{all} = L_1 \cup L_2 \cup ... \cup L_N$, where L_i denote the layouts feasible for circuit C_i. For example, in Fig. 4.2 (b), the two layouts differ in the number of

qubits, therefore the layout for one circuit is not feasible for the other. However, there may also be cases, where the two circuits i and j have equal number of qubits, and therefore the same layout may belong to both L_i and L_j. For the rest of this paper, we shall remove the index, and simply use L to denote the list of layouts for any circuit i. The index is same as that for the circuit in the context.

Furthermore, it may not be possible to accommodate all the circuits simultaneously on the hardware. This may be because there are not sufficient quality layouts, or the total number of qubits required exceeds the number of physical qubits in the hardware. Therefore, the goal is to accommodate the largest subset of circuits simultaneously, and repeat this process of intra-device scheduling to schedule all the circuits in k batches $B_1, B_2, ..., B_k$, where each batch is a set of circuits which can be executed simultaneously, and k is expected to be significantly smaller than N. Note that this method can accommodate for instances where the circuits are added to the job queue dynamically.

Next we discuss the variables, constraints and the objective function of the ILP for scheduling the largest subset of circuits simultaneously on a hardware.

1. **Variables**

   (a) *Indicator variables*: We associate x_{ij} for each circuit $i \in C$ and layout $j \in L$ such that

   $$x_{ij} = \begin{cases} 1 & \text{if circuit } C_i \text{ is scheduled to layout } j \\ 0 & \text{otherwise.} \end{cases}$$

   (b) *Score variables*: A score variable q_{ij} is associated with each x_{ij} which is the mapomatic score when circuit C_i is placed on layout j.

2. **Constraints**

   (a) Since x_{ij} are indicator variables, we require that $\forall\ i, j$

   $$x_{ij} \in \{0, 1\} \tag{4.1}$$

(b) The second requirement is that every circuit C_i is assigned to at most one layout. Note that if a circuit is not scheduled in a particular batch, then it is not assigned any layout, and hence $\forall\ j,\ x_{ij} = 0$. Formally, this constraint can be represented as

$$\sum_{j \in L} x_{ij} \leq 1 \tag{4.2}$$

Note that this constraint should hold for all circuits $C_i \in C$, so there are $|C|$ such constraints.

(c) The third requirement is that no circuit is placed on a layout which has overlap with a previously mapped circuit, i.e., all circuits must be mapped to non-intersecting layouts.

$$x_{ij} + \sum_{k \neq i \in C} \sum_{j \cap_b l \neq \phi, j, l \in L} x_{kl} \leq 1 \tag{4.3}$$

This constraint implies that two circuits i and k should not be placed on b-overlapping layouts. As before, this constraint should hold for all circuits $i \in C$, and therefore, there are $|C|$ such constraints.

3. **Objective Function**: The objective of this optimization problem is to maximize the overall fidelity, which translates to minimizing the overall score Q along with maimising resource utilisation on the given hardware device. Therefore, the objective function is given by:

$$Minimize \sum_{i \in C,\ j \in L} \sum q_{ij} A_i x_{ij} - \sum_{i \in C,\ j \in L} \sum x_{ij} \tag{4.4}$$

where A_i, $0 < A_i \leq 1$ denotes the area of circuit i normalized over all available circuits. The area of a circuit is defined as the product of the number of qubits and its depth.

Note that the second term ensures that the objective function can have negative values; otherwise the least attainable value would have been 0, which could be attained if no circuit is placed at all. Inclusion of this term

ensures that circuits with higher area (i.e., more amenable to noise) are placed in layouts with lower mapomatic score to ensure quality of outcome.

The solution to this ILP shall provide the largest subset/batch of circuits $B \subseteq C$ which can be executed simultaneously on a given hardware. When $B \subset C$, the same ILP can be solved for $C \setminus B$ repetitively to schedule all the circuits into batches. Let $B_1, B_2, ..., B_k$ denote the batches, where it is expected that $k < N$. This reduces the number of times the quantum hardware is accessed, and in its turn increases the throughput and the hardware utilization.

Note that in current cloud providers, jobs enter the queue dynamically. Therefore, the set of all circuits C may not be static and known a priori. However, this can be easily accounted for as follows. At a particular timestamp T, let the list of available circuits be C, of which C' has been allocated simultaneously to the hardware. Let $\bar{C}$ denote $C \setminus C'$. While these circuits in C' are being executed, let C_T be the set of new circuits which are added to the queue. IN the next iteration the ILP is solved on $\bar{C} \cup C_T$ to find the largest subset of circuits to be sent for simultaneous execution at the next timestep.

Obtaining the outcomes of the individual circuits

Let $B = \{c_1, c_2, ..., c_r\}$ be the batch of circuits executed simultaneously on the hardware. If the number of qubits on circuit c_i be n_i, then effectively the hardware computes a circuit of $n = \sum_{1 \leq i \leq r} n_i$ qubits. The outcome of this execution will be a probability distribution over n qubits. The outcome of the circuit c_i can now be obtained by marginalizing over the rest of the $n - n_i$ qubits.

For example, in Fig. 4.2 (b), the hardware effectively executes a circuit with $(13 + 8) = 21$ qubits. The outcome will be a probability distribution encompassing all the 21 qubits. The distribution of the first circuit (in purple) can now be obtained by marginalizing over the remaining 8 (in orange) qubits, and vice versa. Note that although the hardware executes a single circuit, it is essentially a combination

of multiple disjoint circuits with no entanglement flowing from one to the other. Therefore, such a marginalization does not lead to any loss of information.

Noise-aware intra device scheduling is NP-Hard

Here we show that intra-device scheduling of N circuits is essentially a bin packing problem. The N circuits are the items. The bins are the batches $B_1, B_2, ..., B_k$ and the capacity of each batch is the total number of qubits on the hardware. Thus the objective of this problem is to minimize the number of batches, each having a capacity equal to the number of qubits on the hardware, to pack N quantum circuits in bins of equal capacity. Since the intra-device scheduling is equivalent to a bin packing problem, it is NP-Hard.

Note that our problem at hand imposes additional constraints to this. It requires that the circuits (i.e., the items) placed in each batch (i.e., each bin) do not have b-overlapping layouts. This constraint implies that the layouts of two circuits with b-overlap cannot be placed in the same batch even if the capacity supports it. The bin-packing problem is reducible to this problem in polynomial time, therefore our problem is NP-Hard. Further, the problem with the layout buffer constraint is at least as hard as the bin packing problem.

The ILP formulation of this NP-Hard problem being computationally expensive and not scalable, we propose in the next section a polynomial time graph-based greedy heuristic algorithm for noise-aware intra device scheduling.

4.3 Polynomial time heuristic algorithm

In this section we present a polynomial time heuristic algorithm for solving the noise-aware intra device scheduling problem described in the previous section. First, we create a compatibility graph for the circuits and their layouts: each

vertex denotes a distinct circuit and its corresponding layout, and an edge between two vertices denotes that the two circuits are distinct, with the two layouts not b-overlapping. The edges are weighted with a function of the mapomatic score of the two associated vertices. We finally propose a polynomial time heuristic algorithm to determine a *maximal* clique from this compatibility graph, which denotes the set of circuits that can be executed simultaneously. We shall discuss each step of this approach in detail.

4.3.1 Generation of the compatibility graph

The generation of the compatibility graph G is given in Algorithm 3.

Lemma 4.4

Algorithm 3 constructs the compatibility graph for given set C of N quantum circuits and the lists of their respective layouts in $\mathcal{O}((N.M)^2 \times n)$ where M denotes the overall number of layouts, and n is the length of the largest layout, or, in other words, the number of qubits in the largest circuit.

Proof. In order to construct the compatibility graph, the algorithm first generates the set of vertices, each of which is a (circuit, layout) pair, in $\mathcal{O}(N.M)$ by iterating through each layout for each circuit. Next, for each pair of vertices, it calculates overlap between the two associated layouts l_1 and l_j in time $\mathcal{O}(\max\{|l_i|, |l_j|\} + k_i \times k_j)$ as per Algorithm 2 to generate the compatible edges. The time to calculate overlap is dominated by the length of the largest layout, say n. Since, there are $\mathcal{O}(\binom{N.M}{2})$ possible pairs of vertices, the overall time requirement for generating the set of edges is $\mathcal{O}(\binom{N.M}{2} \times n)$. Finally, subtracting the weight of each edge from the maximum weight requires $\mathcal{O}(\binom{N.M}{2})$. Hence, the overall time required

Algorithm 3 Generate compatibility graph

Input: A set C of circuits ; for each $i \in C$ a list L_i of isomorphic layouts and the normalized circuit area A_i; for each $i \in C$ and $j \in L_i$ a mapomatic score q_{ij}; a buffer distance b

Output: Compatibility graph G

$G \leftarrow$ empty graph.
for each $i \in C$ **do**
  for each $j \in L_i$ **do**
    vertex $v = (i, j)$
    Add vertex v to G
  end for
end for
for each pair of vertices (i, j) and (k, l) in G **do**
  if i == k **then**
    Continue
  end if
  overlap $\leftarrow$ overlap between j and l calculated using Algorithm 2
  if overlap $\geq b$ **then**
    edge $e = ((i, j), (k, l))$
    weight of edge $w(e) = q_{ij}.A_i + q_{kl}.A_k$
    Add weighted edge $\{e, w(e)\}$ to G
  end if
end for
max_weight $\leftarrow$ max$\{w(e)$ for edge $e \in G\}$
for each edge $e \in G$ **do**
  $w(e) =$ max_weight $- w(e)$
end for
return G

to generate the compatibility graph is $\mathcal{O}(N.M)+\mathcal{O}(\binom{N.M}{2} \times n)+\mathcal{O}(\binom{N.M}{2}) = \mathcal{O}((N.M)^2 \times n)$.

□

Scheduling the optimal number of circuits simultaneously, thus, boils down to finding the maximal clique from this graph. Since each vertex in a clique is connected to every other vertex, each of these circuits can be executed simultaneously. Each

vertex layout is provided a weight which is the product of the mapomatic score for that layout and the normalized circuit area of the circuit. Every edge is associated with a weight which is the sum of the two weights of the associated vertices. However, recall that the lower the mapomatic score, the better (less noisy) is the layout. To convert this to a maximization problem, we find the largest edge weight, and subtract each edge weight from it. Thus, the edges corresponding to higher mapomatic score, i.e., more noisy layouts, now have lower weights, and vice versa. Hence, selecting the maximal clique from this graph ensures selection of edges corresponding to lower mapomatic scores.

In Fig. 4.3 (a) we take three example circuits, and show the construction of the compatibility graph in (b) and (c) of the same figure. We have selected the circuits to be of the same number of qubits and depth for this example, making $A_i = 1$ for all of them. In particular, Fig. 4.3 (b) shows the scenario where each circuit has two compatible isomorphic layouts, and their mapomatic score. Usually, there will be many more such layouts for each circuit, but for this example we stick to two layouts for brevity. The compatibility graph is created from this information where each vertex i, j corresponds to the layout j for vertex i. From Algorithm 3, no two vertices corresponding to the same vertex will have associated edge, since the same circuit is not to be placed twice. Therefore, no edge exists between any vertex with the same circuit index.

Two vertices are connected by an edge only if the layouts are not b-overlapping. For this example figure, we have selected $b = 1$. Thus, there is an edge between, say vertices (00) and (11), but not between (00) and (10) since they have a common qubit (39) in their layouts. The mapomatic scores for, say vertices (00) and (11), are 0.0932 and 0.0833. So, initially we assign the weight of the edge to be the sum of these two mapomatic scores, i.e., 0.1765. After assigning weights to every edge similarly, we see the largest weight 0.1805 is associated with the edge corresponding to vertices (01) and (10). Therefore, we subtract all the edge weights from this value, thus yielding the final weight of the edge corresponding to vertices (00) and (11) to be 0.004. The other edges and their weights are similarly calculated.

A maximum clique in this compatibility graph provides the optimal noise-aware intra device scheduling. However, finding a maximum clique in any arbitrary graph is NP-Hard as well. In the following subsection we propose a greedy approach to find a *maximal* clique in the compatibility graph for our scheduling problem.

4.3.2 Greedy algorithm to find a *maximal* clique in the compatibility graph

Algorithm 4 first determines the connected components of the compatibility graph. For each connected component it finds a maximal clique using a greedy method. It first selects the edge with the largest weight, and then keeps adding edges, sorted in descending order of weight, as long as the vertices are connected to all the vertices already selected (i.e., it is a clique). This ensures that the layouts selected are compatible with *all* other layouts, and all the selected circuits can be scheduled simultaneously on the hardware. Note that this method generates one maximal clique for each connected component. Finally, the maximal clique with the largest weight among the ones found for each of the connected components is selected as the solution to the noise-aware intra-device scheduling.

Lemma 4.5

Algorithm 4 finds a *maximal* clique in the compatibility graph in $\mathcal{O}(|V| \cdot |E| + |E| \log |E|)$

Proof. Let $G = (V, E)$ be the compatibility graph. First, the algorithm identifies the connected components of the graph. This can be achieved using a Breadth-First-Search in $\mathcal{O}(|V| + |E|)$ time. Let $g = (V_g, E_g)$ denote a connected component. For each connected component, the algorithm first sorts the edges in $\mathcal{O}(|E_g| \log |E_g|)$ time. Next, for each edge, the algorithm checks whether the two associated vertices are connected to all the vertices already present in the constructed clique. For each edge, this can be performed in $\mathcal{O}(V_g)$. Therefore, performing this check for all the edges requires $\mathcal{O}(|V_g| \cdot |E_g|)$ time.

This exercise is repeated for all the connected components. Therefore, the overall time requires is $\sum_g \mathcal{O}(|V_g| \cdot |E_g|) + \mathcal{O}(|E_g| \log |E_g|)$. Now from Lemma 4.6 and Lemma 4.7: $\sum_g \mathcal{O}(|V_g| \cdot |E_g|) + \mathcal{O}(|E_g| \log |E_g|) = \mathcal{O}(|V| \cdot |E|) + \mathcal{O}(|E| \log |E|)$. Finally, the weight of the clique for each connected component can be calculated in $\mathcal{O}(|E_g|)$, thus requiring a total of $\sum_g \mathcal{O}(|E_g|) = \mathcal{O}(|E|)$ for all the components.

Hence the time complexity of Algorithm 3 is

$$\mathcal{O}(|V| + |E|) + \mathcal{O}(|V| \cdot |E|) + \mathcal{O}(|E| \log |E|) + \mathcal{O}(|E|) = \mathcal{O}(|V| \cdot |E| + |E| \log |E|)$$

□

Lemma 4.6

$$M \geq M_1 \log M_1 + M_2 \log M_2 + ...$$

Proof.

$$\begin{aligned} M &= \sum_g M_g \\ M \log M &= \sum_g M_g \log \sum_g M_g \\ &= M_1 \log \sum_g M_g + M_2 \sum_g M_g + ... \\ &\geq M_1 \log M_1 + M_2 \log M_2 + ... \end{aligned}$$

where the final inequality follows since $M_g \geq 0 \; \forall \; g$ and logarithm is a monotonically increasing function. □

Lemma 4.7

$$N \cdot M \leq \sum_{g} M_g \cdot N_g$$

Proof.

$$\begin{aligned} N \cdot M &= (\sum_{g} M_g) \cdot (\sum_{g} N_g) \\ &= \sum_{g} M_g \cdot N_g + \sum_{g \neq h} M_g \cdot N_h \\ &\leq \sum_{g} M_g \cdot N_g \end{aligned}$$

where the final inequality follows since $M_g \geq 0$ and $N_g \geq 0 \; \forall \; g$. □

In the next section, we present experimental results of our method to show the improvement in throughput obtained and the quality of the outcome.

4.4 Experimental results

For our experiments with our proposed greedy method, we have considered 4 benchmarks circuits, namely Real Amplitude, Trotterized Clustered Unitary, QAOA, and Ripple carry adder. Although our formulation (Sections 4.2 and 4.3) do not impose any constraints on the type of circuits that can be scheduled together, we report here for only the circuits from the same family to study intra-device scheduling. A more rigorous experiment, with circuits from different families scheduled together, will be reported in another article separately.

For each circuit, we created its mirrored version. For a given circuit with unitary U, a mirrored circuit of it can be obtained by appending $U^{\dagger}$ to the original circuit.

This simple modification implies that the ideal outcome of the circuit is always $|0\rangle^{\otimes n}$, n being the number of qubits in the circuit. The advantage of such a circuit is that the ideal outcome is known without any simulation. However, the disadvantage is that the depth of such a circuit is twice that of the original circuit, and is hence more amenable to noise. In Qiskit [AA+19], it is necessary to put a barrier between U and $U^\dagger$ in order to avoid simplification of the circuit to identity. An assumption for this work is that all the circuits that are being batched require the same number of shots.

We first provide the rationale behind selecting the ϵ for layouts in our experiment (refer to Sec. 4.2), and then show the fidelity obtained and the increase in throughput for a 27-qubit fake backend, and a 127-qubit IBM Quantum device.

4.4.1 Selection of ϵ for the layouts

It is expected that there is overlap between the layouts returned by mapomatic. We have taken only the layouts having a score which is at least 50% of the highest score and further we check for overlap among them and other conditions. If this percentage is increased, we can accommodate more circuits with less fidelity.

Table 1 gives a comparison of the values of fidelity for taking the best score, the worst score and the last of top 50% in the noisy simulator of IBMQ Kolkata for 5-qubit circuits.

Table 4.1: Comparison of values of fidelity for the best score, the worst score and the last of the top 50% in noisy simulator of IBMQ Kolkata for 5-qubit circuits

Benchmark Circuit	2Q Depth	Fidelity		
		Best	Worst	Last of the top 50%
Real Amplitude	8	0.944	0.805	0.925
QAOA	8	0.879	0.638	0.842
Trotterized	18	0.749	0.29	0.638

4.4.2 Fidelity and hardware utilization in intra-device scheduling

In Table 2, we consider the Real Amplitude circuits of different qubits to be run into the noisy simulator or IBMQ Kolkata (27-qubit) with and without using the intra-device scheduling. A total of 7 circuits of each type was chosen for the experiments. Here 2 circuits are placed in the hardware simultaneously exhibiting a better throughput and resource utilization. We show that the values of fidelity where we are using intra device scheduling is almost reaching the fidelity if the circuits are one to one mapped in the best available hardware. In our experiments we have used buffer b=1, i.e., between two circuit mapped there should be a gap of at least 1 qubit. This is to minimize the cross talk where it is maximum if two circuits are placed without a single qubit barrier [BM+23].

Table 4.2: Fidelity for different sized Real Amplitude circuits with and without using our intra-device scheduling to be run on Noisy IBMQ simulator with the noise profile and coupling map of 27-qubit IBMQ Kolkata

Circuit size # qubits	Circuit Count	$Fidelity_{Int}$	$Fidelity_{NoInt}$
3	7	0.9542837452	0.959822345
5		0.9244571429	0.959822345
7		0.8468928571	0.862323176
10		0.6612723723	0.675571234

In Fig. 4.4, we consider the benchmark circuits QAOA, Trotterized, Real Amplitude having different number of qubits to be run on (a) Noisy IBMQ simulator with the noise profile and coupling map of 27-qubit IBMQ Kolkata , and (b) 127-qubit IBMQ Brisbane hardware. With intra-device scheduling, 3 circuits are placed in the hardware to be executed simultaneously and thereby exhibiting a better throughput and resource utilization. We show that the values of fidelity where we are using intra-device scheduling is almost reaching the fidelity if the circuits are mapped one by one in the best available hardware. We have also given the mean and standard deviation for each of the points, where Mean is the average value of the dataset, indicating the central point and Standard Deviation is the

measure of the spread or dispersion of the dataset around the mean.

Table 4.3: Hardware utilization in intra device scheduling

Circuit size	Hardware size m	# circuits placed simultaneously	Gain w.r.t. time
7	27	2	2x
10	27	2	2x
7	127	3	3x
10	127	3	3x

If we have included more circuits in the hardware simultaneously of course the hardware utilization would be better but we constrained our solution with the top 50% score of the best score from the hardware layout. Note thatL: Number of swap gates for each of these layout will be equal because the mapomatic solution uses graph isomorphism to cimpute the possible layouts. The time and utilizatation can be improved with the expense of fidelity.

