Noise Resilient Approximate Quantum Circuits for NISQ Devices

by

Sohrab Sajadimanesh

A dissertation
Submitted to Lakehead University in
partial fulfillment of the requirements of the degree of
Doctor of Philosophy (Ph.D.)
in
Department of Electrical and Computer Engineering
Lakehead University

Thunder Bay, Ontario, Canada
December 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis.

Supervisor(s): **Dr. Ehsan Atoofian**
Department of Electrical and Computer Engineering,
Lakehead University, Thunder Bay, ON, Canada
Dr. Dimiter Alexandrov
Department of Electrical and Computer Engineering,
Lakehead University, Thunder Bay, ON, Canada

External Examiner: **Dr. Jean Paul Latyr Faye**
Department of Physics
University of Cheikh Anta Diop, Dakar, Senegal

Internal Member: **Dr. Yushi Zhou**
Department of Electrical and Computer Engineering,
Lakehead University, Thunder Bay, ON, Canada
Dr. Thiago E Alves de Oliveira
Department of Computer Science,
Lakehead University, Thunder Bay, ON, Canada

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners..

I understand that my thesis may be made electronically available to the public.

Abstract

The rapid progress of noisy intermediate-scale quantum (NISQ) devices has enabled the execution of quantum algorithms on real hardware, yet their limited qubit counts, short coherence times, and susceptibility to noise pose major challenges to reliable computation. Circuit complexity—including qubit count, circuit depth, and total gate operations—plays a pivotal role in determining the fidelity of quantum programs. Reducing these complexity metrics helps mitigate the detrimental effects of noise and improves the reliability of quantum computations. To address these challenges, this work introduces a series of optimized designs for quantum arithmetic circuits, quantum neural networks (QNNs), and quantum random access memory (QRAM).

We investigate arithmetic circuits, including addition, multiplication, division, and square root operations. By leveraging approximate computing and dynamic circuits, we propose novel designs that significantly reduce depth and gate counts while maintaining acceptable accuracy. Approximate arithmetic units such as quantum adder, multiplier, divider, and square root demonstrate superior performance compared to existing designs, and our quantum division circuits exploit mid-circuit measurement and approximation to enhance fidelity, enabling successful deployment on IBM quantum computers.

The proposed designs reduce the complexity of exact circuits while maintaining the same level of precision in outputs. All approximate circuits presented in this work are superior to existing quantum circuits in terms of depth, gate counts, and number of qubits. We show that running proposed approximate circuits on real quantum computers generates meaningful results.

Building upon these foundations, we develop a noise-resilient quantum neural network (NR-QNN) framework tailored for NISQ devices. NR-QNN employs quantum pruning,

which removes gates with negligible rotation angles, and sensitivity-aware qubit mapping, which allocates critical logical qubits to more reliable physical qubits. These optimizations mitigate the impact of noise, thereby enhancing the robustness of QNNs and enabling them to achieve meaningful inference results on real quantum hardware. Finally, we explore the design of approximate quantum random access memory (QRAM) architectures, a crucial component for enabling large-scale quantum data access. Our approach applies pruning techniques to simplify QRAM circuits and re-trains approximate QNN-based modules to mitigate accuracy loss. This strategy reduces circuit depth and improves error resilience, allowing approximate QRAMs to serve as practical building blocks for QNN applications on NISQ computers.

Acknowledgements

With deep appreciation for those who have profoundly influenced my academic journey, I extend my heartfelt thanks:

I am sincerely grateful to my supervisor, Dr. Ehsan Atoofian, for his unwavering support, insightful guidance, and academic mentorship. His encouragement and expertise have played a pivotal role in shaping the direction of my research. It has truly been an honor to work under his supervision.

I deeply appreciate the profound insights provided by Dr. Jean Paul Latyr Faye. His guidance has been invaluable in navigating the complexities of quantum computing, fostering my growth and enthusiasm throughout this learning journey.

I am especially thankful to my beloved family. The steadfast support, care, and understanding of my wife and parents have been invaluable throughout this journey. Their belief in me has been a constant source of motivation and strength.

Dedication

To my beloved wife, for her unwavering love, patience, and endless support, to my Mother, for her endless care and faith in my potential, and to the cherished memory of my father, whose values, sacrifices, and love continue to inspire me every day.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AQDFC Approximate Quantum Divider with Full Carry 85, 86, 130, 133

AQDHC Approximate Quantum Divider with Half Carry 85, 86, 130, 133

AQDNC Approximate Quantum Divider with No-Carry 85, 86, 130, 132, 133

AQDOC Approximate Quantum Divider with One Carry 85, 86, 130, 133

AQMFC Approximate Quantum Multiplier with Full Carry 67, 68, 125, 126, 128, 130

AQMHC Approximate Quantum Multiplier with Half Carry 67, 68, 125, 126, 130

AQMNC Approximate Quantum Multiplier with No-Carry 67, 68, 125, 126, 128, 130

AQMOC Approximate Quantum Multiplier with One Carry 67, 68, 125, 126, 128, 130

BB-QRAM Bucket-Brigade Quantum Random Access Memory 49

BCI Breast Cancer Image 142

C-AQAFC Controlled Approximate Quantum Adder with Full Carry 59–63, 67, 83, 124

C-AQAHC Controlled Approximate Quantum Adder with Half Carry 60–63, 67, 124

C-AQANC Controlled Approximate Quantum Adder with No-Carry 57–63, 67, 98, 100, 124, 128

C-AQAOC Controlled Approximate Quantum Adder with One Carry 61, 63, 67, 124

CCQC Circuit-Centric Quantum Classifier 44

ED Error Distance 55, 61

EQGAN Entangling Quantum Generative Adversarial Network 50

KNN K-nearest neighbors 143

LSB Least Significant Bit 59, 61, 63, 65, 70, 76, 79, 90, 126

LSQ Least Significant Qubit 71, 74, 77, 79, 96, 97

MAC multiplication-and-accumulation 40, 41

MED Mean Error Distance 55, 100

MSB Most Significant Bit 59, 61, 67, 70, 76, 79, 126

MSE Mean Squared Error 117, 154

MSQ Most Significant Qubit 71–73, 79, 82, 85, 91, 94–97

NC-AQAFC Non-Controlled Approximate Quantum Adder with Full Carry 82, 83

NC-AQAHC Non-Controlled Approximate Quantum Adder with Half Carry 82, 83

NC-AQANC Non-Controlled Approximate Quantum Adder with No-Carry 81–83, 98

NC-AQAOC Non-Controlled Approximate Quantum Adder with One Carry 82, 83

NISQ Noisy Intermediate-Scale Quantum 1–5, 7–9, 11, 12, 22, 27, 30–34, 36, 38, 40, 41, 51, 54, 57, 58, 64, 80–82, 97, 99, 105, 108, 110, 112, 115, 117, 118, 120–122, 124–127, 135, 138, 139, 144–147, 150, 155, 157–160, 162, 163

NMED Normalized Mean Error Distance 55, 60–62, 68, 85, 100

NN Neural Network 5, 6, 9, 11, 40, 41, 106, 116

NR-QNN Noise Resilient Quantum Neural Network 101, 121, 122, 147, 149, 151, 153, 154

PQC Parameterized Quantum Circuit 41, 42, 44, 48, 50, 51, 102–105, 107–109, 115, 116, 118, 119, 149–151, 153–155

PST Probability of Success Trial 2, 3

QCA Quantum-Controlled Adder 71, 72, 74, 77, 79, 83–85, 93, 94, 97, 100, 129–132, 138–140, 144

QCAS Quantum-Controlled Adder/Subtractor 77, 79, 84, 85, 93, 95–99, 129, 130, 132, 138–140, 144

QFA Quantum Full-Adders 80, 82

QISKIT Quantum Information Science Kit 8, 9, 27, 112, 113, 126, 134, 148, 154

QML Quantum Machine Learning 36, 51

QNN Qunatum Neural Network 5–7, 9–12, 16, 24, 36–41, 43, 44, 47, 48, 51, 101–109, 112–116, 119, 147–157, 159, 163

QOC Quantum on-chip 48

QRAM Quantum Random Access Memory 5–12, 39, 48–51, 101, 102, 115–122, 154–157, 159

QSQR Quantum Square Root 31–34, 93, 94, 97, 98, 100, 121, 122, 139–146

QSUB Quantum Subtractor 71–74, 77, 78, 83–85, 129–132

RAM Random Access Memory 48, 49

RCAS Reversible Controlled Adder and Subtractor 32

RCSM Reversible Controlled-Subtract-Multiplex 32

SQR Square Root 87–90, 92, 93, 138, 142, 143

SSIM Structural Similarity Iindex Metric 125, 126, 134, 142

VLSI Very Large-Scale Integration 33

VQA Variation-aware Qubit Allocation 47

Chapter 1

Introduction

Over the last few years, quantum computing has emerged as one of the most rapidly developing research areas. Quantum computing is a promising paradigm with tremendous potential for applications that require significant computing power from the underlying hardware. It is able to accelerate certain problems and offer substantial computational power beyond the capability of classical computing in many critical domains such as molecular dynamics, database search, material design, etc [2],[3],[4]. However, most efforts in this area focused on the theoretical aspect of quantum computing. The last few years represent significant milestones in quantum computing as some companies, such as IBM, Google, and Rigetti, have offered real quantum computers that make it feasible to run quantum programs on real quantum machines [5],[6],[7]. These quantum computers are called Noisy Intermediate-Scale Quantum (NISQ) devices and comprise up to a few hundred quantum bits.

However, these early quantum computers are susceptible to noise and can accommodate small-scale quantum circuits. Public access to these devices [8] has led to many attempts

to solve real-world applications on these devices. As John Preskill stated [5]: "The 100-qubit quantum computer will not change the world right away − we should regard it as a significant step toward the more powerful quantum technologies of the future." We are at the early stages of quantum computing, and there are many unknowns in this area, which provide an ample opportunity for research on exploring quantum computing for practical applications. For example, Google claimed a 53-qubit quantum computer that can offer quantum supremacy by completing a computational task in 200 seconds, whereas the same task may take $10K$ years [9] on a state-of-the-art supercomputer. Later on, the estimation for execution time of the task on classical computers was lowered to 2.5 days [10]. Recently, Google Quantum AI team achieved a significant milestone toward practical quantum advantage by using its 65-qubit superconducting processor to perform a computation that would take the Frontier supercomputer approximately 3.2 years, completing it in just over two hours, which is about 13000 times faster [7].

1.1 Introduction to Quantum Computing

Quantum computers rely on qubits to store and process data. Qubits are susceptible to coherent, gate, measurement, and crosstalk errors and can hold quantum states for only tens of microseconds. Therefore, contemporary quantum computers inherently operate in a noisy environment [5]. In the NISQ computing model, a program is run on a real quantum computer, and the output of the program is measured. This process is repeated for several trials, and the distribution of expected output (output with no error) is computed. This metric is called the Probability of Success Trial (PST). If PST is a reasonable value, then the program is treated as a working program with a correct answer on the quantum computer. PST depends on the error rate in the quantum computer as well as the size of

the quantum circuits. Recently, various techniques have been proposed to increase PST by either performing computation with the presence of noise to generate results that are better than the worst-case scenario [11], [12] or reducing the size of quantum circuits and thus the number of computations [13], [14]. One of these techniques to deploy algorithms to noisy quantum computers is error correction codes. However, quantum error correction requires tens of qubits to support one fault-tolerant qubit [1].

Other approaches to increase the accuracy of NISQ devices are decreasing the number of required qubits and reducing the depth of the quantum circuit. The quantum circuit depth is determined by the critical path, which is the longest path from inputs to outputs. The other factor that impacts the accuracy of a quantum circuit is the number of quantum gates. Quantum gates, including single-qubit and multi-qubit gates, are not ideal, and they cause errors in qubits. A circuit with a higher number of quantum gates is more susceptible to errors. Therefore, minimizing the runtime and complexity of a quantum circuit is crucial as they do not just reduce the time-to-solution, but they also enhance the accuracy of the solution itself.

Recently, the design of arithmetic units, such as multipliers with a Clifford+T[1] gate set [15], has received significant attention in the literature [16], [17],[18], [19]. Clifford+T gate set is an attractive option for the implementation of quantum circuits, as they can be made fault-tolerant with error correction codes [20],[21]. However, the increased tolerance to errors comes with the excessive implementation overhead associated with the quantum T gate because the implementation of the fault-tolerant T-gate, including T gates and the conjugate transpose of the T gate ($T^\dagger$), is more costly than other Clifford+T gates [15]. As a result, the complexity of quantum circuits is also measured based on T-count and T-depth. T-count refers to the number of T gates in a circuit. T-depth indicates the

[1]A universal set of quantum gates

number of back-to-back T gates in the critical path of a circuit.

In the era of NISQ) devices, quantum processors are limited by qubit count, coherence times, and gate fidelity. These constraints make the implementation of exact arithmetic operations too noisy, as precise quantum circuits often require a large number of qubits and deep gate sequences, which amplify noise and reduce computational reliability. Approximate arithmetic provides a practical alternative by trading off a small amount of precision for substantial reductions in circuit depth, gate count, and overall qubit requirements. This approach aligns naturally with NISQ devices, enabling the execution of complex algorithms within the hardware's operational limits while still achieving meaningful computational results.

The other factors that impact the complexity of quantum circuits are ancilla and garbage. The design of a quantum circuit is different from a classical circuit. A quantum circuit should be reversible, which dictates a one-to-one mapping between inputs and outputs. Any constant input in a quantum circuit is called an ancilla. A garbage output refers to any output that exists to preserve a one-to-one mapping in a quantum circuit but is neither a primary input nor a useful output. In a NISQ device such as an IBM quantum computer, an ancilla can be reset and used for another qubit. This reduces the impact of ancilla on the complexity of quantum circuits. Thus, in this work, we mainly focus on reducing the number of required qubits, gate count, and circuit depth, and do not consider the number of ancillas in quantum circuits.

1.2 Motivations

The development of practical quantum computing faces two fundamental challenges: noise resilience and efficient resource utilization. Current quantum processors operate in the

NISQ era, where limited qubit counts, restricted connectivity, and error-prone gates significantly impact algorithm performance. To bridge the gap between theoretical models and realizable implementations, it is essential to design circuits and architectures that are robust against noise while maintaining computational efficiency.

To ensure that quantum circuits and architectures remain robust in realistic settings, it is essential to incorporate explicit error models into their design and evaluation. Common sources of errors in NISQ devices include depolarizing noise, readout errors, and gate imperfections, all of which can significantly degrade computational fidelity as circuit depth increases. By simulating and analyzing these error channels, designers can identify critical bottlenecks, optimize gate sequences, and develop noise-mitigation strategies tailored to the hardware. Integrating these models into the design of arithmetic circuits, Qunatum Neural Network (QNN) layers, and Quantum Random Access Memory (QRAM) architectures enables a more accurate assessment of performance and guides the creation of circuits that are both practically realizable and resilient under realistic operating conditions.

One of the most critical areas in this context is the design of quantum arithmetic circuits, which form the computational backbone for many quantum algorithms. Arithmetic operations such as addition, multiplication, and division are required in applications ranging from quantum simulation to cryptography. However, conventional arithmetic circuit designs are often too deep or resource-intensive to be executed reliably on NISQ devices. This motivates the exploration of optimized and noise-resilient arithmetic units that minimize gate counts, required qubits, and adapt to hardware constraints without compromising accuracy.