4.5 Conclusion

In this paper, we addressed the critical challenge of optimizing quantum circuit scheduling to enhance the throughput and efficiency of quantum computing hardware. By drawing analogies to the classical bin packing problem, we demonstrated the NP-Hard nature of our problem, which involves placing multiple quantum circuits onto quantum processing units while considering the inherent noise and limited qubit connectivity. Our proposed solution, using integer linear programming or the greedy heuristic based solution on compatibility graphs and maximal cliques, effectively balances the trade-off between noise reduction and throughput optimization. The experimental results showed significant improvements in time utilization, achieving 2x and 3x better efficiency for 27-qubit and 127-qubit hardware, respectively. These findings highlight the potential of intra-device scheduling to maximize the performance of NISQ-era quantum computers, paving the way for more reliable and scalable quantum computing solutions in the future.

It is intuitive that if we increase the number of layouts allowed for further pro-

cessing from 50% of top scores, then the utilization will be better where as the fidelity can be worse. This trade-off between number of layout vs fidelity will be studied experimentally as a future work. Moreover how the buffer distance affects the fidelity is also a work which is left for future studies.

Recent advancements in hardware, such as tunable coupler designs for superconducting qubits [SZU+21] employed by Google and IBM, have significantly mitigated the impact of hardware crosstalk. This development allows for reduced buffer distances and enables tighter packing of quantum circuits, potentially further enhancing hardware utilization. Future work could explore adapting scheduling algorithms to leverage these improvements in hardware design for even greater efficiency.

As we transition from the NISQ era to the fault-tolerant era, our attention shifts in the next chapter to innovative machine learning approaches for decoding quantum error correction codes.

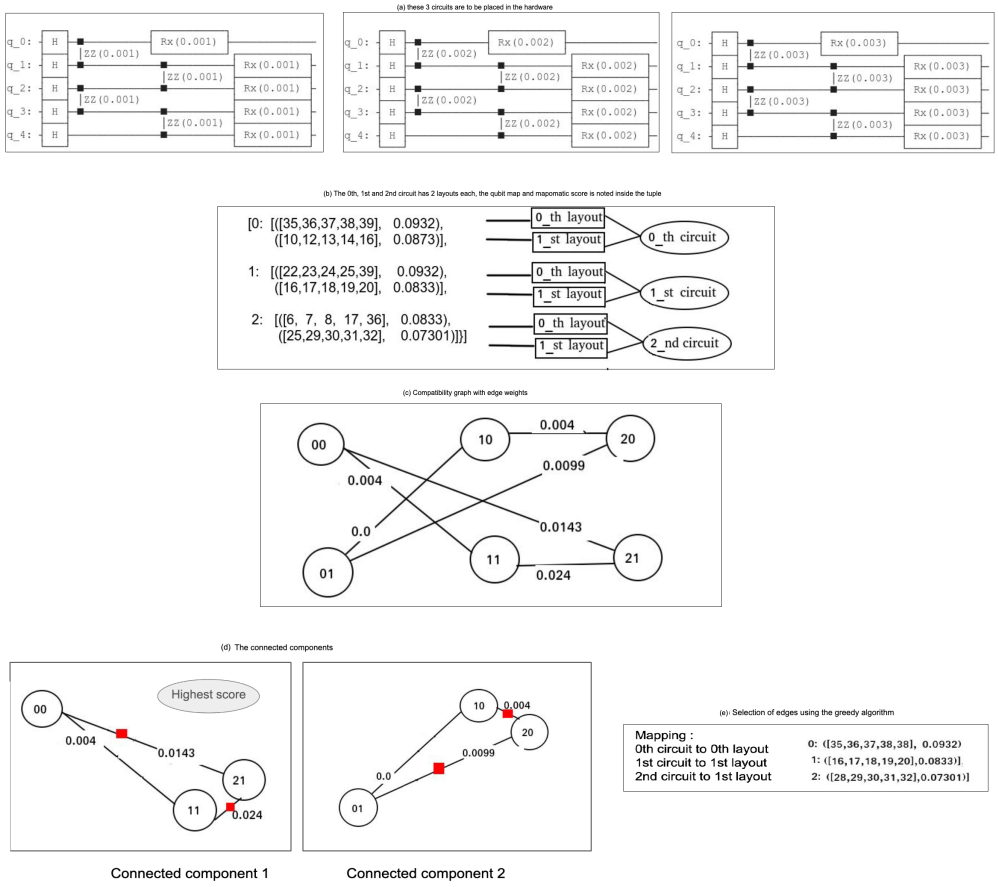


Figure 4.3: The entire workflow of our heuristic algorithm: (a) schematic diagrams of three circuits that are to be placed in the hardware; (b) two possible layouts for each of the three circuits and their corresponding mapomatic scores; (c) the compatibility graph with edge weights, (d) the connected components of the graph and (e) greedy selection of edges in each of the components.

Algorithm 4 Finding maximal clique in a compatibility graph

Input: compatibility graph $G = (V, E)$ where E is the weighted edge list
Output: maximal clique in G

$G_C \leftarrow$ the set of all connected components of G
max_clique $\leftarrow \phi$
max_clique_weight $= 0$
for each $g \in G_C$ **do**
  $V_g \subseteq V \leftarrow$ set of vertices in g, $E_g \subseteq E \leftarrow$ set of edges in g
  selected_circuits $\leftarrow \phi$, selected_layouts $\leftarrow \phi$, selected_edges $\leftarrow \phi$
  $E_{g_{sorted}} \leftarrow$ sorted E_g in the descending order of edge weight
  for each $e = (u, v) \in E_{g_{sorted}}$ **do**
    $l_u, l_v \leftarrow$ layouts associated with u and v respectively
    $c_u, c_v \leftarrow$ circuits associated with u and v respectively
    if selected_edges is ϕ **then**
      add e to selected_edges, c_u and c_v to selected_circuits l_u and l_v to selected_layouts, e to selected_edges
    else if $c_u \in$ selected_circuits **and** $c_v \in$ selected_circuits **then**
      Continue
    else if $c_u \notin$ selected_circuits **and** $c_v \notin$ selected_circuits **then**
      is_connected = True
      for all $c \in$ selected_circuits **do**
        if $(u, c) \notin E_g$ **or** $(v, c) \notin E_g$ **then**
          is_connected = False
          break
        end if
      end for
      if is_connected **then**
        add e to selected_edges, c_u and c_v to selected_circuits l_u and l_v to selected_layouts, e to selected_edges
      end if
    else if $c_u \notin$ selected_circuits **and** $l_u \in$ selected_layouts **then**
      is_connected = True
      for all $c \in$ selected_circuits **do**
        if $(u, c) \notin E_g$ **then**
          is_connected = False
          break
        end if
      end for
      if is_connected **then**
        add e to selected_edges, c_u and c_v to selected_circuits l_u and l_v to selected_layouts, e to selected_edges
      end if
    else if $c_v \notin$ selected_circuits **and** $l_v \in$ selected_layouts **then**
      is_connected = True
      for all $c \in$ selected_circuits **do**
        if $(v, c) \notin E_g$ **then**
          is_connected = False
          break
        end if
      end for
      if is_connected **then**

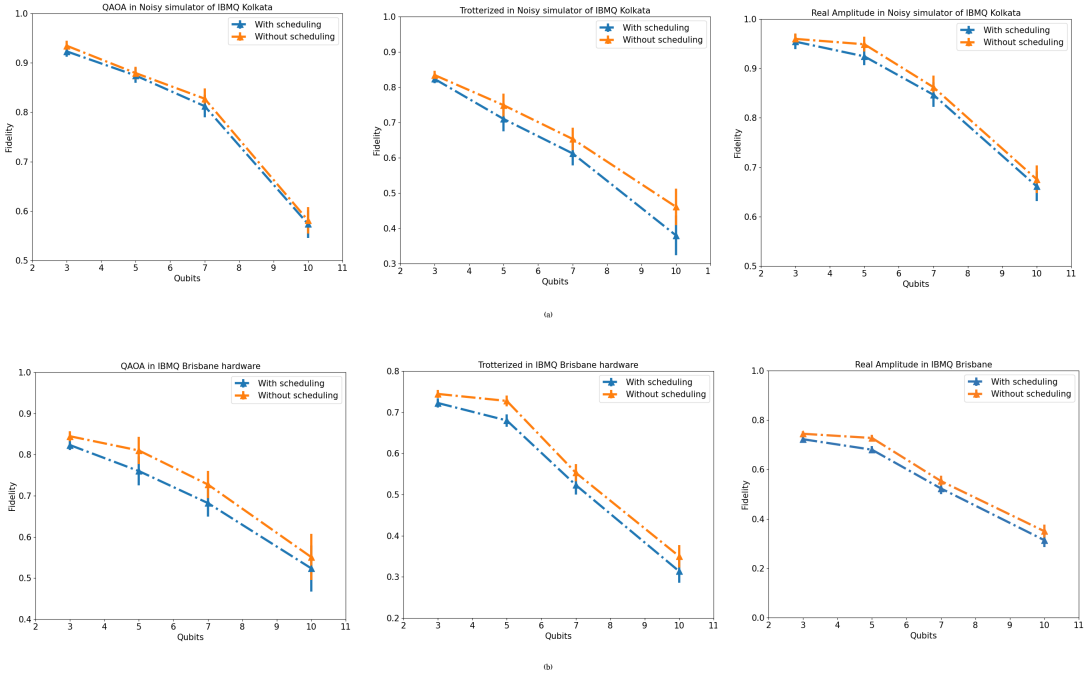


Figure 4.4: Fidelity (along with the mean and standard deviation) for benchmark circuits (QAOA, Trotterized, Real Amplitude) with and without using our intra-device scheduling executed in (a) Noisy IBMQ simulator with the noise profile and coupling map of 27-qubit IBMQ KOlkata, (b) 127-qubit IBMQ Brisbane hardware.

Part II

Error correction in Fault Tolerant Era

CHAPTER 5

Machine-Learning based Decoding of Surface Code Syndromes in Quantum Error Correction

Contents

5.1 Introduction 100

5.2 Design methodology of our ML based decoder 102

5.2.1 Mapping surface code onto a square lattice 102

5.2.2 Error injection and syndrome extraction 103

5.2.3 Training our ML model 104

5.3 Experimental Results 106

5.3.1 Noise models 106

5.3.2 Machine Learning Parameters 108

5.3.3 More sophisticated ML models 110

5.3.4 Empirical train-test-ratio for optimal accuracy 112

5.3.5 Performance on Training with Symmetric Noise Models and Testing with Asymmetric Noise Models 118

5.4 Conclusion 119

5.1 Introduction

Quantum states are highly susceptible to errors due to interactions with the environment, causing unwanted rotations in their Hilbert space vectors. Shor demonstrated that any unitary quantum error can be decomposed into Pauli matrices (I, X, Y, Z) [Sho95]. Thus, a quantum error-correcting code (QECC) that corrects Pauli errors can handle any unitary error. Notable QECCs include the 9-qubit, 7-qubit, and the optimal 5-qubit code [Sho95, Ste96, LMPZ96].

Traditional QECCs often require operations between non-adjacent qubits, which are slow and error-prone. The surface code addresses this by organizing qubits in a 2D grid, allowing operations only between adjacent qubits [BK98]. The efficiency of these protocols was demonstrated, and various decoding algorithms have been developed to enhance their performance [DKLP02, FSG09, WFH11].

QECCs encode multiple physical qubits into fewer logical qubits, making them more resistant to noise. Decoders detect errors in logical qubits and apply corrections. A distance d QECC can correct up to $\lfloor \frac{d}{2} \rfloor$ errors, but beyond this, logical errors may occur due to environmental interactions or faulty decoding. The efficacy of a decoder is measured by its pseudo-threshold and threshold values, indicating its performance relative to physical error rates [FWH12].

Decoding time is crucial, as qubits are decoded multiple times during computation. The widely-used Blossom Decoder for surface codes has a worst case time complexity of $\mathcal{O}(N^3 \log N)$ where N is the number of nodes in the detector graph. Machine learning (ML) offers a faster alternative, with linear time complexity, as demonstrated by recent studies [VBA19, CR18, VCB17, S+18]. ML-based decoders, such as deep neural networks, show potential for improved performance and speed over traditional methods.

ML decoders can address errors due to faulty decoding by using two-level decoding: a traditional low-level decoder followed by a high-level ML decoder [VCB17, VBA19]. However, questions remain about their performance under asymmetric noise and the effectiveness of more sophisticated ML models [IM07].

This chapter aims to explore the use of ML-based low and high-level decoders for both symmetric and asymmetric noise models, focusing on the mapping of surface code decoding to ML classification and evaluating the impact of ML model sophistication on decoding performance.

Machine learning (ML) is a branch of artificial intelligence where a machine learns from data without being explicitly programmed. Depending on the type of training data, ML algorithms can be supervised, unsupervised, or semi-supervised. ML has numerous applications, including soil properties prediction [KG22], human pose estimation [SB22], object recognition [JW22], video tracking [PR20], and predicting the efficacy of online sales [SM21]. Here, we employ ML to decode error syndromes for quantum error correction.

Classical algorithms like Minimum Weight Perfect Matching (MWPM) may perform poorly in some cases, as they do not consider error probabilities. ML decoders learn these probabilities during training, improving their accuracy. ML techniques have shown promise in reducing decoding time and improving performance, even for complex scenarios.

Since a decoder can introduce logical errors, two-stage decoding with low-level traditional decoders and high-level ML decoders can improve accuracy. This approach allows for better handling of logical errors that may arise during physical error correction.

Overall, ML decoders are expected to outperform traditional methods in both performance and speed, making them a valuable tool for quantum error correction.

5.2 Design methodology of our ML based decoder

Artificial neural networks (ANN) are made to emulate the way human brains learn, and are one of the most widely used tools in ML. Neural networks consist of one input layer, one output layer, and one or more hidden layers consisting of units that transform the input into intermediate values from which the output layer can find patterns that are too complex for a human programmer to teach the machine. The time complexity of training a neural network with N inputs, M outputs and L hidden layers is $\mathcal{O}(N \cdot M \cdot L)$. In this work we are using neural networks as both low-level and high-level decoder for distance 3, 5, and 7 surface code.

In order to apply ML techniques to surface code decoding, we first map the decoding problem to the classification problem as follows. Given a set of data points, a classification algorithm predicts the class label of each data point. These techniques are purely classical. Next, we describe in detail the formulation of a decoder for surface code as a classification problem.

5.2.1 Mapping surface code onto a square lattice

For ease of implementation, we have mapped the surface code to a square lattice (refer Fig. 5.1) in this work. This has been achieved by padding a few dummy nodes (labelled as 0_D in the figure). A distance d surface code is converted into a (d+1) $\times$ (d+1) square lattice which has $d^2 - 1$ stabilizers, when encoding a single logical qubit. Therefore, 2(d+1) dummy nodes are required for this square lattice. The dummy nodes are basically don't care nodes, and their value is always 0 irrespective of the error in the surface code. The syndrome changes the values of the stabilizers only.

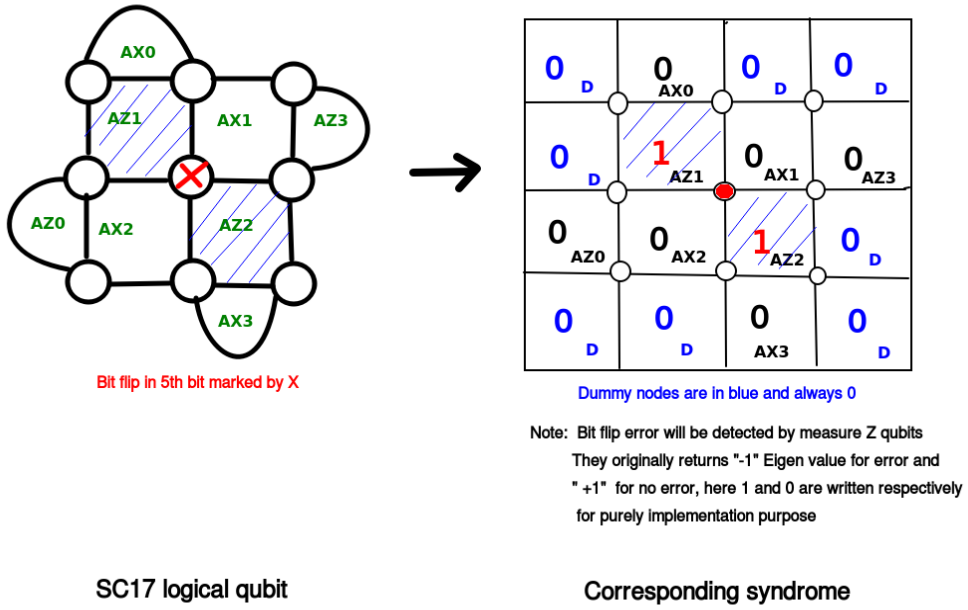


Figure 5.1: SC17 to syndrome generation

5.2.2 Error injection and syndrome extraction

Once the distance d surface code is transformed to a (d+1) × (d+1) square lattice, the next step is to extract the syndrome for errors. First, we create a training dataset, where in each data we randomly generate errors on each physical qubit. If p_{phys} is the probability of error on a physical qubit, the total probability of error after the 8 steps of surface code cycle (Fig. 2.7) is 1 - (1-p_{phys})^8. We have trained the networks with p_{phys} ranging from 0.0001 to 0.25. One can argue that 0.25 is an unreasonably high error probability. However, we have ranged the error probability that far to show an interesting observation regarding the ML decoder performance (Sec 4).

For generating the training data we have considered bit flip errors, symmetric and asymmetric depolarizing noise models. We have not separately considered phase flip errors since they are similar to bit flips and have a rotational symmetry (i.e., the logical errors of bit flip and phase flip model are equivalent up to a rotation

by $\frac{\pi}{2}$).

From the training data (which may or may not contain errors), we generate the syndrome (measured by ancilla qubits of the surface code) (Fig. 5.1). The syndrome, in our implementation, contains both the ancilla and the dummy nodes. However, the dummy nodes are always 0, whereas the values of the ancilla changes with different errors. Henceforth, in terms of implementation only, syndrome for a distance d surface code will imply $(d+1)^2$ values including ancilla and dummy nodes. The final training data contains the syndrome, and its corresponding label is the true set of errors that have occurred in the system. Note that this method can lead to multiple labels having the same syndrome. This agrees with the fact that surface code does not have one-to-one mapping from error to syndrome.

Ideally, the dataset to achieve the best decoding performance should include all possible error syndromes. But as the code distance increases, the state space also increases exponentially. Therefore, we can at most include only a small percentage of the entire input dataset. The dataset size that we have used is 100000 from which 70000 is used for training and the rest for testing purpose.

5.2.3 Training our ML model

For the low level decoder, we train a neural network where the input layer is the syndrome and the output layer denotes the types of errors along with the physical data qubit where each error has occurred. For a distance d surface code, the number of input nodes are $(d + 1)^2$ containing d^2 - 1 measure qubits and $2(d+1)$ dummy nodes. For example, if we consider a distance-3 surface code (SC17), it has 8 ancilla qubits and 8 dummy nodes. Therefore, in the input layer, there are 16 nodes (Fig. 5.1). In the output layer, there are 2 nodes for each data qubit to differentiate among I, X, Y and Z errors. The size of the hidden layer can be adjusted by trial-and-error.

We have used two types of neural networks, (i) Feed Forward Neural Network

(FFNN) and (ii) Convolutional Neural Network (CNN). In our reported results,

(i) FFNN consists of 2 hidden layers having 32 and 16 nodes respectively. For the cost function we have used the mean squared error rate, and as the activation function we have used Rectified Linear Unit (ReLU).

(ii) For CNN, the first layer is a 64 dimension convolution layer where input is a 4×4 matrix and the kernel size is also 4×4. Then we flatten it and add two fully connected layers of dimension 64 and 32. After that we add the fully connected output layer of dimension 9. For the first 3 layers (convolution, dense, dense) we have used ReLu as the activation function and for the output layer we have used sigmoid activation function since it will be a multi-label classification problem.

These values were adjusted after multiple trial-and-errors. We later show in the result section that building a more complex neural network cannot provide any significant improvement in the performance of the decoder, but requires significantly more decoding time. Therefore, we stick to these parameters.

The high-level decoder simply tries to predict any logical error that has been incorporated by the low level decoder. Therefore, its input remains the same as the low-level decoder (i.e., the syndrome) whereas it has 4 nodes in the output, each corresponding to a logical Pauli operator.

First, the network is trained for low-level decoder. After the low-level decoding is done, the predicted corrections are applied, and rechecked by using the high-level decoder whether any logical error has been inserted by the low-level decoder. The entire workflow is given in Fig. 5.2.

5.3 Experimental Results

First, we focus on the decoding performance of an ML-based low-level and the high-level decoder for surface codes of distances 3, 5, and 7 for both symmetric and asymmetric depolarizing noise models with varying degrees of asymmetry. Our model outperforms the performance of the existing decoders for symmetric noise model. We also show that although the performance of ML is slightly poorer for asymmetric noise models than that for the symmetric one, it still outperforms MWPM. Furthermore, we provide an empirical study to estimate the minimum train-test-ratio needed for optimal accuracy to obtain a better estimate of the minimum number of training data required to obtain the best (or near best) decoding results with ML decoder.

In the following subsections, we first introduce the noise model that we have considered, followed by the parameters of our ML decoder. Finally, we show the results of our decoder and compare its performance with the traditional MWPM decoder.

5.3.1 Noise models

Given a quantum state ρ in its density matrix formulation [NC10], the evolution of the state in a depolarization noise model is given as

$$\rho \rightarrow (1 - p_x - p_y - p_z)\rho + p_x X\rho X^\dagger + p_y Y\rho Y^\dagger + p_Z Z\rho Z^\dagger$$

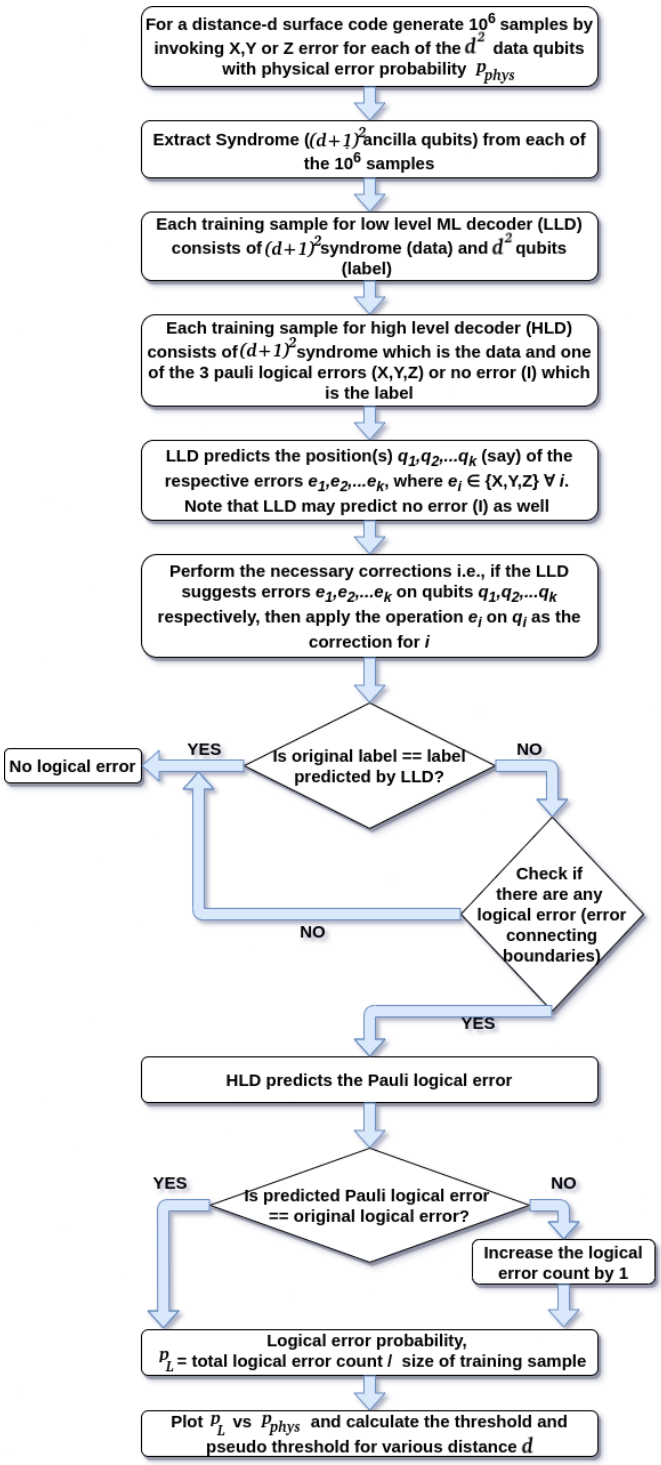


Figure 5.2: Outline of the ML based syndrome decoding for surface code

where p_x, p_y, p_z represent the probability of occurrence of unwanted Pauli X, Y, and Z error. In symmetric depolarization noise model, $p_x = p_y = p_z$. Moreover, quantum channels are often asymmetric or biased, i.e., the probability of occurrence of Z error is much higher than that of X or Y error. Furthermore, each error correction cycle in surface code requires eight steps. We have considered that an error can occur on one or more of the d^2 data qubits in each of the eight steps, where d is the distance of the surface code. Therefore, if $p_x + p_y + p_z = p$, then the overall probability of error for each error correction cycle is $1-(1-p)^8$. This error model is in accordance with [FSG09]. We assume noise-free measure qubits (which are almost half the total number of qubits) and ideal measurements.

5.3.2 Machine Learning Parameters

For our study, we have trained the ML model with batches of data, not the entire data set at once. This is often beneficial in terms of training time as well as memory capacity. We have used batch size = 10000, epochs = 1000, learning rate = 0.01 (with Stochastic Gradient Descent), and we have reported the average performance of each batch over 5 instances. This is repeated for each value of the p_{phys} considered here.

Low and high-level decoder

In Fig. 5.3, we show the increase in the logical error probability with physical error probability p, which is the probability of error per step in the surface code cycle. The results of MWPM and CNN-based low-level decoder for both symmetric and asymmetric noise models are shown. In Tables 5.1 and 5.3, we depict the performance of FFNN decoder as well. In Fig. 5.3, the blue, yellow, green, and red lines respectively are the decoder curves which show the probabilities of logical error for symmetric depolarization, bit flip (X), phase flip (Z), and Y errors. The cyan straight line consists of the points where the probabilities of physical and logical error are equal.

The point where the decoder curves and the straight line intersects, defines the value of pseudo-threshold for the decoder. As expected, the pseudo threshold improves with increasing distance of the surface code. Nevertheless, the threshold value is the probability of physical error beyond which increasing the distance leads to poorer performance. Therefore, threshold is independent of the distance and is a property of the surface code and the noise model only. In Tables 5.1 and 5.3, we show the pseudo-threshold and threshold of the low and high-level decoders for distance 3, 5 and 7 surface code in symmetric and asymmetric noise models respectively. Fig. 5.3 shows the pseudo-thresholds for MWPM and CNN decoder for a distance 3 surface code using low-level decoder (LLD) only. From Table 5.1 we observe $\sim 10\times$ increase in the pseudo threshold for ML-decoders as compared to MWPM.

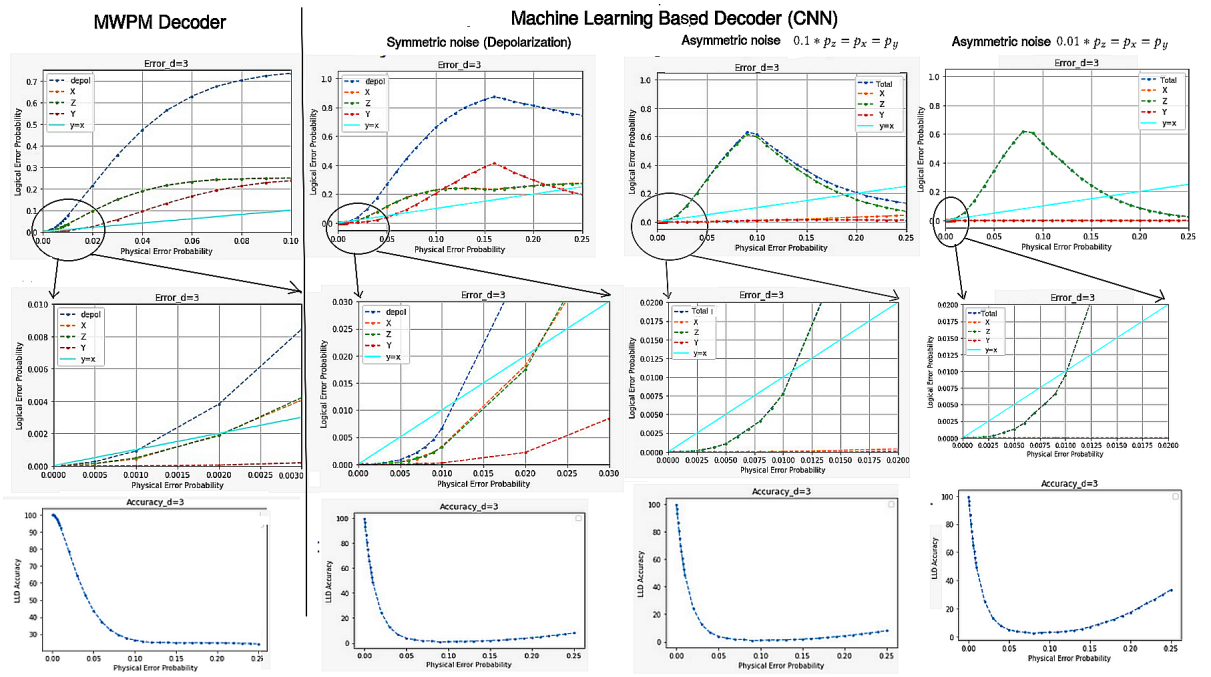


Figure 5.3: Pseudo-threshold and accuracy — MWPM vs ML-based decoder for distance 3 surface code

Fig. 5.4 shows the thresholds and decoder accuracy for MWPM and ML-decoders surface codes of distance 3, 5 and 7. Table 5.3 depicts the threshold values for MWPM and ML-decoders. From Table 5.3 we observe $\sim 2\times$ increase in the

threshold for ML-decoders as compared to MWPM.

As already mentioned earlier, this result assumes error-free stabilizers, and ideal measurements. For a distance d surface code, there are $d^2 - 1$ stabilizers. Therefore, in our setting, nearly half of the total qubits in the surface code structure are considered ideal. We focused more on the mapping of decoding to Machine Learning in this research. A separate study is being carried out on the performance of the ML decoder in the presence of erroneous stabilizers and measure qubits to determine the threshold and pseudo-threshold. Our conjecture is that ML decoders will still outperform MWPM decoder in that scenario, but the increase in performance will be much lower.

In Fig. 5.3, we observe that at very low error probability the accuracy remains good, then it falls drastically. However, for ML decoders, it again increases beyond a certain physical error probability (0.15). On the other hand, the logical error also decreases in most of the cases for both symmetric and asymmetric ML decoders after more or less that same value of physical error probability. This is due to the bias in the back-end working principle of any machine learning model. When the error probability is low or high, the ML decoder effectively learns the probability and in most of the cases can avoid logical errors. But when the error probability is in the mid range, the ML model gets confused. For example, if in a training set, out of 12 events with same value of the features, 10 events are certainly in class A, and the rest in B, then the ML definitely learns it with high accuracy. Similarly, if those same 10 events are in class B, accuracy will be high. But the ML is confused when 6 of them are in class A and 6 of them in class B. This is an interesting observation in the ML-decoder which is absent in MWPM-decoder.

5.3.3 More sophisticated ML models

A natural question is whether the use of more sophisticated ML models (e.g, adding more hidden layers, increasing the number of nodes in each layer, etc.) can improve the performance of the decoder. We have addressed this issue as reported below.

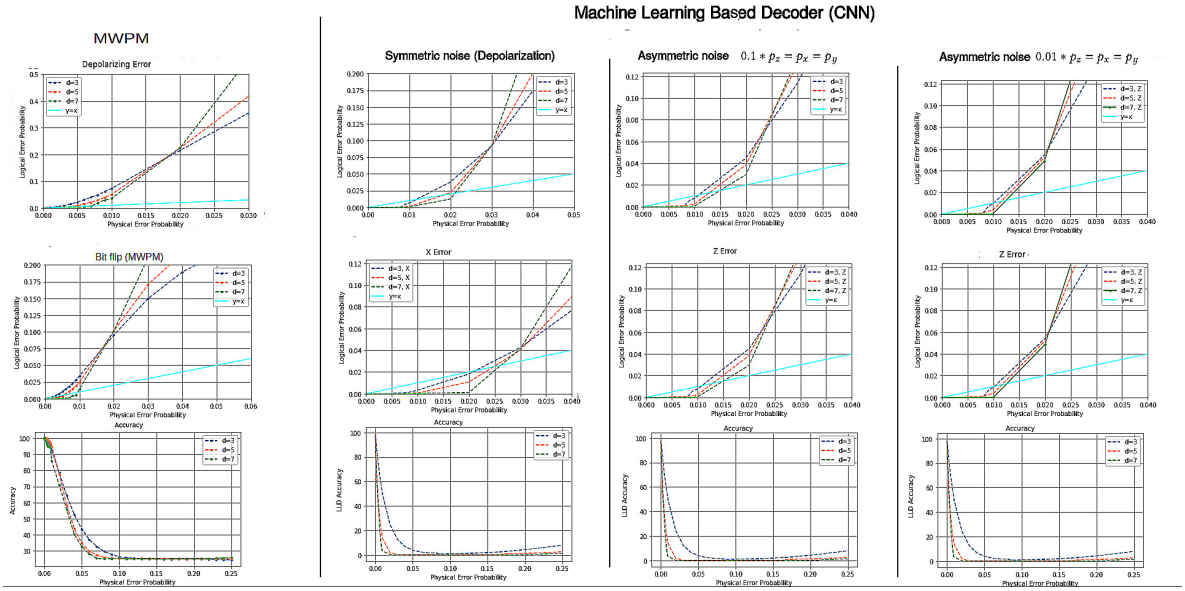


Figure 5.4: Threshold and accuracy — MWPM and ML-based decoder for surface code with $d = 3, 5, 7$

In Table 5.2, Simple FFNN has 1 hidden layer (dense) whereas Complex FFNN has 5 hidden layers (dense) and Simple CNN has 1 convolution (64 dimensions) followed by 2 dense layers of dimension 256 and 64 respectively before the output layer whereas Complex CNN has 3 convolutions (64 dimensions) layers followed by 4 dense layers of dimension 512, 256, 128 and 64 respectively before the output layer. The more sophisticated models naturally require more time for training and prediction. But from Fig. 5.7, we see that the decoder graphs are more or less overlapping for the simple and complex ML models. Therefore, it can be concluded that using more complex CNN / FFNN model does not lead to a better performance for the decoder. This can be further verified by the accuracy plots in Fig. 5.5. Since the more complex models are performing almost at par with the simpler models for d=3 and 5 and the complex models are significantly more time-consuming, we performed the experiments for $d = 7$ with only simple CNN and FFNN models.

While our results demonstrate that more sophisticated architectures such as CNNs and FFNNs do not outperform simpler models, this observation may not general-

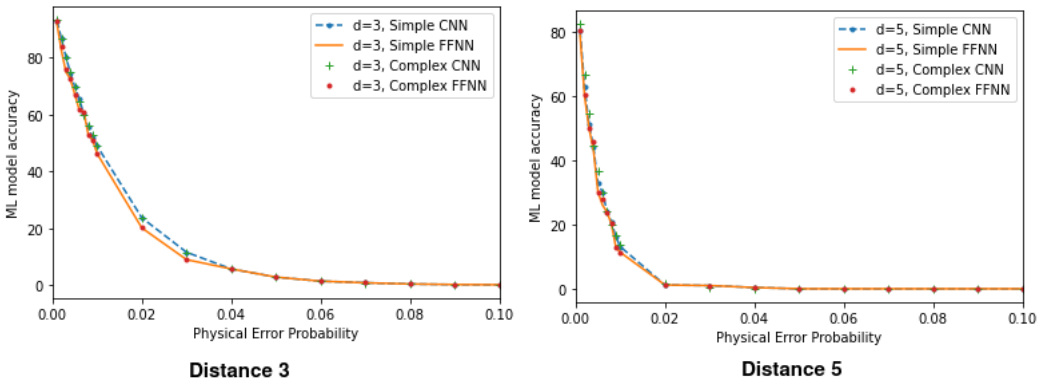


Figure 5.5: ML model accuracy vs physical error probability for various ML models in $d = 3$ and 5 surface code

ize to other ML techniques or differently structured datasets. An open problem remains to identify or design ML architectures and data representations that can leverage the complexities of quantum error correction and decoding tasks better.

5.3.4 Empirical train-test-ratio for optimal accuracy

In general, the higher the number of training samples, better is the accuracy of the ML model up to a certain threshold, beyond which increasing the number of training samples does not improve the performance of the model [SSBD14]. However, generation of training data is a humongous task in current quantum devices since it takes up a significant amount of device lifetime. Therefore, lower the size of the training sample required, higher is its usability. But naively reducing the size of the training set may lead to performance degradation. We explore this requirement by studying the minimum train-test-ratio required to obtain the optimal decoder performance.

Given a distance d code, with t types of errors possible on each qubit, the total number of distinct error patterns is t^{d^2}. When the probability of error is very low, most of the test cases will have no error. This observation is reflected in the curve [Fig 5.6] with $p = 0.001$. As p increases, it is natural that the performance of

Table 5.1: Pseudo-threshold of the low and high level decoders for distance $d = 3$, 5 and 7 surface code

Noise Model →		Symmetric			Asymmetric					
		$p_z = p_x = p_y$			$0.1p_z = p_x = p_y$			$0.01p_z = p_x = p_y$		
Decoder ↓	d →	3	5	7	3	5	7	3	5	7
MWPM	LLD	0.0011	0.0038	0.0075	0.0012	0.0041	0.0072	0.00098	0.0038	0.0067
	HLD	-	-	-	-	-	-	-	-	-
Our FFNN	LLD	0.012	0.0205	0.0219	0.0109	0.0121	0.0152	0.0120	0.0122	0.0131
	HLD	0.0143	0.0234	0.0241	0.0124	0.0164	0.0189	0.0123	0.0165	0.0189
Our CNN	LLD	0.0121	0.0211	0.0228	0.0112	0.0125	0.0151	0.0111	0.0121	0.0132
	HLD	0.0152	0.0241	0.0247	0.0134	0.0161	0.0192	0.0121	0.0162	0.0195

Table 5.2: Comparison of training times for different ML models

ML Model		$d = 3$			$d = 5$		
		Parameter space	Training time (sec)	Prediction time (sec)	Parameter space	Training time (sec)	Prediction time (sec)
FFNN	Simple	2258	53.12	2.1×10^{-5}	5618	103.18	3.5×10^{-5}
	Complex	84754	324.9	3.55×10^{-5}	88114	394.99	3.72×10^{-5}
CNN	Simple	165650	785.27	5.27×10^{-5}	429874	1852.74	7.4×10^{-5}
	Complex	240246	1485.69	6.02×10^{-5}	504370	4241.58	9.74×10^{-5}

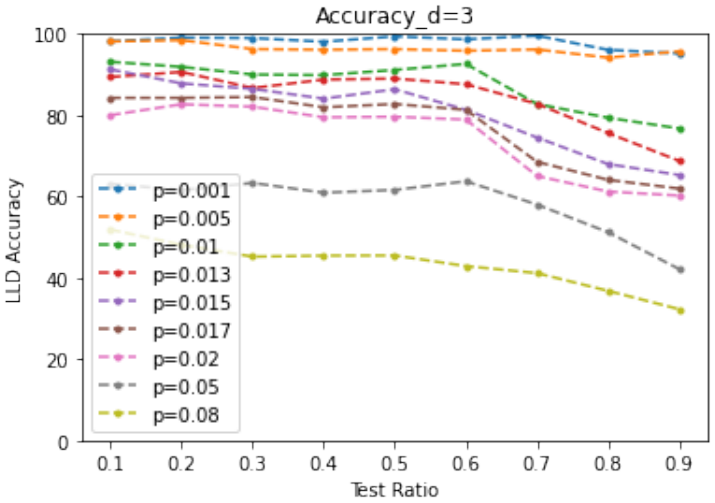
the decoder will degrade. However, when d is small, the total number of distinct errors is also small. Since the training and testing data is generated uniformly at random, we expect that for a given error probability p, the most likely error patterns are all exposed to the ML decoder for a reasonable size of training set.