In parallel, the integration of Neural Network (NN) with quantum circuits has emerged as a promising direction for building hybrid quantum–classical systems. NNs provide powerful tools for pattern recognition, error mitigation, and adaptive learning, which can be

leveraged to enhance quantum processing. QNN models, in particular, offer the potential to achieve compact representations of high-dimensional data and to process information in ways that are intractable for classical models. This motivates research into architectures that embed neural-inspired mechanisms within quantum circuits to improve noise tolerance and generalization.

Another critical motivation comes from the concept of QRAM, which provides a mechanism for storing and retrieving data. QRAM circuits are central to algorithms that rely on large-scale data access, such as quantum machine learning. In quantum machine learning pipelines, QRAM plays a pivotal role by enabling fast access to large datasets, allowing quantum algorithms to exploit the parallelism inherent in quantum computation. This capability is essential for tasks such as quantum classification, clustering, and generative modeling, where repeated access to high-dimensional data is required. Yet, practical QRAM realizations remain challenging due to circuit depth, qubit requirements, and susceptibility to noise. Designing efficient, hardware-aware QRAM structures that balance scalability with resilience is therefore a key step toward enabling real-world quantum machine learning applications.

Overall, the motivation for this research lies in advancing the design of noise-resilient quantum arithmetic circuits, hybrid NN–quantum architectures, and practical QRAM models. Together, these elements form a foundation for building quantum computing systems that are not only theoretically powerful but also practically realizable on near-term hardware. By addressing the challenges of noise, depth, and scalability, this work seeks to contribute solutions that bring quantum computing closer to practical deployment in machine learning, optimization, and secure information processing.

1.3 Objectives

The primary objective of this research is to design and evaluate quantum computing models that are both noise-resilient and practically realizable on NISQ hardware. To achieve this, the work is guided by the following specific objectives:

1. Design of noise-resilient quantum arithmetic circuits

    - Develop optimized implementations of fundamental arithmetic operations (e.g., addition, multiplication, division, and square root) that utilize approximate computing to minimize circuit depth and gate count while maintaining the accuracy of operations on real quantum systems.
    - Incorporate hardware-aware techniques such as qubit reuse, gate reduction, and dynamic circuit generation to improve performance under realistic noise conditions.
    - Benchmark these circuits against conventional designs to demonstrate improvements in scalability and fidelity.

2. Develop practical and noise-resilient QNN models

    - Develop QNN architectures that exploit quantum parallelism while maintaining robustness against noise and propose a practical QNN that generates high accuracy in prediction.
    - Evaluate the effectiveness of QNN models in processing classical datasets and quantum-encoded information.

3. Develop practical and scalable QRAM circuits

- Propose noise-resilient QRAM architectures that enable efficient data storage and retrieval.
- Investigate circuit-level optimizations to reduce depth, improve gate efficiency, and adapt to hardware connectivity constraints.
- Validate QRAM-based models on benchmark tasks such as classification.

4. Evaluate performance on NISQ devices

   - Implement the proposed circuits and models using state-of-the-art quantum frameworks (e.g., Quantum Information Science Kit (QISKIT), PennyLane).
   - Assess performance through metrics such as circuit depth, gate counts, qubit count, and accuracy, comparing results across noiseless simulators and available noisy quantum hardware.
   - Demonstrate the applicability of proposed methods to practical tasks, highlighting their potential to bridge the gap between theoretical advances and hardware limitations.

1.4 Methodology

The methodology adopted in this research is structured to address the design, integration, and validation of noise-resilient quantum models that can be executed on NISQ hardware. The process begins with the design of optimized quantum arithmetic circuits. Arithmetic operations such as addition and multiplication are implemented with an emphasis on reducing circuit depth and minimizing the use of costly quantum gates. Hardware-aware techniques, including qubit reuse and measurement-and-reset strategies, are applied to

adapt the circuits to the architectural constraints of current quantum processors. These arithmetic units are then formally analyzed in terms of complexity and resilience to noise, and compared with existing baseline implementations to demonstrate improvements in scalability and fidelity.

Building on this foundation, the second phase focuses on the development of artificial NN–quantum hybrid models. In this stage, classical NNs are employed to support parameter optimization, adaptive learning, and error mitigation for quantum models. At the same time, QNNs are constructed by embedding neural-inspired structures into parameterized quantum circuits. These hybrid models leverage both classical optimization techniques and quantum parallelism, and are trained using frameworks such as PyTorch, PennyLane, and QISKIT. Their effectiveness is evaluated on benchmark datasets to assess performance in terms of generalization and noise tolerance.

The third component of the methodology addresses the design of practical Quantum QRAM circuits. QRAM serves as a crucial building block for large-scale quantum algorithms, enabling data storage and retrieval. In this work, QRAM architectures are designed with an emphasis on noise resilience and scalability. Circuit-level optimizations are applied to reduce depth and gate overhead, while embedding strategies such as amplitude and angle encoding are explored for efficient address–data mapping. The proposed QRAM models are validated through applications in classification, demonstrating their role as a functional memory resource within quantum systems.

Finally, the methodology incorporates extensive simulation, experimentation, and validation. All proposed designs are first implemented in simulation using Python-based frameworks, where noise models and qubit connectivity constraints are introduced to replicate the behavior of real NISQ hardware. Selected circuits are then deployed on available quantum devices to evaluate their performance under experimental conditions. Across

these experiments, performance metrics such as circuit depth, fidelity, mean-squared error, and classification accuracy are systematically measured and compared. This comprehensive evaluation ensures that the proposed methods are not only theoretically efficient but also practically realizable on current hardware, bridging the gap between abstract models and real-world quantum computing applications. Figure 1.1 illustrates the overall workflow adopted in this study, outlining the sequential methodology from quantum arithmetic design to noise-resilient QNN, QRAM construction.

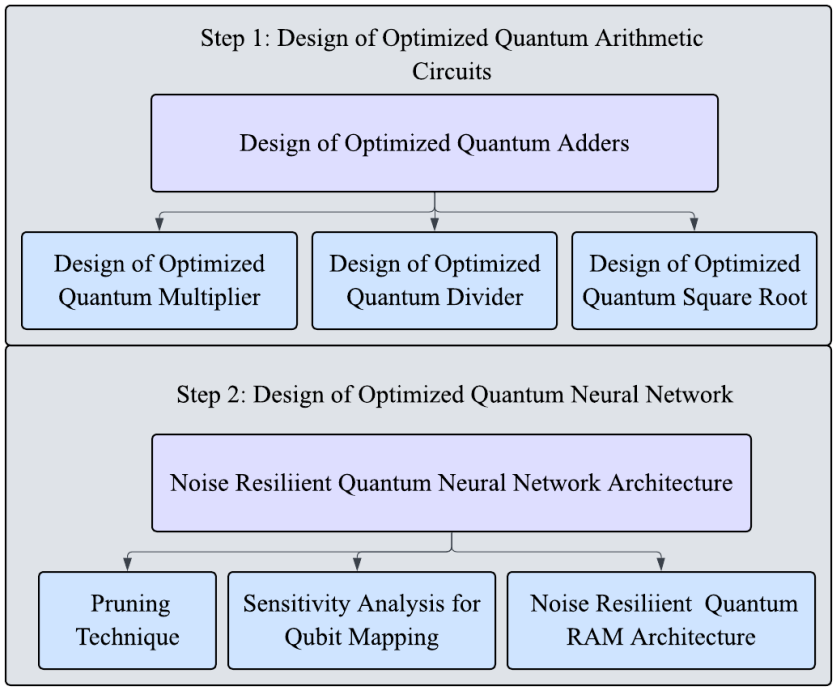


Figure 1.1: Overall workflow of methodology

1.5 Contributions

This work makes several contributions to the field of quantum computing, with a particular emphasis on noise-resilient circuit design, hybrid learning architectures, and practical memory models for NISQ devices.