To test this hypothesis, in Fig. 5.6 we have varied the train-test ratio for the simple CNN decoder for a distance 3 surface code. We originally generated 10^5 error data uniformly at random, and varied the train-test ratio starting from 90:10 and moving up to 10:90, lowering the training proportion by 10% in each step. The obtained accuracy for increasing p_{phys} is plotted in Fig. 5.6. We observe that even when we use up $\simeq 50 - 60\%$ of the data as test set, the performance of the decoder remains more or less constant for a given p. The performance takes a dip downwards beyond this value. Therefore, we posit that for $d = 3$ and $t = 4$

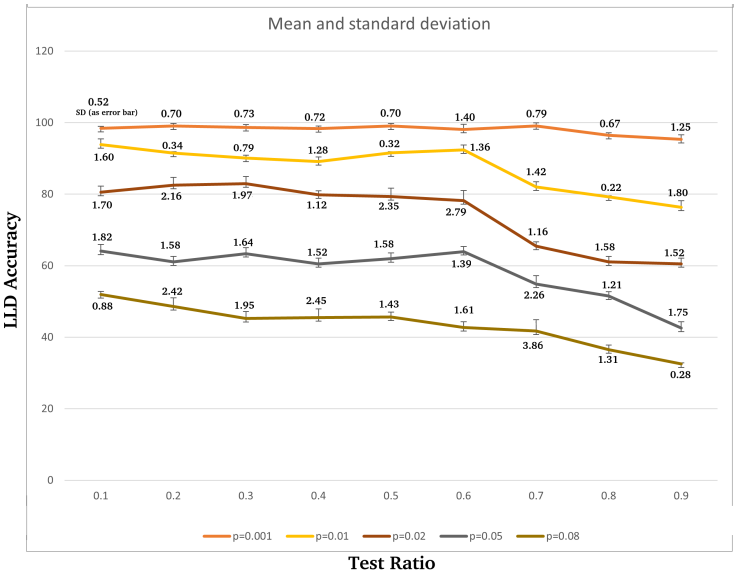
Table 5.3: Comparison of threshold of the low and high level decoders

Threshold (LLD)	Threshold (HLD)	Decoder Model	Error model
0.0181	N/A	MWPM	Symmetric
0.0302	0.035	Our FFNN	Symmetric
0.0218	0.025		Asymmetric $0.1 * p_z = p_x = p_y$
0.0221	0.0279		Asymmetric $0.07 * p_z = p_x = p_y$
0.0216	0.0257		Asymmetric $0.04 * p_z = p_x = p_y$
0.0213	0.0251		Asymmetric $0.01 * p_z = p_x = p_y$
0.0311	0.034	Our CNN	Symmetric
0.0225	0.026		Asymmetric $0.1 * p_z = p_x = p_y$
0.0229	0.0281		Asymmetric $0.07 * p_z = p_x = p_y$
0.0223	0.0258		Asymmetric $0.04 * p_z = p_x = p_y$
0.0212	0.0252		Asymmetric $0.01 * p_z = p_x = p_y$

(depolarizing noise model), this decoder is exposed to all of the most likely errors within a small fraction of the training set, which is generated uniformly at random.



(a) Average accuracy of our low level decoder



(b) Average Accuracy with its standard deviation

Figure 5.6: Average accuracy (along with its standard deviation) of our low level decoder vs Test Ratio for different values of p_{phys} in distance 3 surface code

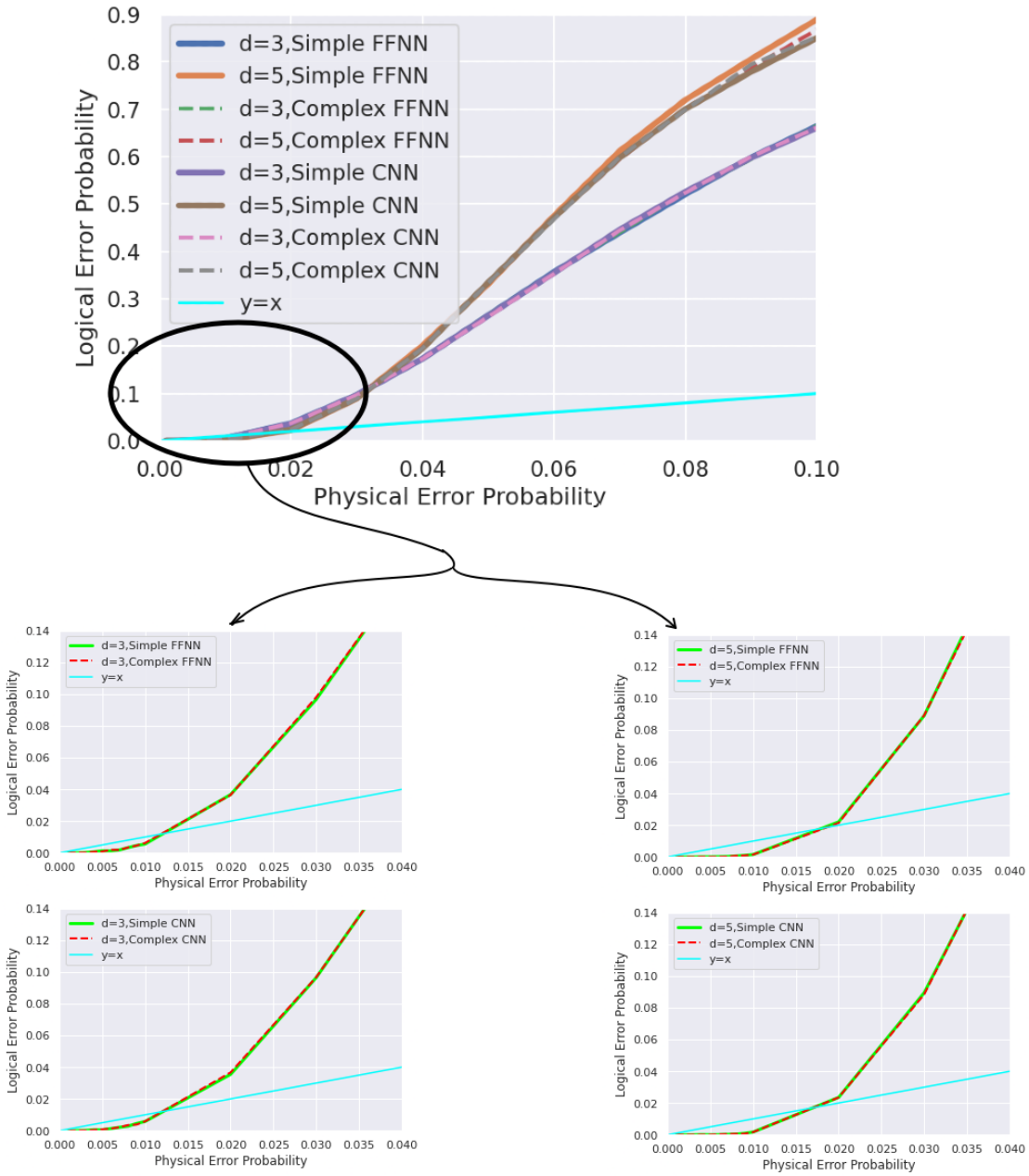


Figure 5.7: Logical vs physical error probability for various ML models in $d = 3$ and 5 surface code

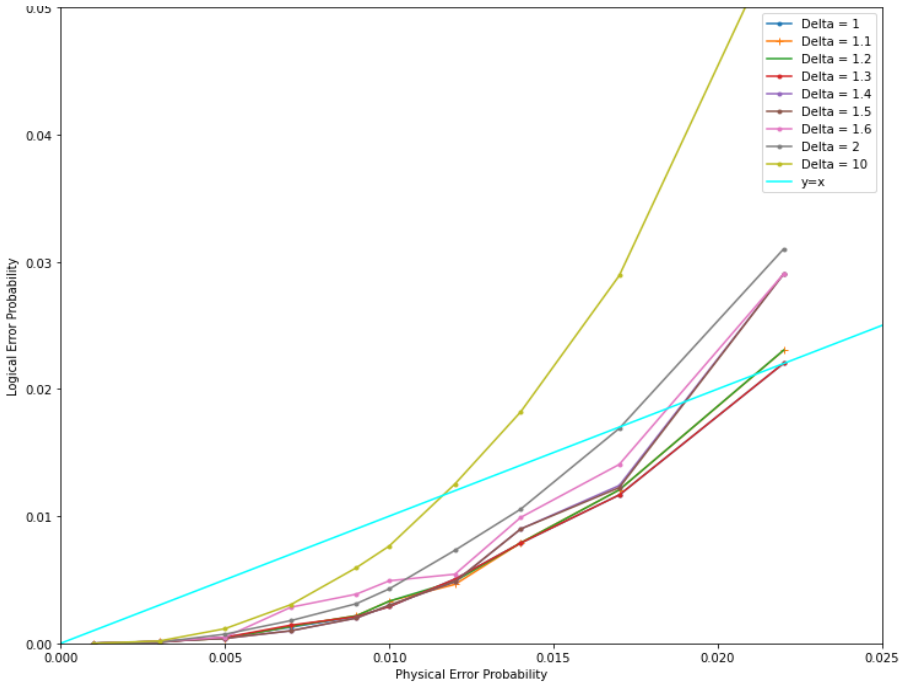


Figure 5.8: Training with symmetric noise Models and testing with asymmetric noise models

Since this is a ML based method, and Fig. 5.6(a) shows the mean value only, in Fig. 5.6(b) we have also plotted the standard deviation (SD) with a few values of physical error probability for all the test-ratio as an *error bar* plot. We observe that for p=0.001 the accuracy varies between 95.32 to 99.11 (min SD = 0.52, max SD = 1.40). For $p = 0.02$ the accuracy varies between 60.54 to 82.94 (min SD = 1.12, max SD = 2.79) and for $p = 0.08$ the accuracy varies between 32.51 to 51.92 (min SD = 0.28, max SD = 3.86). With increasing p_{phys}, the SD also increases. This supports intuition because as the p_{phys} increases, the decoding performance decreases due to the capacity of the machine learning model to correctly classify the errors. Hence, the performance of ML (which depends on the errors in the dataset), varies more with higher value of physical error probability (p_{phys}).

Moreover, as d increases, for same p, the set of probable errors increases exponentially. Therefore, for the same number of generated error data, we expect that to retain the same accuracy, a much larger portion of the data will need to be

devoted for training. In our future research, we shall explore this direction in a more extensive manner and determine the size of the training sample required to retain an accuracy ϵ for a given train-test ratio.

5.3.5 Performance on Training with Symmetric Noise Models and Testing with Asymmetric Noise Models

As we have discussed, the real life noise models are asymmetric. But this asymmetry can change and we may not know the exact level of asymmetry beforehand always. It would be beneficial if the decoder can be trained once with symmetric noise dataset and tested with different asymmetric noise datasets. Now we analyze how the performance of a decoder trained with symmetric noise model behaves while testing with asymmetric noise model with increase in asymmetry (Δ).

An increase in asymmetry (Δ) in the depolarizing error channel is denoted by $p_x = p/(\Delta+2)$, $p_y = p/(\Delta+2)$, $p_z = p\Delta/(\Delta+2)$, where, p is the value of physical error rate for a given physical error. For example, $\Delta = 10$ denotes $0.1 * p_z{=}p_x = p_y$. For symmetric noise model, $\Delta = 1$ and for asymmetric noise model, $\Delta > 1$. We define *crossover* point to be the value of Δ beyond which increasing Δ leads to lower pseudo-threshold, when the decoder is trained with symmetric noise. We say that the channel is weakly asymmetric if $\Delta <=$ cross-over point, strongly asymmetric otherwise. And we find the crossover point empirically.

As the asymmetry increases in the testing data, step-wise, we see a decrease in the performance of the decoder, as inferred from their decreased pseudo-threshold values. We check for $\Delta{=}1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2, 10$.

From Fig. 5.8 we can say that $\Delta = 1.3$ is the crossover point as upto this point, the logical error more or less overlaps with the symmetric noise model (i.e. $\Delta{=}1$). Hence upto $\Delta = 1.3$, the channel is weakly asymmetric and above $\Delta = 1.3$ it is strongly asymmetric. Hence, upto $\Delta = 1.3$ we can train the decoder with

symmetric noise model and test with the desired symmetric or asymmetric noise model, as it will not degrade the performance much. However, beyond this degree of asymmetry, the model must be trained with the asymmetric noise model if the optimum pseudo-threshold is to be obtained.

5.4 Conclusion

In this work, we have proposed an ML-decoder to correct both symmetric and asymmetric depolarizing noise on surface codes. Our decoder has two levels — in the low-level it tries to accurately predict the error on the qubits, followed by the high level that tries to detect any logical error that may have been introduced by the low-level decoder. Both these decoders have been implemented using neural network (FFNN and CNN) for surface code of distances 3, 5 and 7. Our proposed ML-decoder outperforms MWPM, and we observe $\sim 2\times$ increase in threshold and $\sim 10\times$ increase in pseudo threshold. We further show that the decoder performance is equally good for asymmetric errors as well, which is more realistic in quantum devices.

We have used ML models with different levels of sophistication, (i.e. varying number of hidden layers and node-density of each layer). Our results show that the mere increase of complexity in ML model requires an increased amount of time for decoding but hardly yields any better performance.

In this work, we have assumed, noise-free measure qubits and ideal measurements. A future prospect of this research can be to consider noisy measure qubits and imperfect measurements.

Extending our machine learning techniques, we next investigate efficient syndrome decoding for heavy hexagonal QECCs in the next chapter.

CHAPTER 6

Efficient Syndrome Decoder for Heavy hexagon QECC via Machine Learning

Contents

6.1 Introduction 121

6.2 Designing ML based decoder for heavy hexagon code . 122

6.3 Reducing error classes for heavy hexagon code 123

6.3.1 Reducing bit flip error classes by search based gauge equivalence 126

6.3.2 Reducing phase flip error classes by search based gauge equivalence 131

6.3.3 Reducing bit flip error classes by rank based gauge equivalence 134

6.4 Simulation Results 138

6.4.1 Machine Learning Parameters 139

6.4.2 Estimating Logical Error Rate with our Machine Learning based Syndrome Decoder 140

6.4.3 Comparison of ML-based decoder results with MWPM . 141
6.4.4 Scalability of our ML based decoder 148
6.4.5 Machine Learning based Decoding of Heavy hexagon QECC for Asymmetric Quantum Noise 150
6.5 Discussion 153

6.1 Introduction

Recently, industry research labs have been shifting towards the hexagon architecture for their quantum computers. This architecture has the advantage of reducing the number of distinct frequencies, and thus crosstalk. The surface code [BK98] structure has been modified to a topological code with a heavy hexagon structure [CZ+20] in order to become more suitable for these architectures. The heavy hexagon code [CZ+20] uses a combination of degree-two and degree-three vertices in the topology, and can be considered as a hybrid of a surface code and a Bacon-Shor code [Bac06]. This QECC reduces the distinct number of frequencies required in their realization by introducing more ancilla qubits (termed as flag qubits) for entanglement in the syndrome measurement [CZ+20].

This work initiates the study of using ML for decoding heavy hexagon code and to the best of our knowledge it is the first work to attempt ML decoding for heavy hexagon codes. The heavy hexagon code is a hybrid of surface code and Bacon-Shor code, where the later is a subsystem code [Bac06]. Being a subsystem code, the entire codespace of the heavy hexagon code is partitioned into equivalent classes (details given in Sec. 6.3). By this property, distinct errors can be clubbed into certain equivalent error classes, called gauge equivalence. We identify a unique representative element from each such class. This approach reduces the number of error classes, resulting in a classification problem with fewer classes, and thus the training of ML model becomes faster and more accurate.

For the depolarization noise model [NC10], we use the entire syndrome (i.e., the

syndromes for both X and Z stabilizers) to train the ML decoder even for determining the probability of logical X or Z errors individually. We show by simulation that the naïve ML decoder itself achieves a threshold of 0.0137 for logical X errors in bit flip noise, which is much higher than that for the MWPM decoder [CZ+20]. This is further improved to a threshold of 0.0158 using gauge equivalence. We also show that our ML decoder achieves a threshold of 0.0245 for logical X errors in depolarizing noise model, which is better than that for the MWPM decoder [CZ+20]. Similar improvements are observed for phase flip errors as well. In this work we propose the application of classical machine learning for quantum error decoding to facilitate error correction. Quantum machine learning has not been employed in the work presented here.

6.2 Designing ML based decoder for heavy hexagon code

Artificial neural networks (ANN) are brain-inspired techniques for replicating the procedure of how we humans learn and they are heavily used in machine learning. Neural networks consists of a single input and output layer along with a few hidden layers. The hidden layers transform the input into an intermediate form and the output layer finds patterns from its previous hidden layers. The time complexity to train a neural network with m input nodes, one hidden layer with h nodes and L output nodes is $\mathcal{O}((m + L) \cdot h)$. In this work we are using feed forward neural network for decoding heavy hexagon code of distance 3, 5, and 7 .

For application of ML in decoding, we first reduce the decoding problem to classification, a well-studied problem in machine learning. Classification is the process of predicting the class of given data points. Classes are also known as labels. It is the task of approximating a mapping function f from input variables x to output variables y. The methodology to map heavy hexagon code is similar with that for surface code which has already been discussed in detail in [BSM+22].

A distance d heavy hexagon code has (i) $(d^2-1)/2$ syndrome bits in case of bit flip error (refer to Z stabilizers in Fig. 2.12); (ii) $d-1$ syndrome bits in case of phase flip error (refer to X stabilizers in Fig. 2.12). This syndrome is the input data to the ML model. The label of the ML model is the erroneous data qubit of the heavy hexagon code structure, hence it has d^2 qubits in each entry. Our syndrome decoding problem is mapped into a *multi-class multi-label classification* problem in which there are 2^{d^2} classes, and each class label consists of d^2 bits. In our feed forward neural network, the input layer gets the syndrome (measured ancilla qubits) and the output layer identifies the type of error and its location in the lattice.

6.3 Reducing error classes for heavy hexagon code

The subsystem property of a QECC asserts that if $\Pi_j g_j$ denotes the product of one or more gauge operators, then

- an error $e = \Pi_j g_j$ can be safely ignored as the system transforms it to an equivalent subsystem;
- if for two errors e_1 and e_2, $e_1 \Pi_j g_j = e_2$, then e_2 can be considered as e_1 as both take the state to equivalent erroneous subsystems.

We term the second scenario as *gauge equivalence*, which provides significant advantage in designing ML based decoders. For both σ_x (bit flip) and σ_z (phase flip) errors, a distance d heavy hexagon code having d^2 qubits mandates a classification of the 2^{d^2} possible errors, each being termed as an *error class* henceforth. However, for a subsystem code such as the heavy hexagon code, there exists $i \neq j$ such that Q_i is gauge equivalent to Q_j. From now on, if $|\psi\rangle \equiv \Pi_j g_j |\phi\rangle$, where $\Pi_j g_j$ denotes the product of one or more gauge operators g_j, then we write $|\psi\rangle \equiv |\phi\rangle$ modulo $(\Pi_j g_j)$.

For example, in Fig 6.1 there are 4 X-gauge generators $G1, G2, G3, G4$, 6 Z-gauge generators $g1, g2, g3, g4, g5, g6$ and the qubits are $Q1, Q2, ..., Q9$. If $|\psi\rangle$ is the codeword, then,

- $X_4X_7X_8 |\psi\rangle \equiv X_5 |\psi\rangle$ modulo (G_3), where X_k denotes bit flip error on qubit Q_k.
- $Z_7 |\psi\rangle \equiv Z_1 |\psi\rangle$ modulo (g_4g_1), where Z_k denotes phase flip error on qubit Q_k.

The notion of gauge equivalent error strings creates a problem for machine learning based decoding since the problem of mapping syndromes to error strings is not well defined because there can be multiple Pauli error strings which are gauge equivalent having the same syndrome. In order to remedy the above, we identify a representative element from each such an error class. Every error string in the training data is mapped to the representative element of its corresponding class. Next, can we reduce the number of error classes? Reduction in the number of error classes implies a classification problem with fewer classes, which helps in improving the performance further for the ML model. We now formally define the criteria for two qubits to belong to the same error class.

Lemma 6.1

Given a codeword $|\psi\rangle$ such that $|\psi\rangle \equiv \Pi_j g_j |\psi\rangle$, where $\Pi_j g_j$ implies the product of one or more gauge operators g_j, any error e acting on the codeword is equivalent to $e(\Pi_j g_j)$.

Proof. Consider an error e acting on the codeword $|\psi\rangle$ such that $|\psi\rangle \equiv \Pi_j g_j |\psi\rangle$. Therefore,

$$e |\psi\rangle \equiv e\Pi_j g_j |\psi\rangle \Rightarrow e \equiv e(\Pi_j g_j)$$

□

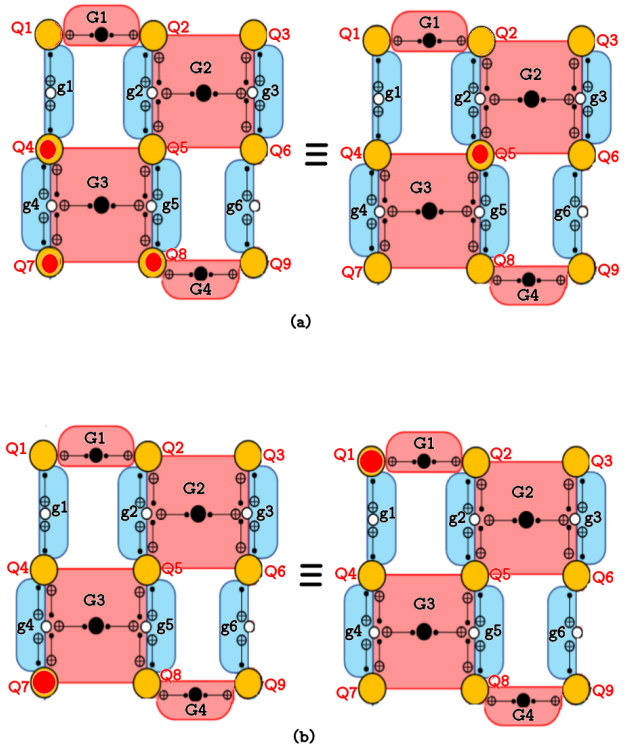


Figure 6.1: (a) X gauge equivalence for bit flip error: simultaneous errors on data qubits 4, 7 and 8 is equivalent to an error on data qubit 5, because by applying X gauge operator $G3$ (consisting of X operators on data qubits 4, 5, 7 and 8) on data qubits 4, 7 and 8, data qubit 5 has an error. (b) Z gauge equivalence for phase flip errors: an error on data qubit 7 is equivalent to an error on data qubit 1 because by applying Z gauge operator $g4$ (consisting of Z operators on qubits 4 and 7) followed by Z gauge operator $g1$ (consisting of Z operators on qubits 1 and 4) on data qubit 7, data qubit 1 has an error. This figure is adapted from [CZ+20]

Lemma 6.1 implies that e and $e(\Pi_j g_j)$ as the same error class. Such equivalence leads to a set of error classes whose cardinality is less than 2^{d^2}, where errors in the same class act equivalently on the codeword. Therefore, it is not necessary to distinguish between the errors in the same *error class*. It suffices to identify the error class only.

We propose two methods to find gauge equivalence and determine the error classes for the heavy hexagon code, namely *search based equivalence* and *rank based equivalence*.

6.3.1 Reducing bit flip error classes by search based gauge equivalence

The heavy hexagon code, being similar to surface code [BK98] and Bacon-Shor code [Bac06] for bit flip and phase flip errors respectively [CZ+20], we employ the subsystem code property to modify our training dataset into another equivalent training dataset with fewer class labels. In this subsection we present an algorithm to convert a given trining data set to an equivalent one having fewer class labels for bit flip errors.

Before presenting the Algorithm, we prove in Lemma 6.2 the number of error classes obtained due to the subsystem property. As multiple errors can fall under the same class, we require a class representative. We show a method to find the class representative, followed finally by the Algorithm to obtain such equivalence.

Lemma 6.2

For a distance d heavy hexagon code, the total number of bit flip error classes is $2^{\frac{d^2+1}{2}}$.

Proof. For a distance d heavy hexagon code, the total number of data qubits is d^2 and the number of X gauge generators is $\frac{d^2-1}{2}$. Therefore, the total possible combinations of these can provide $2^{\frac{d^2-1}{2}}$ X gauge operators. For each of these

operators g, any error $e \equiv e.g$. Therefore, e and $e.g$ belong to the same error class. The total number of error classes is $2^{d^2}/2^{\frac{d^2-1}{2}} = 2^{\frac{d^2+1}{2}}$.

□

We thus obtain a square root order reduction in the number of error classes. Table 6.1 presents the reduction in the number of error classes for codes of distance 3, 5 and 7 due to gauge equivalence. As multiple errors belong to the same error class, a *class representative* needs to be chosen efficiently for our ML decoder.

Table 6.1: Number of bit flip error class labels without and with gauge equivalence

Code distance	# Class labels without equivalence	# Class labels with equivalence
3	2^9	2^5
5	2^{25}	2^{13}
7	2^{49}	2^{25}

Choice of class representative: If errors $e_1, e_2, \ldots, e_k$ belong to the same error class $\mathcal{E}$, then we choose a class representative $e_i \in \mathcal{E}$ which is to be considered as the error on the codeword for any error $e \in \mathcal{E}$ occurring on the codeword. For example, in Fig. 6.1 it can be verified that bit flip errors on qubits 4, 7 and 8 produces the same syndrome as that on 5 . But for training the ML decoder, we choose the second one as the common label for both as it has a lower weight. However, simply considering the weight of the errors is not sufficient to obtain a class representative. For example, bit flip errors on qubits 1 and 2 lead to the same syndrome, and both of these errors have weight 1. Therefore, we use the method of *lexicographic minima* to obtain the class representative.

For finding the lexicographic minima, we represent the qubits in the codeword as a characteristic vector of size d^2, where the i^{th} bit is 1 if an error occurred on qubit i, and 0 otherwise. For example, the bit flip errors on qubits 4, 7 and 8 in Fig. 6.1 is denoted as $e = [000100110]$. For each error class $\mathcal{E}$, we choose error e as the class representative for which $L(e) = \sum_{i=0}^{d^2-1} 2^i * e[i]$ is minimum. For example, in a distance 3 heavy hexagon code, bit flip error on qubits 1 and 2 are written as $e_1 = [100000000]$ and $e_2 = [010000000]$ respectively. We note that X-operator on

qubits 1 and 2 form an X gauge operator, which when applied on e_1 gives us e_2, and vice-versa. So e_1 and e_2 belong to the same error equivalence class $\mathcal{E}$. Here, $L(e_1) = 2^1 * 1 + 2^2 * 0 + 2^3 * 0 + ... + 2^9 * 0 = 2$ and $L(e_2) = 4$, so e_1 is chosen as the class representative.

Algorithm 5 finds the gauge equivalent errors for each error e in the training dataset $E[1:N]$. Its time and space complexity are given in Lemmata 6.3 and 6.4.

Algorithm 5 Search based method to find gauge equivalence class representatives for bit flip errors

Input: G_x, the list of X-gauge operators; d, distance of the heavy hexagon code; $E[1:N]$, the list of all errors $e_1, e_2, ..., e_N$ in the training dataset.

Output: for all errors in $E[1:N]$, equivalent error vector $MinWeightEquiv$ having weight $MinWeight$

for $e = e_1$ to e_N **do**
  $MinWeightEquiv = e$
  $MinWeight = \sum_{i=0}^{d^2-1} 2^i * e[i]$
  for all $g_x \in G_x$ **do**
    $GaugeEquiv = g_x \oplus e$ applying g_x on e^*
    $weight_{GE} = \sum_{i=0}^{d^2-1} 2^i * GaugeEquiv[i]$
    if $MinWeight > weight_{GE}$ **then**
      $MinWeight = weight_{GE}$
      $MinWeightEquiv = GaugeEquiv$
    end if
  end for
  return $MinWeightEquiv$
end for

Theorem 6.1

Given a training sample e in which each qubit may be either error-free or has a bit flip error only, its corresponding gauge equivalent class representative $e_{min} \equiv e$ can be computed by Algorithm 5 in $\mathcal{O}(N \cdot d^2 \cdot 2^{d^2/2})$ time using $\mathcal{O}(d^2 \cdot 2^{\frac{d^2-1}{2}} + N.d^2)$ space, where d is the distance of the code and N is the number of training samples.

Proof. For an error e, let $MinWeightEquiv \equiv e$ be the lexicographic minimum in $E[1 : N]$ which is gauge equivalent to e. The gauge operators and errors are represented as $\{0, 1\}$ strings. Let $L(e)$ denote the lexicographic value corresponding to the binary characteristic vectors representation of e. Therefore, $L(MinWeightEquiv) \leq L(e_{equiv}) \ \forall \ e_{equiv} \equiv e$. Algorithm 5 finds $L(e_{equiv})$ in decimal for all $e_{equiv} \equiv e$, and returns the vector $MinWeightEquiv$ which has the minimum value. Hence, $L(MinWeightEquiv) \leq L(e_j) \ \forall \ e_j \equiv e$. Therefore, $MinWeightEquiv = e_{min}$.

The proofs for time and space complexity are given below in Lemmata 6.3 and 6.4 respectively. □

Lemma 6.3

Given a training sample e in which each qubit may be either error-free or have bit flip error only, Algorithm 5 computes its corresponding gauge equivalent class representative $e_{min} \equiv e$ in $\mathcal{O}(N \cdot d^2 \cdot 2^{d^2/2})$ time, where d is the distance of the code and N is the number of training samples.

Proof. For a distance d heavy hexagon code, the total number of data qubits is d^2 and the number of X gauge generators is $\frac{d^2-1}{2}$. Therefore, the total possible combinations of the X gauge operators is $2^{\frac{d^2-1}{2}}$.Each of the line numbers 2, 5, and 6 of Algorithm 5 requires d^2 bitwise operation over the length of an error string.

There are $2^{\frac{d^2-1}{2}}$ possible gauge operators. Therefore the inner loop (line 4 to 11) runs $2^{\frac{d^2-1}{2}}$ times. So the time for each training sample is $d^2 + d^2 \cdot 2^{\frac{d^2-1}{2}} + c$, where c

is a constant, leading to a time complexity of $\mathcal{O}(d^2 \cdot 2^{d^2/2})$ for the inner *for* loop of Algorithm 5. Therefore, the total time complexity of Algorithm 5 to find the gauge equivalence for the entire training dataset consisting of N samples is $\mathcal{O}(N \cdot d^2 \cdot 2^{d^2})$.

□

Lemma 6.4

Given a training sample e in which each qubit may be either error-free or has a bit flip error only, Algorithm 5 computes its corresponding gauge equivalent class representative $e_{min} \equiv e$ with space complexity of $\mathcal{O}(d^2 \cdot 2^{\frac{d^2-1}{2}} + N.d^2)$, where d is the distance of the code and N is the number of training samples.

Proof. Each of the $2^{\frac{d^2-1}{2}}$ possible X gauge operators need d^2 bits. Hence the total space required to store all the gauge operators is $\mathcal{O}(d^2 \cdot 2^{\frac{d^2-1}{2}})$. Furthermore, the entire training dataset of N samples needs space complexity of Algorithm 5 is $\mathcal{O}(d^2 \cdot 2^{\frac{d^2-1}{2}} + N.d^2)$. For a distance d heavy hexagon code, there are d^2 data qubits in the lattice for a logical qubit and $\frac{d^2-1}{2}$ X gauge generators. □

Table 6.2: Number of error class labels without and with gauge equivalence in case of phase flip

Code distance	# Class labels without equivalence	# Class labels with equivalence
3	2^9	2^3
5	2^{25}	2^5
7	2^{49}	2^7

In the expression of space complexity $\mathcal{O}(d^2 \cdot 2^{\frac{d^2-1}{2}} + N.d^2)$, the second term $(N.d^2)$ dominates when $N > 2^{\frac{d^2-1}{2}}$.

Instead of calculating all possible combinations of X gauge operators beforehand, if we calculate them on the go, then the space complexity can be reduced to $\mathcal{O}(d^4 + N.d^2)$. However, for such an algorithm, the entire set of gauge operators need to be generated for every training sample, leading to a time complexity of upto $\mathcal{O}(N \cdot d^4 \cdot 2^{d^2/2})$.

6.3.2 Reducing phase flip error classes by search based gauge equivalence

In this subsection, we present studies for phase flip error similar to those for bit flip error in the previous subsection. The Bacon-Shor structure [Bac06] of phase stabilizers allow for more equivalent subsystems, leading to fewer error classes. We first depict this equivalence for phase flip errors in Lemma 6.5, and provide the algorithm to find the class representatives for gauge equivalent phase flip error as Algorithm 6.

Lemma 6.5

For any column $1 \leq j \leq d$ of physical qubits in the lattice for heavy hexagon QECC, the phase operator $Z_{i,j}$ on i^{th} qubit of the j^{th} column is gauge equivalent to the phase operator $Z_{1,j}$.

Proof. (by induction)
Base Case: If a phase flip error occurs on the first qubit of any column of qubits j, then it is trivially equivalent to $Z_{1,j}$.

Induction Hypothesis: For all positive integer k, $1 \leq k < d$, $Z_{k,j}$ is gauge equivalent to $Z_{1,j}$.

Induction Step: Consider a phase operator $Z_{k+1,j}$. Now, for all k, $1 \leq k < d$, there exists a Z gauge operator given by $Z_{k,j} \cdot Z_{k+1,j}$. Therefore, $Z_{k+1,j}$ is gauge equivalent to $(Z_{k+1,j}) \cdot (Z_{k,j} Z_{k+1,j}) = Z_{k,j}$. By the induction hypothesis, as $Z_{k,j}$ is gauge equivalent to $Z_{1,j}$, $Z_{k+1,j}$ is gauge equivalent to $Z_{1,j}$.

□

Corollary 6.1

There are 2^d non-equivalent error classes for phase flip errors.

Proof. This follows from Lemma 6.5 because q_j^i is gauge equivalent to q_j^1 for the j^{th}qubit column, $1 \leq i \leq d$, each qubit column corresponds to one possible phase flip error class. There being d qubit columns, there are 2^d non-equivalent phase flip error classes.

□

Algorithm 6 below finds the gauge equivalence class representatives for each error e in the training dataset. Its time and space complexity analyses appear in Lemmata 6.6 and 6.7.

Algorithm 6 Search based method to find gauge equivalence class representatives for phase flip errors

Input: d, distance of the code, error string $E[1 : N]$, the list of all errors $e_1, e_2, ..., e_N$ in the training dataset.

Output: for all errors in $E[1 : N]$, equivalent error vector $MinWeightEquiv$ having weight $MinWeight$

1: **for** $e = e_1$ to e_N **do**
2: **for** $j = d$ to $d^2 - 1$ **do**
3: $l = j\%d$
4: $e[l] = e[l] \oplus e[j]$
5: **end for**
6: $MinWeightEquiv = e$
7: return $MinWeightEquiv$
8: **end for**

Theorem 6.2

Given a training sample e in which each qubit may be either error free or having phase flip error only, its corresponding gauge equivalent minima $e_{min} \equiv e$ can be computed by Algorithm 6 in $\mathcal{O}(N.d^2)$ time using $\mathcal{O}(N.d^2)$ space, where d is the distance of the code and N is the number of training samples.

Proof. For an error e, let e_{min} be the lexicographic minimum corresponding to e. For any column $1 \leq j \leq d$ of qubits where $Z_{i,j}$ denotes the phase operator on i^{th} qubit of the j^{th} column (where $1 \leq i \leq d$), $Z_{i,j}$ is gauge equivalent to $Z_{1,j}$. For an error $e = \{e^1 e^2 \dots e^k\}$, where $e^i \in \{I, Z\}$, $1 \leq i \leq k$ (k is the number of qubits), Algorithm 6 finds the gauge equivalence e^i_{equiv} of each e^i using Lemma 6.5. The final gauge equivalent error e_{equiv} is determined as $e_{equiv} = \{e^1_{equiv} \cdot e^2_{equiv} \dots e^k_{equiv}\}$. As it finds the equivalent error as the top most row's error in the lattice, hence it will be of the minimum weight. Hence $e_{equiv} = e_{min}$ The proofs for time and space complexity are given below in Lemmata 6.6 and 6.7 respectively.

□

Lemma 6.6

Given a phase flip error e, its corresponding gauge equivalent minima $e_{min} \equiv e$ can be computed by Algorithm 6 in $\mathcal{O}(N.d^2)$ time, where d is the distance of the code.

Proof. Lemma 6.5 asserts that for each column of the heavy hexagon code lattice, a phase flip error on a qubit in that column is gauge equivalent to the phase flip error on the top most qubit of that column. Therefore, Algorithm 6 starts from the second row of the lattice, leaving aside the topmost row (length d) which consists of the first qubits of each column. Each of line numbers 3 and 4 of Algorithm 6 requires a constant time operation for $d^2 - d$ error strings. The outer loop, i.e. line number 1 runs N times, as the size of the entire training dataset is N. Therefore, the time complexity of Algorithm 6 is $N \cdot c \cdot (d^2 - d)$, where c is a constant, i.e., $\mathcal{O}(N.d^2)$.

□

It is to be noted that we obtain a quadratic reduction in the number of error classes for both bit flip and phase flip errors by employing gauge equivalence but the reduction is much higher for phase flip errors.

Lemma 6.7

Given a phase flip error e, its corresponding gauge equivalent minima $e_{min} \equiv e$, computed by Algorithm 6 has a space complexity of $\mathcal{O}(N.d^2)$ where d is the distance of the code, and N number of training samples.

Proof. In order to store N number of training samples we need $\mathcal{O}(N.d^2)$ amount of space. Therefore, the total space needed for Algorithm 6 is $\mathcal{O}(N.d^2)$. □

We observe that based on linear search the time complexity for finding equivalence class representatives of bit flip errors is $\mathcal{O}(N.d^2 \cdot 2^{d^2})$ and for phase flip errors is $\mathcal{O}(N.d^2)$. In the next subsection, we propose a method called rank based gauge equivalence which can further reduce the time complexity for the case of bit flip errors, which is much higher than that for phase flip errors.

6.3.3 Reducing bit flip error classes by rank based gauge equivalence

This algorithm relies on finding the rank of a matrix $M = [GEN_x]$, constructed with a column corresponding to each of the X gauge generators. Recall that the rank of a matrix denotes the number of linearly independent columns in it [Str06]. Since the gauge generators are linearly independent, M exhibits full rank equal to $(d^2 - 1)/2$, the number of X gauge generators.

For two gauge equivalent errors e_i and e_k, if we obtain another matrix $M' = [GEN_x, e_i - e_k]$ by appending one more column to M, then M and M' has the same rank because e_k can be expressed as $(\Pi_j g_j)e_i$. We use this notion to find the gauge equivalence for bit flip errors faster than Algorithm 5.

In Algorithm 7, we provide an algorithm to find the equivalent for each error e in the training dataset, and depict its time and space complexities in Lemmata 6.9 and 6.10 respectively.

The search space of Algorithm 7 differs from that of Algorithm 5. Let $EQUIV(e)$ denote the set of all gauge equivalent errors corresponding to an error e. The cardinality of $EQUIV(e)$ scales exponentially in the number of gauge *generators*. On the other hand, Algorithm 7 determines $EQUIV_{TS}(e)$ which is the set of all gauge equivalent errors corresponding to an error e present in the set of training samples. Therefore, $EQUIV_{TS}(e) \subseteq EQUIV(e)$. Since Algorithm 7 only searches for the gauge equivalent errors present in the training samples, the search space for this algorithm is never greater than that for Algorithm 5. Both Algorithms 5 and 7 reduce the error classes for bit flip using gauge equivalence. While in Algorithm 5 the class label for the equivalence classes is of minimum possible weight, in Algorithm 7 the class label is the minimum possible weight for errors present in the training set. But our target is to reduce the number of error classes. So both the algorithms serve that purpose. It is to be noted that the equivalence error classes remain the same for both methods. Only the class representatives are likely to differ.

Theorem 6.3

Given a training sample e in which each qubit may be error free or has bit flip error only, its corresponding gauge equivalent minima $e_{min} \equiv e$ can be computed, by Algorithm 7 in $\mathcal{O}(N^2 \cdot d^4)$ time using $\mathcal{O}(d^4 + N.d^2)$ space, where d is the distance of the code and N is the number of training samples.

Proof. We prove this theorem in two steps. The proofs for time and space complexity are shown individually in Lemmata 6.8 and 6.9 respectively.

□

Lemma 6.8

Given a bit flip error e, its corresponding gauge equivalent minima $e_{min} \equiv e$ can be computed by Algorithm 7 in $\mathcal{O}(N^2 \cdot d^4)$ where d is the distance of the code and entire dataset consists of N errors.

Algorithm 7 Rank based method to find gauge equivalence class representatives for bit flip errors

Input: GEN_x, the list of X gauge generators, d, distance of the code, List of all errors in the dataset $E[1:j]$ which consists $e_1, e_2, ..., e_j$.