First, this work introduces optimized implementations of fundamental quantum arithmetic circuits, including addition, multiplication, division, and square root. These designs reduce circuit depth and gate count through approximate computing and hardware-aware strategies such as qubit reuse, measurement-and-reset. Compared to conventional implementations, the proposed arithmetic circuits demonstrate improved scalability and fidelity, making them better suited for execution on near-term quantum processors.

The proposed quantum arithmetic circuits are superior to existing works concerning the number of qubits and quantum gates, T-count, and T-depth. We show that running approximate arithmetic circuits on a real quantum computer generates meaningful results with a negligible level of noise. Examples of applications are also presented to show the validity of the proposed approximate quantum circuits.

Second, the research contributes to the integration of NN with quantum circuits to support parameter learning, error mitigation, and adaptive optimization. A set of QNN architectures is proposed that embed neural-inspired structures into parameterized quantum circuits. These models are evaluated on benchmark datasets, demonstrating their ability to enhance noise resilience and generalization while leveraging quantum parallelism for efficient information encoding and processing.

Third, this work advances the design of QRAM architectures by proposing noise-resilient and hardware-adapted implementations. The developed QRAM circuits employ efficient address–data encoding strategies, controlled unitaries, and depth-reduction tech-

niques. These contributions provide a pathway toward scalable quantum memory structures that can be integrated into larger algorithms such as search, classification, and optimization.

Finally, the work provides a comprehensive experimental validation of the proposed methods. All circuit designs and models are implemented using state-of-the-art simulation frameworks, incorporating realistic noise models and device constraints. Selected implementations are further executed on real NISQ hardware, and their performance is evaluated using fidelity, mean-squared error, and classification accuracy. The results confirm that the proposed approaches achieve tangible improvements over baseline methods, demonstrating both theoretical novelty and practical feasibility.

1.6 Thesis Organization

The rest of this work is organized as follows.

Chapter 2 provides background information on quantum computing and reviews related work on arithmetic units, quantum neural networks, and memory models.

Chapter 3 begins with a review of the quantum circuit for an exact adder and then introduces the proposed designs for quantum adders, multipliers, dividers, and non-restoring square root circuits. These designs emphasize efficiency, scalability, and noise resilience.

Chapter 4 presents the details of the proposed noise-resilient QNN and QRAM architectures. The chapter explains their design principles, embedding strategies, and methods for mitigating the effects of noise on near-term quantum devices.

Chapter 5 illustrates the experimental validation of the proposed circuits and models. Simulations and real quantum hardware experiments are presented, and results are

analyzed in terms of hardware resources, accuracy, fidelity, and robustness against device noise.

Chapter 6 concludes the work by summarizing the main contributions, discussing limitations, and outlining promising directions for future research, including opportunities for extending noise-resilient designs and advancing scalable quantum memory structures.

1.7 List of Publications

1.7.1 Journal Papers

1. S. Sajadimanesh and E. Atoofian, "Implementation of a quantum division circuit on noisy intermediate-scale quantum devices using dynamic circuits and approximate computing," in Phys. Rev. A, vol. 109, no. 5, p. 052601, May 2024, doi: 10.1103/PhysRevA.109.052601.

2. S. Sajadimanesh, H. A. Rad, J. P. L. Faye, and E. Atoofian, "Inexact Quantum Square Root Circuit for NISQ Devices," in IEEE Access, vol. 12, pp. 125856–125870, 2024, doi: 10.1109/ACCESS.2024.3455997.

3. S. Sajadimanesh, H. Aghaee Rad, J. P. L. Faye, and E. Atoofian, "NR-QNN: Noise-Resilient Quantum Neural Network," in IEEE Access, vol. 13, pp. 40185–40197, 2025, doi: 10.1109/ACCESS.2025.3546956.

4. S. Sajadimanesh, R. Jahadi, and E. Atoofian, "Purifying Readouts in NISQ Devices Using Machine Learning," submitted for publication in Phys. Rev. A, 2025.

5. S. Sajadimanesh and E. Atoofian, "HaQmDc: Hardware-Aware Qubit Mapping Using Dynamic Circuit Solution," submitted for publication in ACM Trans. Quantum Comput., 2025.

1.7.2 Conference Papers

1. S. Sajadimanesh, J. P. L. Faye, and E. Atoofian, "Practical approximate quantum multipliers for NISQ devices," in Proc. 19th ACM Int. Conf. Comput. Frontiers

(CF '22), New York, NY, USA, 2022, pp. 121–130.

2. S. Sajadimanesh, J. P. L. Faye, and E. Atoofian, "NISQ-Friendly Non-Linear Activation Functions for Quantum Neural Networks," in 16th Int. Conf. Networking, Architecture, and Storage (NAS 2022), 2022.

3. S. Sajadimanesh, J. P. L. Faye, and E. Atoofian, "Optimization of Approximate Quantum Random Access Memory for NISQ Devices," in IEEE Int. Conf. Quantum Artificial Intelligence (QAI 2025), 2025.

4. S. Sajadimanesh and E. Atoofian, "Scalable QRAM with Superposition-Based Data Loading for Noise-Resilient Quantum Machine Learning on NISQ Devices," to appear in ACM Int. Conf. High Performance Computing in Asia-Pacific Region (HPCAsia), 2026.

Chapter 2

Background and Related Works

In this section, we introduce the fundamental principles of quantum computing, including qubit representation, superposition, entanglement, and the basic quantum gates that form the building blocks of all quantum algorithms used throughout this thesis. We then provide an overview of prior research related to quantum arithmetic units and QNNs, highlighting their design methodologies, computational advantages, and current challenges. This background establishes the foundation for the proposed approaches and situates our contributions within the broader landscape of quantum information processing.

2.1 Quantum Computing

2.1.1 Quantum Bits

In classical information theory, the basic unit for storing data is a bit. A bit is physically implemented using a two-state device and is commonly represented as either 0 or 1. The

quantum equivalent is named a quantum bit (qubit). It represents a two-level quantum mechanical system, often denoted as $|0\rangle$ and $|1\rangle$. These levels can be, for example, in two different polarizations of a photon (vertical and horizontal), two different alignments of spin (up and down), or two states of an electron orbiting an atom (ground state and excited state). They can also be defined by more complex systems based on very cold superconducting electrical circuits in which several electrons move [5],[1], [22].

Implementing physical qubits is fundamental for building a quantum computer. The problem is that, on one hand, these physical qubits need to be well isolated to preserve their features, but on the other hand, be accessible for the computational tasks and measurements [1]. Achieving a good balance of both requirements is the leading implementation challenge. Moreover, a general rule is that the decoherence time must be longer than the gate operation time. The term decoherence describes here the irreversible interactions of the qubits with the environment, also referred to as leakage [23], [24].

2.1.2 Superposition

Quantum superposition [1] is a phenomenon in quantum mechanics where a quantum system can exist in multiple states or configurations simultaneously, until it is measured or observed, at which point it will collapse into a single state. In other words, a quantum particle such as an electron can exist in multiple positions or energy states at the same time, with each possibility represented by a probability amplitude. In contrast, the state of a classical bit can be either 0 or 1. The quantum states can be represented by two orthonormal basis vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

A qubit can then be described as a normalized vector $|\psi\rangle$ in a two-dimensional complex Hilbert space $\mathcal{H} = \mathbb{C}^2$ [1]. We can express such a state as a superposition of $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.2}$$

where parameters α and β are complex probability amplitudes that fully describe the qubit. Their squared magnitudes give the measurement probabilities, where $|\alpha|^2$ is the probability of obtaining the state $|0\rangle$ and $|\beta|^2$ is the probability of obtaining the state $|1\rangle$, with the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The magnitudes and relative phase of α and β determine the qubit's location on the Bloch sphere and its overall quantum behavior.

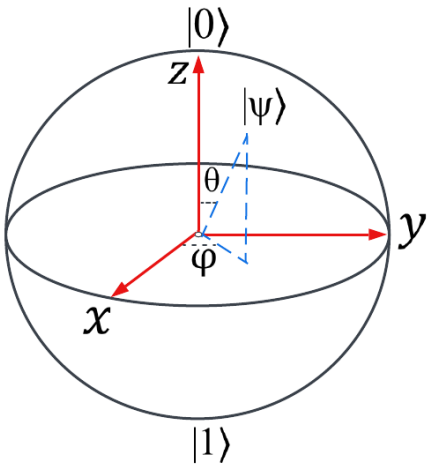


Figure 2.1: Bloch Sphere

In quantum mechanics, the Dirac notation, read as bra $\langle .|$ and ket $|.\rangle$, is the standard notation for states. Here $\langle\psi|$ denotes the conjugate transpose of the vector $|\psi\rangle$. The eponymous bra-ket $\langle\psi_1|\psi_2\rangle$ expresses the scalar product of the vectors $\langle\psi_1|$ and $|\psi_2\rangle$. This description only includes pure states. Since the description of a pure qubit state is a vector with norm 1, we can depict such a state on the surface of a sphere. The visualization of quantum states with two numbers θ and ϕ, depicted in Figure 2.1, is named the Bloch sphere, where in equation 2.2, $\alpha = \cos\left(\frac{\theta}{2}\right)$ and $\beta = e^{i\phi}\sin\left(\frac{\theta}{2}\right)$.

2.1.3 Entanglement

Another fundamental feature that distinguishes quantum computing from classical computing is entanglement [1]. Both superposition and entanglement are essential resources in quantum algorithms. While superposition can be defined for a single particle, entanglement involves two or more qubits. Specifically, a pair or set of qubits is said to be entangled if the state of each qubit cannot be described independently of the others. To illustrate this concept, consider the simplest example of entanglement: a pure two-qubit state known as a Bell state. The Bell states consist of four special two-qubit quantum states that are maximally entangled. This means that the two qubits are correlated in such a way that the state of each qubit cannot be fully described without reference to the other. These states form an orthonormal basis for the two-qubit Hilbert space, commonly referred to as the Bell basis or maximally entangled basis.

$$
\begin{aligned}
|\Phi^+\rangle &= \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big) \\
|\Phi^-\rangle &= \frac{1}{\sqrt{2}}\big(|00\rangle - |11\rangle\big) \\
|\Psi^+\rangle &= \frac{1}{\sqrt{2}}\big(|01\rangle + |10\rangle\big) \\
|\Psi^-\rangle &= \frac{1}{\sqrt{2}}\big(|01\rangle - |10\rangle\big)
\end{aligned}
\tag{2.3}
$$

After measurement, both qubits are either in the state $|0\rangle$ or in the state $|1\rangle$. When measuring only one qubit, the other qubit will collapse into the same state. In this sense, the two qubits are entangled. In general, we name a quantum state of two or more systems entangled if it is not separable. Further, we name a pure state of n subsystems separable if it can be written in the form of:

$$
|\psi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle \tag{2.4}
$$

Where $\otimes$ is the tensor product operation.

2.1.4 Measurement

Similar to the classical bits, there are two possible outcomes for the measurement [1] of a qubit (0 or 1). After measurement, a qubit state collapses in the pure state $|0\rangle$ or $|1\rangle$. It is possible to compute the probabilities of getting a special measurement outcome with probability:

$$
p_0 = \langle\psi|0\rangle\langle 0|\psi\rangle = |\langle 0|\left(\alpha|0\rangle + \beta|1\rangle\right)\rangle|^2 = |\alpha\langle 0|0\rangle|^2 = |\alpha|^2 \tag{2.5}
$$

P_0 indicates that the probability of the measured value will be 0. We call $M_0 = |0\rangle\langle 0|$ a measurement operator. Analogous we get $p_1 = |\beta|^2$ with the operator $M_1 = |1\rangle\langle 1|$. To summarize, a qubit can be in an infinite number of possible states, but after the measurement, we receive binary information similar to the classical bits, and the superposition information is lost. However, with quantum algorithms, this fundamental property of quantum mechanics can be used effectively and is the reason for quantum speedup.

2.1.5 Density Matrix

In quantum computing, the state of a quantum system is commonly described by a *state vector* $|\psi\rangle$ for pure states. However, in realistic scenarios, quantum systems often interact with their environment, leading to *mixed states* that cannot be represented by a single state vector. The *density matrix* formalism provides a complete description of both pure and mixed quantum states.

For a pure state $|\psi\rangle$, the density matrix is defined as:

$$\rho = |\psi\rangle\langle\psi| . \tag{2.6}$$

For a mixed state, where the system is in state $|\psi_i\rangle$ with probability p_i, the density matrix generalizes to:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| , \quad \sum_i p_i = 1. \tag{2.7}$$

The density matrix contains all observable information of the quantum system. The expectation value of an observable O can be computed as:

$$\langle O\rangle = \mathrm{Tr}(\rho O), \tag{2.8}$$

where Tr(·) denotes the matrix trace. This formalism is essential for modeling realistic quantum computations on NISQ devices, as it captures the effects of noise, decoherence, and partial measurements, allowing both pure and mixed states to be analyzed within a unified framework.

2.2 Quantum Gates

In quantum circuits, the state of the qubits is manipulated by quantum gates. A quantum gate is modeled using a unitary matrix. The matrix is applied to the input qubit to convert the state of the input to the desired state. In this work, we use a family of quantum gates, called Clifford+T gates [1], to implement the proposed circuits. The Clifford+T gates represent a universal set of quantum operations that can be used for the synthesis of any quantum circuit. The Clifford+T gates are similar to NAND gates in the classical domain, where any classical digital circuit can be synthesized into a set of NAND gates. Quantum gates used in quantum circuits are categorized into two groups: i) single-qubit gates and ii) multi-qubit gates, which are modeled by a matrix. In single-qubit gates, the matrix is applied to the input qubit to convert the state of the input qubit to the desired state. On the contrary, in a multi-qubit gate, two or more qubits are involved in the operation. Figure 2.2 shows the subset of Clifford+T [1] gates and the list of single and multi-qubits used in this work.

- **NOT gate (X gate):** A NOT gate flips the state of the input qubit. If the state of the qubit is $|0\rangle$, the gate converts it to $|1\rangle$ and vice versa.
- **Hadamard gate:** A Hadamard gate sets the state of a qubit to the superposition of $|0\rangle$ and $|1\rangle$. Superposition provides parallelism in quantum circuits, which is one