Output: Error string $MinWeightEquiv$ which is equivalent to e_j.

1: $MinWeightEquiv = e_j$
2: $MinWeight = \sum_{k \in d^2} 2^k * e_j[k]$
3: $M = [GEN_x]$ where each GEN_x, $0 \leq x \leq (d^2-1)/2$ forms a column of M.
4: Calculate the rank M
5: **for all** $e = e_1$ to e_{j-1} **do**
6: **if** $syndrome(e_j) == syndrome(e)$ **then**
7: Form a matrix M' by appending $e_j - e$ as the last column of M
8: Calculate the rank of M'
9: **if** $Rank(M) == Rank(M')$ **then**
10: $weight_{GE} = \sum_{k \in d^2} 2^k * e[k]$
11: **if** $MinWeight > weight_{GE}$ **then**
12: $MinWeight = weight_{GE}$
13: $MinWeightEquiv = e$
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: Return $MinWeightEquiv$

Proof. In Algorithm 7, each of the lines 2 and 10 requires bitwise operation over the length of the error string which is d^2. Computing the syndrome equality in line 6 requires bitwise operation over the length of the syndrome string which is $(d^2-1)/2$. There are d^2 number of data qubits and the number of gauge generators is $(d^2-1)/2$. Furthermore, calculation of rank of an $m \times n$ matrix is $\mathcal{O}(m \cdot n)$ [CLRS22]. Therefore, calculating the rank of matrix M of size $(d^2 \times (d^2-1)/2))$ in line 4, has a running time of $\mathcal{O}(d^4)$. For a particular error e_j, the time required to calculate the gauge equivalence is given by $\sum_{i=1}^{j-1} \mathcal{O}(d^4+d^2) = \mathcal{O}((d^4+d^2) \cdot j)$. Hence, for the entire dataset consisting of N errors, the time complexity of Algo-

rithm 7 is

$$\begin{aligned}\mathcal{O}((d^4+d^2)\cdot\sum_{j=1}^{N} j) &= \mathcal{O}(N^2\cdot(d^4+d^2)) \\ &= \mathcal{O}(N^2\cdot d^4)\end{aligned}$$

□

Lemma 6.9

Given a bit flip error e, Algorithm 7 computes its corresponding gauge equivalent minima $e_{min} \equiv e$ using $\mathcal{O}(d^4 + N.d^2)$ space where d is the distance of the code and N is the number of training samples.

Proof. For a distance d heavy hexagon code, the total number of data qubits is d^2 and the number of X gauge generators is $\frac{d^2-1}{2}$. Each of the gauge generators consists of d^2 bits. Hence, to store the generator matrix M, the space needed is $d^2 \cdot (d^2-1)/2$. To store the appended matrix M', $d^2 \cdot ((d^2-1)/2+1)$ space is needed and to store N number of training samples, $N \cdot d^2$ space is needed. Hence, the total amount of space needed is $d^2 \cdot (d^2-1)/2 + d^2 \cdot ((d^2-1)/2+1) + N \cdot d^2$ leading to a space complexity of $\mathcal{O}(d^4 + N.d^2)$.

□

A pertinent question, therefore, is the criterion for which rank based equivalence is faster than the search based one. This is answered in Corollary 6.2

Corollary 6.2

Rank based equivalence method finds gauge equivalence faster than the search based method if $2^{d^2} \geq c.(N \cdot d^2)$, where d is the distance of the code, N the number of training samples, and c a constant.

Proof. According to Lemma 6.3, given a bit flip error e, its corresponding gauge equivalent minima $e_{min} \equiv e$ can be computed, according to Algorithm 5, in $\mathcal{O}(d^2 \cdot 2^{d^2})$, where d is the distance of the code. Hence the total time required to find the gauge equivalence for the entire training dataset consisting of N samples is $\mathcal{O}(N \cdot d^2 \cdot 2^{d^2})$. According to Lemma 2.8 given a bit flip error e, its corresponding rank based gauge equivalent minima $e_{min} \equiv e$ can be computed by Algorithm 7 in $\mathcal{O}(N^2 \cdot d^4)$ where d is the distance of the code and entire dataset consists of N errors. Hence, it is gainful to employ rank based gauge equivalence only when

$$\begin{aligned} \mathcal{O}(N \cdot d^2 \cdot 2^{d^2}) &\geq \mathcal{O}(N^2 \cdot d^4) \\ \Rightarrow c_1 \cdot N \cdot d^2 \cdot 2^{d^2} &\geq c_2 N^2 \cdot d^4 \\ \Rightarrow 2^{d^2} &\geq c(N \cdot d^2) \end{aligned}$$

where $c = \frac{c_2}{c_1}$ is a constant. □

We have empirically tested that for $d > 3$, the rank based equivalence method is faster than its search based counterpart for bit flip error.

The two algorithms based on linear search and rank to determine gauge equivalence class representatives, are applicable for both bit flip and phase flip errors for any gauge code. However, for the heavy hexagon code, we exploited the gauge structure to design a better algorithm to find gauge equivalence for phase flip errors. Similarly, structure of other codes may be exploited to design better algorithms to find gauge equivalence for bit and/or phase flip errors.

6.4 Simulation Results

We have implemented and tested our ML based decoders for heavy hexagon code [CZ+20] with bit flip, phase flip, and depolarization noise models. The machine learning parameters used for our decoder are given in Section 1.4.1

To the best of our knowledge, only MWPM based decoders have been studied for the heavy hexagon QECC. Hence we have compared the metrics of our ML based decoder with those for MWPM decoder. We have used the stabiliser structure of the MWPM decoder proposed in [CZ+20], and implemented it using the 'PyMatching' library [Hig22]. The threshold obtained by this implementation of MWPM decoder is the same as that in [CZ+20] (vide Fig. 6.6). Although the original paper on heavy hexagon code used only MWPM decoder, for the sake of completeness, we have shown the threshold and pseudo-threshold values for Union Find decoder as well. We have used the Plaquette library [pla] for the implementation of Union Find decoder.

We have executed our codes for our ML decoders along with our search and rank based algorithms to find the gauge equivalence representatives, on a server of National Energy Research Scientific Computing (NERSC) which has the computing resource Perlmutter, a Cray EX system with AMD EPYC CPUs and NVIDIA A100 GPUs.

6.4.1 Machine Learning Parameters

Our ML model has been trained in batches of data instead of training it with the entire data set (of size 10^6) at once. This is advantageous in terms of both training time and memory capacity. For a distance d heavy hexagon code, there are d^2 data qubits and $d^2 - 1$ measure qubits (involving stabilizers and flag qubits). The data set is generated as follows. For each qubit (both data and measure), we apply an error with probability p_{phys}. If the noise model is bit (phase) flip, then a X (Z) operator is applied on the particular qubit with probability p_{phys}. For depolarization noise model, the error operator is a linear combination of X, Y or Z, each with probability $\frac{p_{phys}}{3}$. Finally, once the syndrome of the qubit error is generated (which is also noisy since the measure qubits may be erroneous as well), we apply measurement error by changing each bit of the syndrome with probability p_{meas}. According to [FWH12, CZ+20], we assume $p_{meas} = p_{phys}$. Under the assumption that errors are not correlated, this model captures a majority of

the noise in real devices and has also been considered in prior research [FWH12, CZ+20]. Thus our dataset is a subset of all possible errors, which is likely to be a good representative of the actual noise in the hardware since it is generated in a manner similar to the assumption for the errors to occur in the quantum hardware [FWH12, CZ+20].

The dataset size that we have used is 100000 from which 70000 is used for training and the rest for testing purpose. For evaluating the model we have used 10 fold cross validation. This approach involves randomly dividing the set of observations into 10 groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining 9 folds.

The batch size of our simulation is 10000, run for 1000 epochs, with a learning rate of 0.01 (using stochastic gradient descent). The results have been averaged over five trials. This method is repeated for each value of the physical error probability (p_{phys}) considered in this study.

Our FFNN has four layers, namely an input layer, an output layer and two hidden layers. For bit flip errors, in the input layer there are $(d^2-1)/2$ nodes which is the size of the syndrome (equals the number of Z stabilizers). For phase flip errors, in the input layer there are d nodes which is the size of the syndrome (equals the number of X stabilizers). The number of nodes in the output layer is always d^2, the number of data qubits for distance d of the heavy hexagon code. The loss function is chosen as binary cross-entropy. We have used the Keras library for creating the FFNN. Table 6.3 includes the details of our FFNN.

6.4.2 Estimating Logical Error Rate with our Machine Learning based Syndrome Decoder

The input of the FFNN based decoder is the syndrome or the list of ancilla measurements. The output of the decoder is the list of the indices of the data qubits for which error is detected. Suppose the size of the testing data is N_t. For each

Table 6.3: Number of nodes in the layers of our FFNN based heavy hexagon QECC syndrome decoder for bit flip and phase flip errors

Layer	Code distance d	# nodes for bit flip errors	# nodes for phase flip errors
Input	3	4	3
	5	12	5
	7	24	7
Output	3	9	9
	5	25	25
	7	49	49
First hidden	3	16	16
	5	32	32
	7	64	64
Second hidden	3	16	16
	5	32	32
	7	64	64

testing data (S_i, e_i), where S_i is the syndrome for the error e_i, the FFNN decoder is provided with the syndrome and it suggests a probable error $e_{i_{FFNN}}$. An error being a Pauli operator, is self-adjoint, and its correction is by simply applying the same operator once more. Therefore, the effective error on the system after performing correction based on the suggestion of the FFNN decoder is $e_{i_{FFNN}}.e_i$. Note that when $e_{i_{FFNN}} = e_i$ it implies perfect correction. We can verify easily whether $e_{i_{FFNN}}.e_i$ is a logical error. We want to emphasize here that the decoder may predict error incorrectly but the result may still not be a logical error, i.e., $e_{i_{FFNN}} \neq e_i$, whereas $e_{i_{FFNN}}.e_i$ is not a logical error. In accordance with [CZ+20], we report here only the probability of logical errors identified as above. If this count of logical errors is $\nu_{logical}$, then our estimate for the logical rate or error probability $p_{logical}$ is reported as $\nu_{logical}/N_t$.

6.4.3 Comparison of ML-based decoder results with MWPM

Bit flip error on data qubit

First, we show the decoding performance of our FFNN-based decoder for heavy hexagon codes of distance 3, 5, and 7 for bit flip noise model on data qubit (assuming ideal stabilizers and measurements).

Table 6.4 shows the values of threshold and pseudo-threshold for UF, MWPM and our proposed ML decoders for heavy hexagon QECC with distance 3, 5, and 7. We note that the ML decoder clearly outperforms MWPM, which in its turn, outperforms UF. Therefore, henceforth, all comparisons of our ML decoder is performed with MWPM.

Table 6.4: Comparison of UF, MWPM and FFNN based decoder result in case of bit flip error (assuming ideal stabilizers and measurements)

Decoding method	Code distance d	Threshold	Pseudo Threshold
UF	3	0.0038	0.0001
	5		0.0008
	7		0.0018
MWPM	3	0.0042	0.0002
	5		0.0012
	7		0.0024
FFNN	3	0.015	0.006
	5		0.0086
	7		0.0115

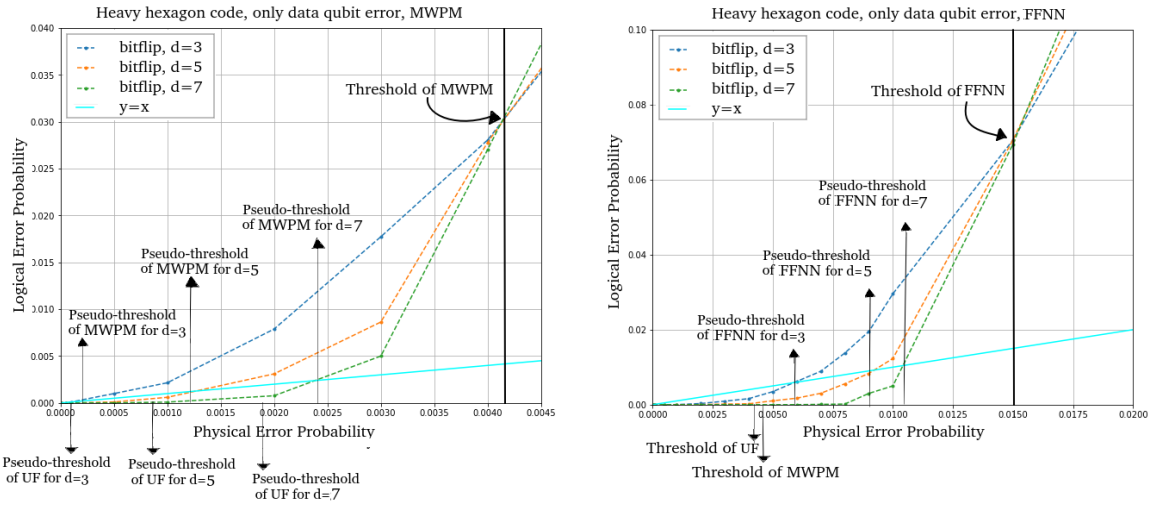


Figure 6.2: Threshold and pseudo-threshold values for MWPM and ML based decoders in case of distance 3, 5, and 7 heavy hexagon code for bit flip error on data qubit

Fig. 6.2 shows the change in the probability of logical error with physical error probability p, which is the per-step error probability in the heavy hexagon code cycle, when the error occurs only on data qubits. The results of MWPM and FFNN-based decoder for bit flip noise models on data qubits are shown. In the

figure, the blue, orange, and green lines respectively are the plots for distances 3, 5, and 7. The points where the blue, orange and green line intersect the cyan straight line indicate the pseudo-threshold for distances 3, 5 and 7 respectively.

Naturally, with increase in the distance of the heavy hexagon code, the pseudo threshold improves. However, threshold is a property of the error correcting code and the noise model only. It is independent of the distance. In Fig. 6, we observe that our ML based decoders are working much better than MWPM decoders for heavy hexagon QECC in terms of threshold.

Performance of FFNN based decoder result without and with gauge equivalence for bit flip error on data qubit along with measurement and stabilizer error

We show the decoding performance of our FFNN-based decoder for heavy hexagon codes of distance 3, 5, and 7 for bit flip noise model on data qubit along with measurement, and stabilizer error for three cases, i.e. without gauge equivalence, with search-based gauge equivalence, and with rank-based gauge equivalence. Our FFNN based model outperforms the existing MWPM decoder. We also show that the performance of FFNN with gauge equivalence is better than that of FFNN without gauge equivalence, and also the performance of search based and rank based gauge equivalence is more or less equal. We then present the comparison of decoding time (in seconds) needed for distance 3, 5 and 7 heavy hexagon code in case of FFNN based decoder for three cases, namely without gauge equivalence, with search based gauge equivalence and with rank based gauge equivalence. This supports the fact that rank based gauge equivalence method is faster than search based gauge equivalence (Fig 6.4). Although the time needed for the basic FFNN decoder is much less than that for the search based gauge or rank based gauge equivalence methods but the decoding performance is poorer as already mentioned above.

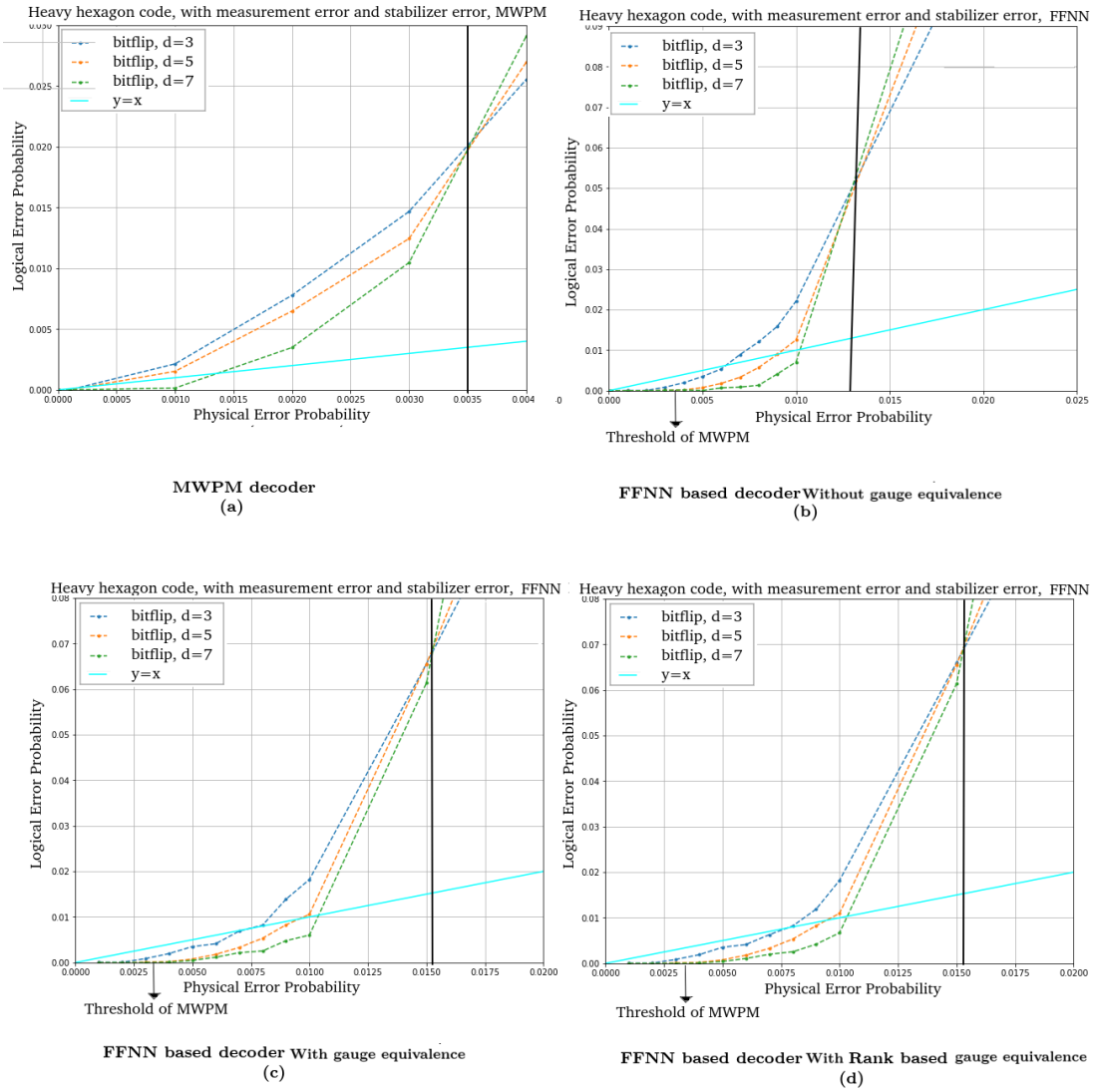


Figure 6.3: Threshold and pseudo-threshold values for MWPM and ML based decoders (without and with gauge equivalence) and in case of distance 3, 5, and 7 heavy hexagon code for bit flip error on data qubit along with measurement, and stabilizer error

Fig. 6.3 shows the variation in the logical error probability with physical error probability p with and without the application of gauge equivalence.