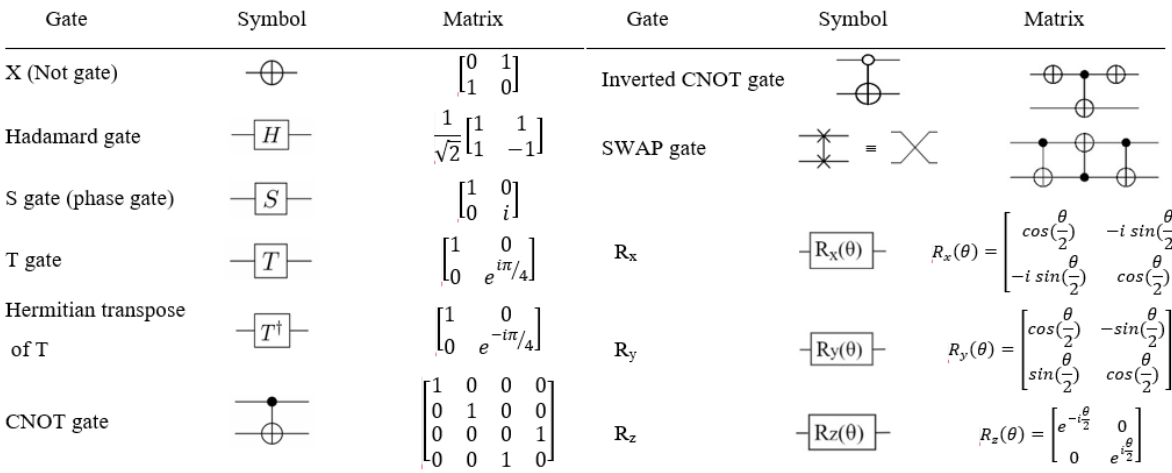


Figure 2.2: Quantum gates and their corresponding matrix representations [1].

of the main advantages of quantum computers over classical computers. This gate turns $|0\rangle$ into $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ into $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

- **S gate:** An S gate is not part of the Clifford+T library. An S gate is similar to a T gate ($S = T^2$); thus, it can be decomposed into two T gates in a row.

- **T gate:** A T gate is a single-qubit quantum gate with an input and an output. The gate does not change the input state if it is equal to $|0\rangle$. However, if the input state is $|1\rangle$, then the T gate performs a rotation by $\frac{\pi}{4}$, thus adding a phase.

- **CNOT/ Inv-CNOT gate:** A CNOT gate is a 2-qubit gate. The first input acts as a control qubit that determines the state of the target (second) qubit. If the control qubit is in state $|0\rangle$, the gate does not change the inputs. However, if the control qubit is in state $|1\rangle$, the state of the second input is flipped. An inverted CNOT gate is similar to a CNOT gate, but its operation is opposite. It flips the state of the target qubit when the control qubit is in state $|0\rangle$; otherwise, it does not change the state of the target.

- **SWAP Gates:** A SWAP gate is a 2-qubit gate and is used to swap the state of two qubits, unconditionally. SWAP gates are commonly used in quantum circuits when logical qubits are mapped to physical qubits. If the physical qubits corresponding to two neighboring logical qubits are not physically adjacent, a sequence of SWAP gates is employed to bring the two physical qubits next to each other [1].

- **Rotation Gates ($R_x(\theta), R_y(\theta), R_z(\theta)$) :** Rotation gates are single-qubit gates where the state of the input is rotated by an angle θ around X-, Y-, or Z-axis. For example, $R_x(\theta)$ rotates the qubit state around the X-axis by θ. In contrast to other quantum gates, which are parameter independent, the operation of rotation gates can be changed through the angle (θ). As we will see later, this property of rotation gates plays an important role in QNNs. These gates are typically parametrized by continuous angles, which act as trainable parameters within the network. The selection and initialization of these parameters directly affect the QNN's trainability and stability. Carefully designed parametrization schemes can improve gradient propagation and enable more efficient optimization. By appropriately controlling the rotation angles during training, the QNN can explore the Hilbert space, allowing it to approximate complex target functions while maintaining stable and effective learning dynamics.

2.2.1 Toffoli Gate

The Toffoli gate is a universal gate, which means any quantum circuit can be expressed as a sequence of Toffoli gates along with single-qubit gates. A Toffoli gate, also known as a controlled-controlled-NOT (CCNOT) gate, is a quantum gate that operates on three qubits and performs a bitwise exclusive-OR (XOR) operation on the last two qubits only when the first qubit is in the state $|1\rangle$. In addition, the Toffoli gate performs a conditional NOT

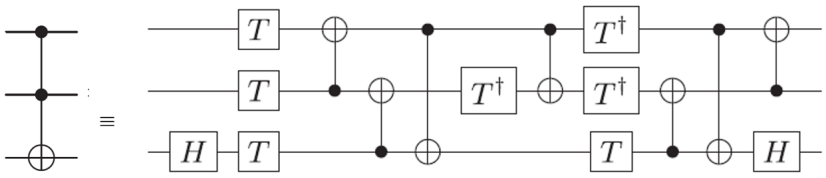


Figure 2.3: Symbol of the Toffoli gate and its implementation with Clifford+T gates (T-count of 7 and a T-depth of 3) [25].

operation on the third qubit, where the first two qubits act as control qubits. The logical AND operation using a Toffoli gate can be easily implemented by initializing the state of the third qubit into the state $|0\rangle$. Under this condition, the output state of the third qubit would be the AND of the first two qubit states. One use case of the logical AND operation is the calculation of carry in quantum full adder circuits. In quantum computing, to satisfy the fault-tolerant implementation of the Toffoli gate, it is decomposed into Clifford+T gates. The most cost-efficient decomposed version of the Toffoli gate is proposed in [25], where it has the T-count of 7 and T-depth of 3 (Figure 2.3).

2.2.2 Logical AND Gate

One of the factors that impacts the complexity of quantum circuits is the number of T gates. Implementation of a fault-tolerant T gate is significantly more costly than other Clifford+T gates [26]. As a result, the runtime of a circuit is quite often dominated by T gates. T-count is the total number of T gates and the Hermitian transpose of T gates in a quantum circuit. Reducing the number of T gates in a circuit directly impacts the cost of implementation. Many quantum applications, such as Shor's algorithm, Grover oracles, arithmetic units, etc., require some form of an AND gate for implementation. Figure 2.4(a) shows a low-cost and implementation-efficient AND gate circuit [26] with a T-count of 4.

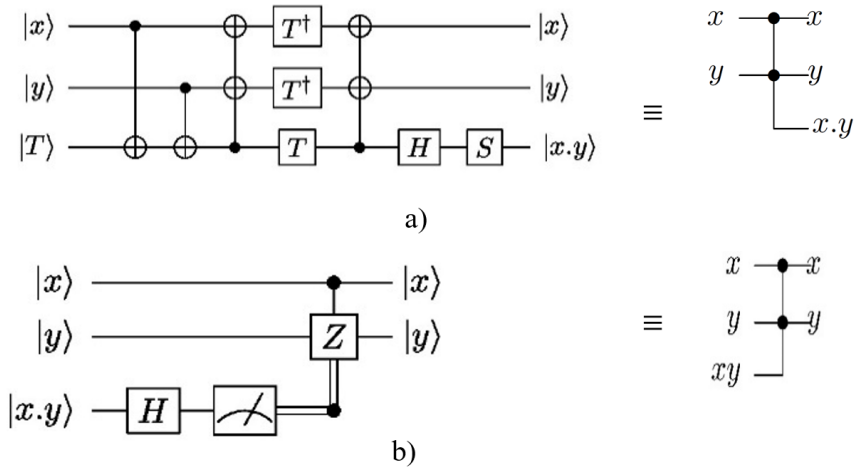


Figure 2.4: (a) A logical-AND gate and its Clifford+T gate implementation with T-count of 4. The T gate needed to produce $|T\rangle$ input is included in the T-count [26], (b) Circuit for uncomputation AND gate [26].

The input $|T\rangle$ is an ancilla in the state of $\frac{(|0\rangle+|1\rangle)}{\sqrt{2}}$.

This circuit is more cost-effective in terms of T-count and T-depth compared with a decomposed version of the Toffoli gate. We use this circuit in the proposed arithmetic units. 2.4(b) shows the circuit for uncomputation and the corresponding symbol.

2.3 NISQ Hardware

Several technologies exist for realizing quantum computing on hardware, such as superconducting [27], ion traps [28], quantum dots [29], and neutral atoms [30]. In this work, we focus on superconducting quantum processors because they currently represent the most mature and widely scalable qubit platform, offering high-fidelity gate operations, fast qubit control, and well-established fabrication techniques. As the leading technology behind many state-of-the-art NISQ devices, superconducting qubits provide a practical and

experimentally validated testbed for implementing and evaluating quantum algorithms.

In this technology, quantum gates manipulate the qubits by applying microwave pulses. The implementation of two-qubit gates, such as CNOT, is based on the cross resonance effect, in which a pulse is applied to the control qubit at the resonant frequency of the target qubit [27], [31]. Quantum computers built by IBM are based on superconducting qubits. IBM contributes to the development of open-source QISKIT [32] and provides access to its superconducting quantum computers via IBM Quantum Experience, allowing researchers, developers, and organizations to experiment with quantum computing through separate public and private cloud services [8]. It is important to note that while in this work, we use superconducting-based NISQ devices, our proposed circuits can be deployed in quantum computers based on other technologies [28], [29], [30].

2.4 Dynamic Circuit

Recently, IBM provided a dynamic circuit in its quantum computers. In a nutshell, a dynamic circuit supports mid-circuit measurement and mid-circuit reset. Thus, a quantum computer equipped with the dynamic circuit capability enables a programmer to reuse a qubit that is not needed for the next operations in the quantum circuit. Dynamic circuits offer several key advantages, including significant qubit savings, a reduction in SWAP gate overhead, and improved overall circuit fidelity [33].

Figure 2.5(a) shows a dynamic circuit that involves a measurement operation to read a qubit and a reset operation to force the state of the measured qubit back to the ground state.

The built-in reset operation in IBM NISQ devices implicitly involves measurement

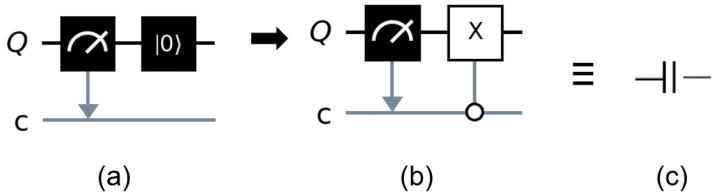


Figure 2.5: (a) Dynamic circuit with measurement and reset. (b) A dynamic circuit with measurement and a classically controlled CNOT gate. (c) Dynamic circuit symbol

pulses, which is a costly operation. Figure 2.5(b) shows a dynamic circuit with the same functionality but without any reset operation [33]. It is composed of a measurement and a classical/quantum CNOT. The control terminal of the CNOT gate is generated by the measured value. If the measured value is 1, then the CNOT flips the state of the qubit; otherwise, the state remains unchanged. The circuit in figure 2.5(b) cuts the execution time of the circuit in figure 2.5(a) by half [33]. For simplicity, we use the notation shown in figure 2.5(c) to represent a dynamic circuit.

2.5 Approximate Computing

There have been numerous research works on approximate computing for classical computers. Gupta *et al.* [34] utilize approximate computing to reduce the complexity of adders at the transistor level. They simplify the circuit of an adder by removing transistors and provide power savings over conventional exact adders. They offer different approximate adders to trade off power vs. accuracy. The proposed approximate adders are used to design approximate multipliers. To maintain a reasonable output quality in multimedia applications, they approximate only the first 9 bits of an Adder. For the rest of the bits, they use exact circuits. Akbari *et al.* [35] propose four adder circuits and use them to build

approximate multipliers. The adders have the flexibility of switching between exact and approximate operating modes in order to provide reconfigurability in multipliers during runtime. A maximum power consumption constraint is set such that if the power of multiplication exceeds this amount, then the multiplication switches from the exact operating mode to the approximate one. In the approximate mode, these adders provide higher speed and lower power consumption at the cost of lower accuracy. Liu *et al.* [36] use majority logic as building blocks for approximate adders and multipliers. The proposed multiplier uses a reduction circuitry with complement bits to compute the sum of partial results. An influence factor is defined to assess the importance of complement bits. The influence factor helps to decide on the level of approximation based on the trade-off between accuracy and complexity of an approximate multiplier. All the above works on approximate computing are evaluated in the context of classical computers. However, to the best of our knowledge, this is the first work on using approximate circuits to implement a quantum arithmetic unit, including adders, multipliers, and square root algorithms on real quantum computers, and generate meaningful results.

The analogy between classical approximate computing and quantum approximations lies in their shared principle of error-tolerant computation. In classical circuits, small inaccuracies are intentionally introduced to reduce complexity, power consumption, or execution time, while still producing acceptable results. Similarly, in quantum computing, circuit simplification and approximation reduce circuit depth and gate count, which mitigates the effects of noise and decoherence on real quantum devices. Although these approximations may decrease the theoretical accuracy of the algorithm, they improve the overall fidelity of the computation, producing outputs that more closely match expected results. This perspective highlights how controlled approximations can be leveraged as a practical strategy to overcome hardware limitations in both classical and quantum computing.

2.6 Related Works on Arithmetic Circuits

Different types of designs have been proposed by researchers for quantum multiplications, as multipliers are the bulk of computation in many applications, such as digital signal processing and quantum cryptography. However, the majority of prior works target reversible computing from a theoretical point of view and suffer from significant noise levels once they are implemented on NISQ devices. Multipliers that exploit the quantum Fourier transform, such as [37], [38], have a high number of Clifford+T gates. For example, the quantum multiplier design proposed by [38] has a T-count of $O(n^3)$.

Jayashree et al. [16] proposed a quantum multiplier that uses swap gates to rotate partial products. The rotate operation is performed in two phases. Each phase uses 2-input swap gates to swap the position of pairs of qubits. The proposed rotation operation avoids sequential shifting of qubits by arranging the swap gates in a way that they can operate in parallel. The carry path is composed of controlled swap gates, and the sum outputs are computed by Toffoli gates. Jayashree et al. [16] focus on optimizing garbage and ancilla and do not address the depth of the circuit.

Lin et al. [18] use controlled adders in a quantum multiplier. The structure of the adders is based on a ripple-carry adder and uses majority modules for the generation of carry and sum. Each majority module in the multiplier circuit is controlled by a bit of the multiplier. The majority module is built out of two Toffoli and one CNOT gates. To shift partial products, the multiplier relies on a set of SWAP gates. However, the swap gates are connected serially, which increases the depth of the circuit by $O(n^2)$.

The quantum multiplier proposed by Babu [39] generates partial products using a decomposed form of the Toffoli gate to perform a bitwise-AND operation. The partial products are added using a tree structure. In each level of the tree, the partial products

are added column-wise to produce the sum and carry. The proposed multiplier suffers from excessive noise on real quantum computers as it uses non-trivial quantum gates, which increase the complexity of the synthesized circuit. In addition, Babu [39] did not consider the cost of T gates used for the implementation of the Toffoli gate to calculate the cost of their proposed circuits.

Coreas and Thapliyal [17] used a controlled adder to design a quantum multiplier. The amount of garbage in the controlled adder is reduced using Bennett's garbage removal scheme [40]. The adder uses the second input operand as a temporary register for the generation of carry qubits. Then, it restores the values of the second operand using uncomputation blocks. While the proposed multiplier requires fewer T gates than previous works [16], [18], [19], it suffers from high depth, which causes noisy outputs on NISQ devices.

Kotiyal et al. [19] proposed a binary tree-based design for a quantum multiplier. The proposed multiplier performs the addition of partial products in parallel using a ripple quantum adder. First, using a set of Toffoli gates, partial products are generated. In the next step, pairs of partial products are added following the structure of a binary tree to generate results. This technique suffers from high T-count as well as high depth, which makes it impractical on real quantum computers.