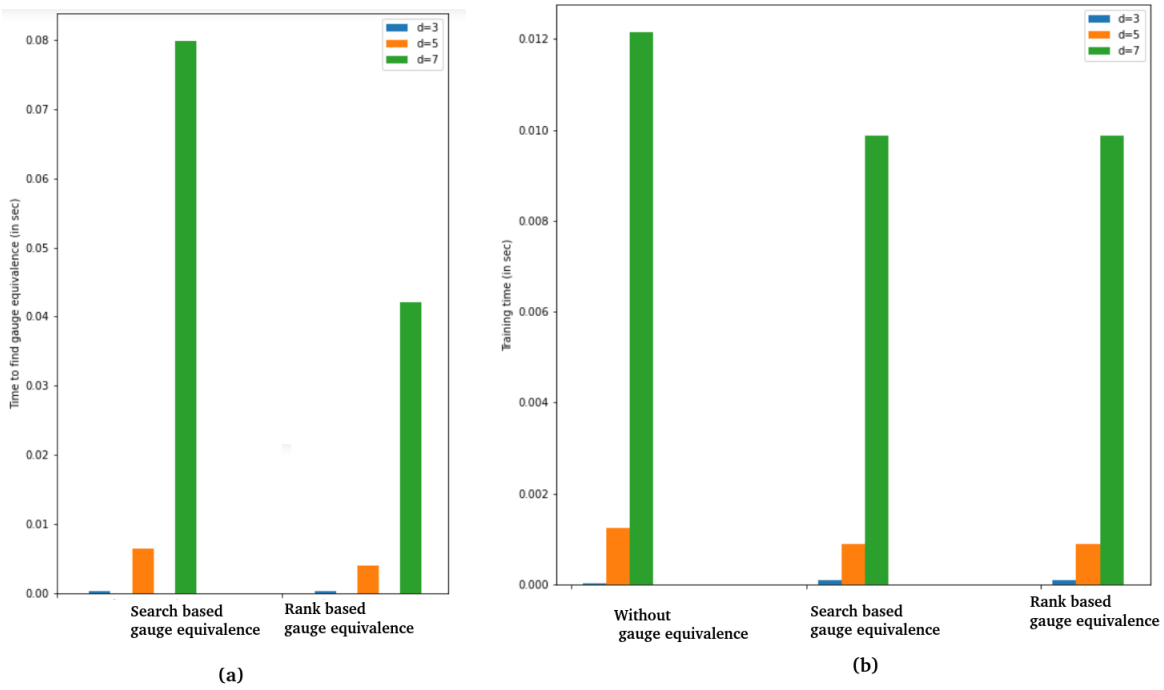


Figure 6.4: Comparison of time (in seconds) needed to (a) pre-process and find gauge equivalence based on linear search and on rank for distance 3 (blue), distance 5 (orange) and 7 (green) heavy hexagon code, (b) train the FFNN based decoder for the three cases, namely without gauge equivalence, with search based gauge equivalence and with rank based gauge equivalence

In Table 6.5 we note the threshold value for MWPM and FFNN based decoders (With and Without Using Gauge equivalence) and pseudo-threshold values for both decoders in case of distance 3, 5, and 7, for this same noise model. Here we observe $\sim 1.15\times$ (that is about $\sim 14\%$) increase in the threshold for ML-decoders using gauge equivalence as compared to ML-decoders without using gauge equivalence.

In Fig 6.4(a) we compare the time to find the gauge equivalence. which is a preprocessing step before training the ML decoder. We observe that the rank based method is faster than the search based method, which supports our Corollary 16 in Section 4.3.

Table 6.5: Comparison of FFNN based decoder, result without and with gauge equivalence in case of bit flip error on data qubit along with measurement and stabilizer errors

Decoding method	Code distance d	Threshold	Pseudo Threshold
MWPM	3	0.0035	0.00018
	5		0.00023
	7		0.0013
FFNN without gauge equivalence	3	0.01375	0.0055
	5		0.008
	7		0.011
FFNN with search based gauge equivalence	3	0.01586	0.0081
	5		0.0102
	7		0.0112
FFNN With rank based gauge equivalence	3	0.01587	0.0082
	5		0.0103
	7		0.0112

In Fig 6.4(b) we have shown the comparison of training time (in seconds) needed for distance 3, 5 and 7 heavy hexagon code by our FFNN based decoder for three cases, namely without gauge equivalence, with search based gauge equivalence and with rank based gauge equivalence. Decoding using gauge equivalence for both search and rank based methods, takes less time than without gauge equivalence. The gauge equivalence technique reduces the number of error classes, resulting in a classification problem with fewer classes, and thus the training of ML model becomes not only faster but also more accurate. For all the three cases, the time required for $d = 7$ (green bar) is significantly higher than that for $d = 5$ (orange bar) or $d = 3$ (blue bar).

For an instance of a decoding problem for $d = 3$ heavy hexagon QECC using bit flip noise model, while the decoding time for MWPM decoder is approx 6×10^{-6} seconds, it is approx 3.8×10^{-6} seconds for our FFNN based decoder using rank based gauge equivalence. This supports our intuition that ML based decoding is faster than MWPM based decoder for heavy hexagon QECC.

Performance of FFNN based decoder for phase flip error

Fig. 6.5 depicts the performance of our FFNN-based decoder for heavy hexagon code having distances 3, 5, and 7, using search based gauge equivalence for phase

flip error on data qubit along with measurement and stabilizer errors. We observe that the probability of physical error above which the probability of logical error increases with the distance of the QECC, is 0.0118 in the case of our FFNN based decoder, i.e., $\sim 19\times$ greater than that of MWPM based decoder, for which it is 0.00063. For decoding phase flip errors in heavy hexagon code, we have primarily made use of the Bacon-Shor code because as stated earlier heavy hexagon code is a combination of surface code and subsystem code (Bacon Shor code) [CR18]. It may be pointed out that the definition of threshold as given in Section 1 is not applicable to Bacon-Shor codes [Bac06].

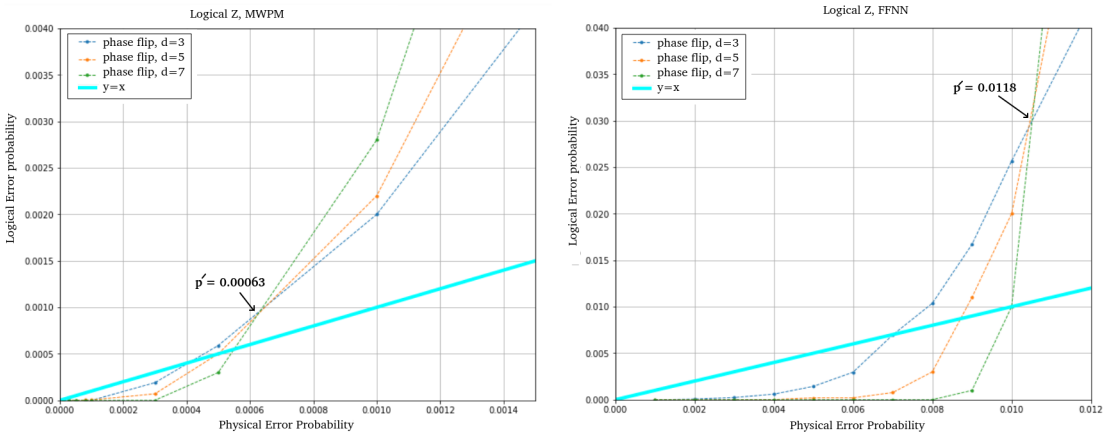


Figure 6.5: Phase flip error rates for the heavy hexagon code – (a) In MWPM based decoder (b) In FFNN based decoder with gauge equivalence.

Performance of FFNN based decoder for depolarization noise

Lastly, we present the performance of our FFNN-based decoder for heavy hexagon codes of distance 3, 5, and 7 in the case of depolarizing noise in terms of logical X (bit flip) and logical Z (phase flip) errors in Table 6.4.3. Our model outperforms the existing decoders.

Table 6.6: Comparison of FFNN based decoder with MWPM decoder for logical X in depolarization noise model

Decoding method	Code distance d	Threshold	Pseudo-threshold
MWPM	3	0.0045	0.0005
	5		0.002
	7		0.0032
FFNN (with gauge equivalence)	3	0.0245	0.0105
	5		0.0125
	7		0.0207

Hence for depolarization noise model, the threshold of our FFNN based decoder is 0.0245 in the case of heavy hexagon code, which is much higher than that of MWPM, i.e., 0.0045 [CZ+20]. Hence our proposed ML-based decoding method achieves $\sim 5\times$ higher values of threshold than that by MWPM. Further, ML based decoders have better (higher) pseudo threshold that MWPM. Note: For depolarization noise model, the threshold for FFNN based decoding was 0.03 with surface code [BSM+22].

6.4.4 Scalability of our ML based decoder

A supervised ML model consists of a training and a testing phase. It is the training phase which often has a significant time requirement. For training a neural network having four layers with i, j, k and l nodes respectively, N training samples, and n epochs, the worst case time complexity is $\mathcal{O}(nN.(ij + jk + kl))$ [Alp21]. The number of nodes in each layer scales linearly with the number of qubits, which is $\mathcal{O}(d^2)$, d being the distance of the QECC. The number of nodes for each layer is bounded by $\mathcal{O}(d^2)$. Therefore the time complexity for training is $\mathcal{O}(nN.d^4)$. While the training time of the ML model increases polynomially with the distance for a fixed-size set of training samples, it increases linearly with the size of the training set for a fixed distance.

Once the training is performed, decoding of each syndrome is essentially executing

the ML model once with that syndrome as the input. The time complexity for decoding each syndrome is, thus, $\mathcal{O}(ij + jk + kl) = \mathcal{O}(d^4)$.

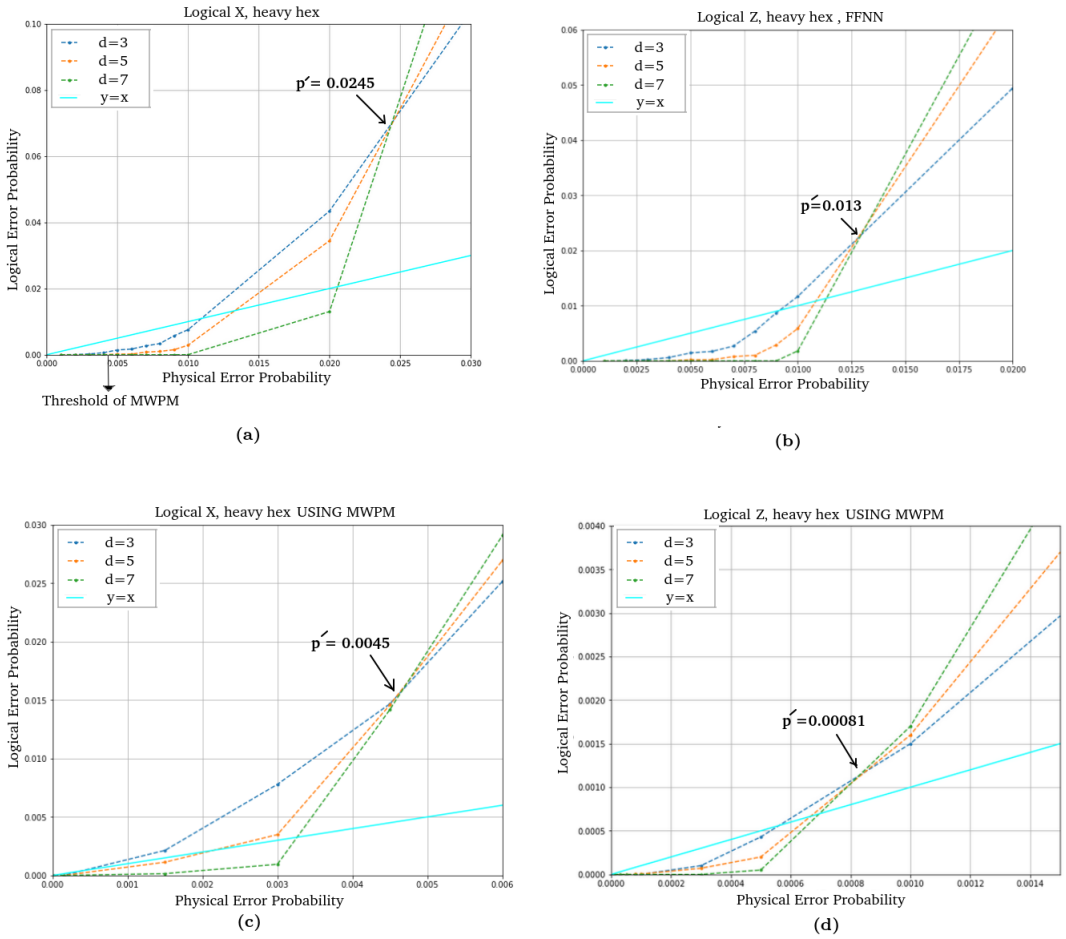


Figure 6.6: In FFNN based decoder for depolarizing noise model (a) Logical X error rates and (b) logical Z error rates for the heavy hexagon code. In MWPM based decoder (c) Logical X error rates and (d) logical Z error rates for the heavy hexagon code.

Training of an ML model can be considered as a preprocessing step. With increase in distance of the code, it is necessary to increase the number of training samples N in order to have a fair training. Since training can become expensive for higher distance codes, it may be possible to use a divide-and-conquer method to divide the

lattice into multiple smaller (and possibly overlapping) sublattices so that each of them can be trained efficiently. However, this method has the challenge of *knitting* the results from these sublattices into the final result corresponding to the lattice. We plan to explore this in future.

For graph based QECC decoders such as MWPM and UF, let $G = (V, E)$ be the syndrome graph, where $|V| = \mathcal{O}(d^2)$. The worst case time complexity of MWPM decoder is $\mathcal{O}(V^3 logV) = \mathcal{O}(2 \times d^6 \times logd)$ [VBA20]. Thus both the decoding time and the performance of MWPM is inferior to that of our proposed ML decoder. On the other hand, UF decoder scales almost linearly with the number of vertices on the graph as $\mathcal{O}(d^2 \cdot \alpha(n))$ [DN21] where $\alpha(n)$ is the Ackermann's function. While this is lower than our ML decoder, it is to be noted that Union Find shows a significantly poor performance compared to both MWPM and our ML decoder in terms of error threshold which is the most important factor in QECC error syndrome decoding.

Therefore, the decoding time of our proposed ML decoder scales polynomially with the distance of the QECC, and performs significantly better than both MWPM and UF decoder overall.

6.4.5 Machine Learning based Decoding of Heavy hexagon QECC for Asymmetric Quantum Noise

We have also proposed ML-based decoders for heavy hexagon code have been implemented and tested under the amplitude damping, amplitude-phase damping combined as well as phase damping noise models. We have utilized the stabilizer structure of the Minimum Weight Perfect Matching (MWPM) decoder proposed in [CZ+20] and implemented it using the 'PyMatchingV2' library [HG23] for the sake of comparison.

Performance of our decoder for various asymmetric noise models

We present the performance of our FFNN based decoder for heavy hexagon codes with distance 3, 5, and 7 for amplitude damping noise in Table 6.7. Our model outperforms the MWPM decoders in terms of threshold and pseudo-threshold. Note that since it has been already established in [BM+24] that MWPM outperforms Union Find decoder, we do not perform that comparison here.

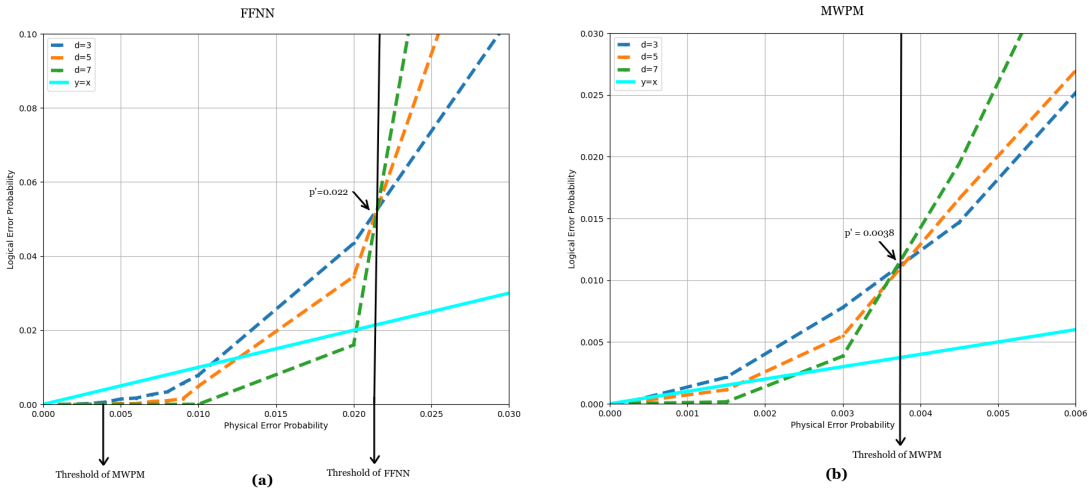


Figure 6.7: (a) Using FFNN based decoder, Logical error rates for the heavy hexagon code and (b) Using MWPM based decoder, Logical error rates for the heavy hexagon code, for Amplitude damping noise model, $\gamma_A = 0.01$

Figs. 6.7 (a) and (b) present the values of the threshold and pseudo-threshold of ML based and MWPM based decoders for amplitude damping noise model with $\gamma_A = 0.01$. We note that the ML-based decoder achieves $\sim 5\times$ higher values of threshold than MWPM for this noise model.

For amplitude phase damping combined noise model, from Table 6.7 we can see that the ML-based decoder achieves $\sim 22\times$ higher values of threshold than MWPM based decoder.

Note that twirling a phase damping noise model generates a phase flip error which

Table 6.7: Comparison of FFNN based decoder with MWPM decoder for logical error in various noise model

Decoding method	Code distance d	Noise Model	Threshold	Pseudo-threshold
MWPM	3 5 7	Amplitude Damping $\gamma_A = 0.01$ (Asymmetric)	0.0038	0.0005 0.0018 0.0025
FFNN (with gauge equivalence)	3 5 7	Amplitude Damping $\gamma_A = 0.01$ (Asymmetric)	0.022	0.0102 0.0122 0.0201
MWPM	3 5 7	Amplitude Phase Damping $\gamma_A = 0.02, \gamma_P = 0.01$ (Asymmetric)	0.00052	0.00039 0.00045 0.00052
FFNN (with gauge equivalence)	3 5 7	Amplitude Phase Damping $\gamma_A = 0.02, \gamma_P = 0.01$ (Asymmetric)	0.0115	0.0068 0.0086 0.0092
MWPM	3 5 7	Amplitude Phase Damping $\gamma_A = 0.01, \gamma_P = 0.001$ (Asymmetric)	0.00048	0.00036 0.00042 0.00049
FFNN (with gauge equivalence)	3 5 7	Amplitude Phase Damping $\gamma_A = 0.01, \gamma_P = 0.001$ (Asymmetric)	0.0112	0.0066 0.0081 0.009

is essentially (for details see [BM+24]). We have verified that the threshold obtained for phase flip model, and that for phase damping model with twirling are equal.

Performance of FFNN based decoder for variation in γ between training and testing

As mentioned earlier, real-world noise models often exhibit asymmetry. However, this asymmetry can vary with time as the system drifts. Therefore, it is useful if a decoder, trained with certain values of γ_A and/or γ_P, performs acceptably well for minor change in these values. Here we numerically examined the performance of our ML based decoder when trained and tested on different values of γ_A and γ_P. Fig. 6.8 shows the threshold obtained for the ML decoder when trained with $\gamma_A = 0.01$ and tested across different values of γ_A. The result shows that even for a $10\times$ variation in γ_A, the threshold drops by $\sim 3.6\%$ only. In fact, daily calibration data from IBM Quantum hardware reports a variation in asymmetry which is much less then $10\times$. Therefore, our ML decoder is shown to be robust against minor variation in asymmetry. Note that while Fig. 6.8 shows the result for amplitude damping noise only, we obtain similar robustness of our decoder for

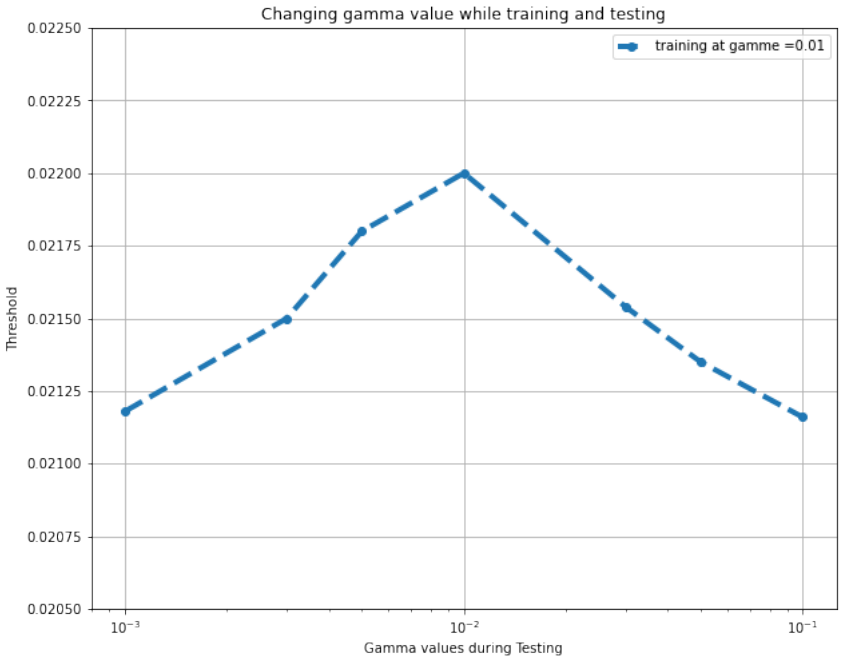


Figure 6.8: Training in one γ_A value and testing in other γ_A values

amplitude-phase damping model as well.

6.5 Discussion

In this work, we have proposed an ML-decoder for heavy hexagon code which makes use of a novel technique based on gauge equivalence to improve the performance of decoding. First we map decoding to a ML based classification problem. Exploiting the properties of heavy hexagon code as a subsystem code, we have defined gauge equivalence, which, in turn, reduces the number of error classes for ML based classification. We have proposed *search based* and *rank based* methods for finding the gauge equivalence; the former being faster for finding equivalence in phase flip errors and the later for bit flip errors.

We have tested our decoders for heavy hexagon codes with distances upto 7 for both symmetric and asymmetric noise models. We have shown that even the naïve

ML decoder outperforms the MWPM and UF based decoders in terms of pseudo-threshold and threshold. Using gauge equivalence leads to further improvement in the performance of our decoder. This research provides a plausible method for scaling current quantum devices to the fault-tolerant era.

Finally, we synthesize our findings and insights in the concluding chapter, offering a comprehensive overview of our contributions to error reduction in quantum computing.

CHAPTER 7

Conclusion and Future directions

Contents

7.1 Summary 155
7.2 Chapter-wise Contributions 156
7.3 Future Directions 158

7.1 Summary

This thesis addresses critical challenges in quantum computing, specifically focusing on error suppression and resource management in both near-term and future quantum devices. The contributions span multiple facets of quantum circuit scheduling and error correction, leveraging machine learning and optimization techniques to enhance the performance and reliability of quantum computations. Through a detailed exploration of various methodologies, this research provides valuable insights and solutions that push the boundaries of current quantum computing capabilities.

7.2 Chapter-wise Contributions

Chapter 3: Time and Noise Optimization for Distributed Scheduling of Quantum Circuits

In Chapter 3, we propose a noise and time optimized distributed scheduler that schedules the subcircuits obtained after circuit cutting to hardware such that the fidelity is maximized, and yet the execution time on each hardware is restricted by a pre-specified limit. This method combines inter-device parallelization with noise-aware scheduling to optimize the fidelity of the circuit. The results demonstrate that our method outperforms the fidelity of the uncut circuit executed on the least noisy device, while significantly reducing execution time on a quantum processor. This chapter also discusses the potential for this method to be particularly useful in the near-term when devices are noisy and execution time on quantum devices is limited.

Chapter 4: Resource-aware Scheduling of Multiple Quantum Circuits on a Hardware Device

Chapter 4 addresses the critical challenge of optimizing quantum circuit scheduling to enhance the throughput and efficiency of quantum computing hardware. By drawing analogies to the classical bin packing problem, we demonstrated the NP-Hard nature of our problem, which involves placing multiple quantum circuits onto quantum processing units while considering the inherent noise and limited qubit connectivity. Our proposed solution, using integer linear programming (ILP) or a greedy heuristic-based solution on compatibility graphs and maximal cliques, effectively balances the trade-off between noise reduction and resource utilization. The chapter details the experimental results, showing significant improvements in throughput and efficiency, making it a viable approach for real-world quantum computing applications.

Chapter 5: Machine-Learning based Decoding of Surface Code Syndromes in Quantum Error Correction

In Chapter 5, we proposed an ML-decoder to correct both symmetric and asymmetric depolarizing noise on surface codes. Our decoder operates on two levels: a low-level that tries to accurately predict the error on the qubits, followed by a high-level that detects any logical error introduced by the low-level decoder. This method employs neural networks (FFNN and CNN) for surface codes of distances 3, 5, and 7. The results indicate that our proposed ML-decoder outperforms the Minimum Weight Perfect Matching (MWPM) method, showing approximately a $2\times$ increase in threshold and a $10\times$ increase in pseudo-threshold. Additionally, our decoder performs equally well for asymmetric errors, which are more realistic in quantum devices. The chapter also explores the performance of various ML models with different levels of sophistication, showing that increased model complexity requires more decoding time but does not necessarily yield better performance.

Chapter 6: Efficient Syndrome Decoder for Heavy Hexagonal QECC via Machine Learning for Symmetric and Asymmetric Noise Model

Chapter 6 presents an ML-decoder for heavy hexagonal code, employing a novel technique based on gauge equivalence to improve decoding performance. We showed that even a naive ML decoder outperforms MWPM and Union-Find (UF) based decoders in terms of pseudo-threshold and threshold. Using gauge equivalence further enhances the performance of our decoder, demonstrating the potential for this approach to scale current quantum devices towards fault tolerance. For symmetric noise, we utilized search-based and rank-based methods for finding gauge equivalence, with the former being faster for phase flip errors and the latter for bit flip errors. For asymmetric noise, the rank-based gauge equivalence method was employed, significantly improving the performance of the ML decoder

compared to MWPM decoders. This chapter suggests future research directions such as studying the applicability of gauge equivalence and ML-based decoders for other noise models, including Pauli and amplitude damping and amplitude-phase damping noise.

7.3 Future Directions

This research opens several avenues for future exploration, which are crucial for advancing the field of quantum computing towards practical and scalable solutions.

Distributed Scheduling

Future research may delve into scheduling circuits where balanced partitioning is too costly, exploring more flexible and adaptive scheduling techniques. Investigating the integration of advanced noise models and real-time adaptability in distributed schedulers could further enhance the fidelity and efficiency of quantum computations.

Intra-device Scheduling

In the realm of intra-device scheduling, developing more sophisticated heuristic methods and exploring their impact on resource utilization could provide deeper insights. Additionally, integrating machine learning techniques to predict and mitigate noise effects dynamically could revolutionize how quantum circuits are scheduled and executed.

Machine Learning for Error Correction

For error correction, future work could focus on considering noisy measure qubits and imperfect measurements, which would provide a more realistic assessment of the ML-decoder's performance in practical scenarios. Further studies could explore the cumulative effects of multiple noise sources and the applicability of gauge equivalence and ML-based decoders to other noise models such as Pauli and amplitude damping noise.

Cross-disciplinary Techniques

Exploring cross-disciplinary techniques, such as integrating classical optimization methods with quantum-specific adaptations, could lead to novel solutions for quantum circuit scheduling and error correction. Collaborations with fields like machine learning, operations research, and classical computing could yield innovative approaches that enhance the overall performance and reliability of quantum devices.

In conclusion, this thesis lays the groundwork for significant advancements in quantum computing, addressing critical challenges in error suppression and resource management. By exploring these future directions, researchers can build upon the contributions of this thesis, paving the way for practical, scalable, and fault-tolerant quantum computing systems.

List of publications by the author

Related to this thesis

In Journals

1. Machine-Learning based Decoding of Surface Code Syndromes in Quantum Error Correction. Debasmita Bhoumik, Pinaki Sen, Ritajit Majumdar, Susmita Sur-Kolay, Latesh K. J., Sundaraja Sitharama Iyengar. Special Issue of Multidisciplinary Sciences And Advanced Technology, Journal of Engineering Research and Sciences, 2022.DOI: 10.55708/js0106004

2. Efficient Syndrome Decoder for Heavy Hexagonal QECC via Machine Learning. Debasmita Bhoumik. Ritajit Majumdar, Dhiraj Madan, Dhinakaran Vinayagamurthy, Shesha Raghunathan, Susmita Sur-Kolay. ACM Transactions on Quantum Computing,Volume 5, Issue 1, 2024. https://doi.org/10.1145/3636516

3. Distributed Scheduling of Quantum Circuits with Noise and Time Optimization. Debasmita Bhoumik, Ritajit Majumdar, Amit Saha, Susmita Sur-Kolay. arXiv preprint arXiv:2309.06005 (2023). Submitted in IEEE TETC on 14th August 2024.

4. Resource-aware Scheduling of Multiple Quantum Circuits on a Hardware Device. Debasmita Bhoumik, Ritajit Majumdar, Susmita Sur-Kolay. arXiv:2407.08930 (2024). Submitted in IEEE TETC on 11th November 2024.

In Peer-reviewed International Conference Proceedings

1. Machine Learning based Decoding of Heavy Hexagonal QECC for Asymmetric Noise. Debasmita Bhoumik, Ritajit Majumdar, Dhiraj Madan,, Susmita Sur-Kolay. IEEE Computer Society Annual Symposium on VLSI 2024 (ISVLSI 2024), Knoxville. USA.

Not included in this thesis

In Journals

1. Surface Code Design for Asymmetric Error Channels. Utkarsh Azad, Aleksandra Lipińska, Shilpa Mahato, Rijul Sachdeva, **Debasmita Bhoumik** and Ritajit Majumdar. 2021. IET Quantum Communication, Volume 3, Issue 3, Pages 174-183, September 2022; DOI : 10.1049/qtc2.12042.

2. Amplitude damping error on QKD: Effect and a probable bypass. Munsi Afif Aziz, Bishwajit Prasad Gond, Srijita Nandi, Soujanya Ray, **Debasmita Bhoumik**, Ritajit Majumdar. Modern Physics Letters A. Vol. 38, No. 10n11, 2350058 (2023).

In Peer-reviewed International Conference Proceedings

1. Optimized QAOA ansatz design for two-body Hamiltonian problems. Ritajit Majumdar, **Debasmita Bhoumik**, Dhiraj Madan, Dhinakaran Vinayagamurthy, Shesha Raghunathan, and Susmita Sur-Kolay. 37th International Conference on VLSI Design and 23rd International Conference on Embedded Systems (VLSID) 2024.

Preprints

1. Synergy of machine learning with quantum computing and communication. **Debasmita Bhoumik**, Susmita Sur-Kolay, Latesh K. J., Sundaraja Sitharama Iyengar. arXiv preprint arXiv:2310.03434. 2023

2. Depth Optimized Ansatz Circuit in QAOA for Max-Cut. Ritajit Majumdar, **Debasmita Bhoumik**, Dhiraj Madan, Dhinakaran Vinayagamurthy, Shesha Raghunathan, and Susmita Sur-Kolay. arxiv:2110.04637. 2021

3. Optimizing Ansatz Design in QAOA for Max-cut. Ritajit Majumdar, Dhiraj Madan, **Debasmita Bhoumik**, Dhinakaran Vinayagamurthy, Shesha Raghunathan, and Susmita Sur-Kolay. arXiv:2106.02812. 2021

Bibliography

[AA+19] G. Aleksandrowicz, T. Alexander, et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019.

[Alp21] E. Alpaydin. Machine learning. 2021.

[Bac06] D. Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Physical Review A*, 73(1):012340, 2006.

[BB+93] C. H. Bennett, G. Brassard, et al. Quantum cryptography: Public key distribution and coin tossing. *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, 1993.

[BC20] T. Bochen and J. Cong. Optimality study of existing quantum computing layout synthesis tools. *IEEE Transactions on Computers*, 70(9):1363–1373, 2020.

[BK98] S. B. Bravyi and A. Y. Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052*, 1998.

[BM+23] V. D. Berg, Z. K. Minev, et al. Probabilistic error cancellation with sparse pauli–lindblad models on noisy quantum processors. *Nature Physics*, pages 1–6, 2023.

[BM+24] D. Bhoumik, R. Majumdar, et al. Efficient syndrome decoder for

heavy hexagonal qecc via machine learning. *ACM Transactions on Quantum Computing*, 5(1):1–27, 2024.

[BS+22] S. Basu, A. Saha, et al. i-qer: An intelligent approach towards quantum error reduction. *ACM Transactions on Quantum Computing*, 3(4):1–18, 2022.

[BSM+22] D. Bhoumik, P. Sen, R. Majumdar, S. Sur-Kolay, KJ Latesh K, and S. S. Iyengar. Machine-learning based decoding of surface code syndromes in quantum error correction. *Journal of Engineering Research and Sciences*, 1(6), 2022.

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[CD+22] T. Chatterjee, A. Das, et al. Qurzon: A prototype for a divide and conquer-based quantum compiler for distributed quantum systems. *SN Computer Science*, 3(4), jun 2022.

[CLRS22] T. H. Cormen, C. E. Leiserson, R. L Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[cpl] IBM ILOG CPLEX Optimization Studio 22.1.1. https://www.ibm.com/products/ilog-cplex-optimization-studio.

[CR18] C. Chamberland and P. Ronagh. Deep neural decoders for near term fault-tolerant experiments. *arXiv preprint arXiv:1802.06441*, 2018.

[CS+19] A. M. Childs, E. Schoute, et al. Circuit transformations for quantum architectures. *arXiv preprint arXiv:1902.09102*, 2019.

[CZ+20] C. Chamberland, G. Zhu, et al. Topological and subsystem codes on low-degree graphs with flag qubits. *Physical Review X*, 10(1):011022, 2020.

[DKLP02] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9), 2002.

[DN21] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, 2021.

[DT+19] P. Das, S.S. Tannu, et al. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019.

[Edm65a] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Edm65b] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[EM+22] A. Eddins, M. Motta, et al. Doubling the size of quantum simulators by entanglement forging. *PRX Quantum*, 3(1):010309, 2022.

[FSG09] A. G. Fowler, A. M. Stephens, and P. Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80(5):052312, 2009.

[FWH12] A. G. Fowler, A. C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012.

[Got97] D. Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052*, 1997.

[GP17] G. G. Guerreschi and J. Park. Gate scheduling for quantum algorithms. *ArXiv e-prints*, 2017.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[HG23] O. Higgott and C. Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[Hig22] O. Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, 2022.

[Hil86] R. Hill. *A first course in coding theory*. Oxford University Press, 1986.

[IM07] L. Ioffe and M. Mézard. Asymmetric quantum error-correcting codes. *Phys. Rev. A*, 75:032345, Mar 2007.

[JATK+24] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D Nation, Lev S Bishop, Andrew W Cross, et al. Quantum computing with qiskit. *arXiv preprint arXiv:2405.08810*, 2024.

[JG+23] A. Javadiabhari, J. M. Gambetta, et al. Validating and estimating runtime for quantum algorithms, February 14 2023. US Patent 11,580,433.

[JW22] M. Johnson and H. Wang. Object recognition in cluttered environments using convolutional neural networks. *Pattern Recognition Letters*, 135:45–53, 2022.

[KG22] A. Kumar and R. Gupta. Predicting soil properties using machine learning: A review. *Journal of Agricultural Science*, 10(5):56–78, 2022.

[KH+19] A. Kole, S. Hillmich, et al. Improved mapping of quantum circuits to ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2375–2383, 2019.

[Kit97] A. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

[KJ17] S. Krastanov and L. Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7(1), 2017.

[KM+17] A. Kandala, A. Mezzacapo, et al. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[KM+23] T. Khare, R. Majumdar, et al. Parallelizing quantum-classical workloads: Profiling the impact of splitting techniques. *arXiv preprint arXiv:2305.06585*, 2023.

[LA+23] B. Luciano, M. B. Agata, et al. Circuit Knitting Toolbox. https://github.com/Qiskit-Extensions/circuit-knitting-toolbox, 2023.

[Llo00] S. Lloyd. Quantum mechanical computers. *Science*, 273(5278):1073–1078, 2000.

[LM+19] M. Li, D. Miller, et al. 2d compass codes. *Physical Review X*, 9(2):021041, 2019.

[LMPZ96] R Laflamme, C Miquel, J P Paz, and W H Zurek. Perfect quantum error correcting code. *Phys. Rev. Lett.*, 77:198–201, Jul 1996.

[LZF20] S. Li, X. Zhou, and Y. Feng. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. *IEEE Transactions on Computers*, 70(11):1777–1788, 2020.

[MB+16] R. Majumdar, S. Basu, et al. Error tracing in linear and concatenated quantum circuits. *arXiv preprint arXiv:1612.08044*, 2016.

[MB+19] P. Murali, J.M. Baker, et al. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 1015–1029, 2019.

[MF21] K. Mitarai and K. Fujii. Constructing a virtual two-qubit gate by sampling single-qubit operations. *New Journal of Physics*, 23(2):023021, 2021.

[MW22] R. Majumdar and C. J. Wood. Error mitigated quantum circuit cutting. *arXiv preprint arXiv:2211.13431*, 2022.

[NC10] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[NK+21] P. D. Nation, H. Kang, et al. Scalable mitigation of measurement errors on quantum computers. *PRX Quantum*, 2(4):040326, 2021.

[NT23] P. D. Nation and M. Treinish. Suppressing quantum circuit errors due to system variability. *PRX Quantum*, 4(1):010327, 2023.

[PH+20] T. Peng, A. W. Harrow, et al. Simulating large quantum circuits on a small quantum computer. *Physical review letters*, 125(15):150504, 2020.

[pla] plaquette — an all-encompassing fault-tolerance software package. *https://docs.plaquette.design/en/latest/*.

[PR20] S. Pal and A. Ray. Granulated approach to video tracking using deep learning. *International Journal of Computer Vision*, 128(7):1925–1940, 2020.

[Pre98] J. Preskill. Lecture notes for physics 229: Quantum information and computation. *California Institute of Technology*, 16(1):1–8, 1998.

[Pre18] J. Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[PS+21] A. M. Perlin, Z. H. Saleem, et al. Quantum circuit cutting with maximum-likelihood tomography. *npj Quantum Information*, 7(1):64, 2021.

[RFV+17] L. Riesebos, X. Fu, S. Varsamopoulos, C.G. Almudever, and K.L.M. Bertels. Pauli frames for quantum computer architectures. In *Proceedings of the FTQC/Bertels Lab*, 2017.

[S+18] R. Sweke et al. Reinforcement learning decoders for fault-tolerant quantum computation. *arXiv preprint arXiv:1810.07207*, 2018.

[SB22] J. Smith and L. Brown. Human pose estimation with deep residual networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):2345–2358, 2022.

[SC+04] T. G. Draper S. Cuccaro et al. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[SD+20] S. Sivarajah, S. Dilkes, et al. t| ket>: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, 2020.

[Sho95] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, Oct 1995.

[Sho96] P. W. Shor. Fault-tolerant quantum computation. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 56–65, 1996.

[Sho97] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[SM21] R. Singhania and P. Mehta. Predicting the efficacy of online sale using machine learning. *Journal of Business Research*, 134:245–258, 2021.

[SSBD14] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[ST+21] Z. H. Saleem, T. Tomesh, et al. Divide and conquer for combinatorial optimization and distributed quantum computation. *arXiv preprint arXiv:2107.07532*, 2021.

[Ste96] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996.

[Str06] G. Strang. *Linear algebra and its applications*. Belmont, CA: Thomson, Brooks/Cole, 2006.

[SZU+21] J. Stehlik, D. M. Zajac, D. L. Underwood, T. Phung, J. Blair, S. Carnevale, D. Klaus, G. A. Keefe, A. Carniol, M. Kumph, Matthias Steffen, and O. E. Dial. Tunable coupling architecture for fixed-frequency transmon superconducting qubits. *Phys. Rev. Lett.*, 127:080505, Aug 2021.

[T+22] M. Treinish et al. Mapomatic: Automatic mapping of compiled circuits to lownoise sub-graphs. *url: https://github. com/Qiskit-Partners/mapomatic*, 2022.

[TAR+21] Thomas T. Ayral, F. Régent, et al. Quantum divide and compute: exploring the effect of different noise sources. *SN Computer Science*, 2(3):132, 2021.

[TT+21] W. Tang, T. Tomesh, et al. Cutqc: using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*, pages 473–486, 2021.

[V+18] S. Varsamopoulos et al. Designing neural network based decoders for surface codes. *arXiv preprint arXiv:1811.12456*, 2018.

[VBA19] S. Varsamopoulos, K. Bertels, and C. G. Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2), 2019.

[VBA20] S. Varsamopoulos, K. Bertels, and C. G. Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2020.

[VCB17] S. Varsamopoulos, B Criger, and K. Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2017.

[WE16] J. J. Wallman and J. Emerson. Noise tailoring for scalable quantum computation via randomized compiling. *Physical Review A*, 94(5):052325, 2016.

[WFH11] D. S. Wang, A. G. Fowler, and Lloyd CL Hollenberg. Surface code quantum computing with error rates over 1%. *Physical Review A*, 83(2):020302, 2011.

[ZD+23] P. Zhu, W. Ding, et al. A variation-aware quantum circuit mapping approach based on multi-agent cooperation. *IEEE Transactions on Computers*, 72(8):2237–2249, 2023.

[Zur03] W. H. Zurek. Decoherence, einselection, and the quantum origins of the classical. *Reviews of Modern Physics*, 75(3):715–775, 2003.

[ZW17] A. Zulehner and R. Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):996–1008, 2017.