Unlike classical computing, research on quantum Quantum Square Root (QSQR) circuits is still in its early stages. Quantum circuits face the challenge of requiring reversible logical components to ensure one-to-one mapping between the input and output states using unitary matrices [41]. A reversible QSQR circuit has been designed for an 8-bit input [42], but it is limited to fixed input sizes and lacks comprehensive error analysis, raising doubts about its practicality.

Another proposal for a reversible QSQR circuit [43] claims scalability for various input sizes; however, the essential internal module details remain undisclosed. This hinders other researchers from understanding, replicating, or advancing the work, and the lack of a comprehensive error analysis raises concerns about its suitability for practical applications.

Bhaskar et al. [44] proposed a quantum algorithm for QSQR operation using multipliers. The algorithm uses a function whose root is the QSQR of an input. To compute the root of the function, the algorithm first assigns approximate values to the parameters of the function. Subsequently, through a series of iterations, the root of the function is computed using Newton's method [45]. The accuracy of the results depends on the number of iterations. The main drawback of this technique is that it is impractical because it requires a large number of multipliers and adders. The implementation necessitates $5 \times \lceil \log_2(n) \rceil$ multiplications and $3 \times \lceil \log_2(n) \rceil$ additions, where n is the size of input. This results in substantial T gate and qubit costs. Running such a circuit on a NISQ device causes excessive noise, and it is not feasible to extract any meaningful result from the circuit.

Sultana et al. [43] used an array of Reversible Controlled Adder and Subtractor (RCAS) blocks based on classical structure presented in [46] to build a QSQR circuit. Due to the dependency of the RCAS blocks, the critical path in the proposed circuit is long, which limits the scalability of the circuit on real quantum computers. The other shortcoming of the circuit is the large number of quantum gates. As an example, a 4-qubit QSQR requires 6 RCAS modules with 12 Peres gates and 84 T gates. Such a large circuit causes excessive noise on NISQ devices and is not practical. In addition, they modeled their proposed circuit in VHDL and did not test it on a real quantum computer.

AnanthaLakshmi and Sudha [47] proposed a QSQR circuit utilizing a Reversible Controlled-Subtract-Multiplex (RCSM) block. Their proposed circuit is based on the non-restoring square root algorithm. An RCSM uses a control signal to either subtract two

inputs or select one of the inputs and assign it to the output. For an n-qubit QSQR, the proposed circuit requires an order of $O(n^2)$ qubits, which compromises the scalability of the circuit on NISQ devices. In addition to quantum computing, reversible logic has applications in energy-efficient Very Large-Scale Integration (VLSI) circuits.

Swathi et al. [48] proposed a QSQR circuit based on reversible multiplexers and subtractors to reduce the number of gates. In addition, they modeled their proposed circuit in VHDL and did not test it on a real quantum computer.

Dutta et al. [49] designed a reversible circuit for computing the square roots of real numbers. For fixed-point numbers, an iterative approach [50] is used, where the input number should be between zero and three, and the algorithm converges best when the input number is close to one. For floating-point computations, the algorithm is only applicable to positive numbers. The circuit first adjusts the exponent to convert it to an even number and then applies the fixed-point method to the adjusted values. The circuits aim to reduce the number of qubits and T-count. However, they used a Cuccaro adder with a T-count of $14n - 7$ and a fused shift and multiply circuit with a T-count of $28n^2 - 21n$. The choice of these arithmetic blocks in the QSQR increases the cost of the circuit and makes it impractical for NISQ devices.

Coreas and Thapliyal [51] proposed the design of a QSQR circuit optimized for the number of qubits and T-count. The quantum circuit requires $2n + 1$ qubits to calculate the square root of an n-qubit number. The proposed circuit relies on the Toffoli and Peres gates, which leads to a large overhead in terms of T-count, particularly for large integer values. Although they proposed a quantum circuit that is more efficient than prior works [44], [43],[47], their quantum circuit is not practical on a real quantum backend since it generates results with too much noise.

In contrast to the above works, our proposed approximate QSQR circuit is optimized for NISQ devices, and generates correct results on contemporary quantum computers in the presence of quantum noise. Numerous efforts have been made to optimize square root algorithms for classical computers [52], [53], [46]. Li and Chu [52] proposed a parallel-array implementation of a non-restoring square root algorithm. The algorithm utilizes simple arithmetic units, such as adders and shifters, for its ASIC implementation. The architecture of the square root operation is pipelined to achieve a throughput of one operation per cycle. The proposed architecture enhances instruction-level parallelism in modern multi-issue processors, as they rely on pipelined functional units to accelerate program execution.

Samavi et al. [46] proposed a modular structure for a non-restoring square root algorithm that reduces the number of logical gates in the ASIC design. Jiang et al. [53] proposed an approximate square root algorithm for classical computers to reduce latency and power. They prune low-order bits of radicand to reduce the number of arithmetic operations required to compute the square root of a number. In addition, the authors proposed removing zeros at bit positions higher than the most significant one. The leading zeros unnecessarily increase the number of steps required to compute the square root, wasting processor cycles and power. Furthermore, some arithmetic units are simplified by removing/reducing transistors and logical gates required for the implementation. Our work is different, as we focus on the quantum implementation of square root operations utilizing approximate computing. Approximate computing is a computational paradigm that involves sacrificing accuracy in favor of improving performance and/or energy [54],[55], [56], [57], [34]. This trade-off is driven by the recognition that many prevalent and emerging applications can produce outputs of acceptable quality, even when there is a certain level of inexactness or approximation in the computations. For example, applications that process noisy inputs, such as data collected from sensors, exhibit an inherent resilience to inexact-

ness. They can tolerate a certain degree of error or approximation without significantly affecting the overall quality or reliability of the output. This error-resilient nature has given rise to a set of techniques known as approximate computing, which leverages the inherent tolerance to the inexactness of certain applications to boost performance and/or energy. This computational paradigm is well-suited for applications resilient to inexactness, and it introduces a new dimension in the design space that offers designers greater flexibility in optimizing classical computing systems.

Kumar et al. [58] proposed an adaptive approximation method for classical square root circuits. They replaced the exact subtractor cells in the square root circuit with approximate cells that use simple AND gates instead of multiplexers. The proposed circuit is modeled in Verilog and simulated for ASIC chips. The simulation results show power reduction with a marginal impact on accuracy.

Arya et al. [59], [60] proposed approximate methods for computing square roots using a restoring array. The approximation stems from inexact subtractor cells. The paper proposed two variations of approximate square root circuits. In the first one, some inputs are connected directly to the outputs, removing those subtractor cells that exist on the path from the inputs to the outputs. The second version employs bit truncation to eliminate corresponding subtractor cells. These approaches aim at decreasing switching activity at subtractor cell nodes, thereby reducing dynamic power dissipation.

Bandil and Nagar [61] proposed a reconfigurable architecture for a square root circuit based on the restoring square root algorithm. The design incorporates elimination, optimization, and simplification of subtractors to inject inexactness into the square root operation. Configurable subtractor cells are integrated into the circuit, enabling dual operation modes for both error-resilient and error-sensitive applications. They modeled the designs using Verilog, offering a low-power and area-efficient design compared to the exact

circuit. Unlike classical computing, there have been limited studies [62] on utilizing approximate computing to simplify complex quantum circuits for quantum adder and multiplier and generate practical and cost-effective designs for contemporary NISQ devices.

Our research aims to show that introducing approximations to specific components in an exact square root circuit can be highly beneficial. By carefully managing these approximations, we simplify the circuit design, making it more suitable for implementation on existing quantum hardware. The use of approximate computing techniques in quantum square root circuits has the potential to lead to significant advancements in quantum computing. By reducing the complexity and resource requirements, we can bring theoretical algorithms closer to practical implementation on NISQ devices. This research is a step towards fully harnessing the power of quantum computing, thereby enabling the development of efficient and practical quantum algorithms across different fields.

2.7 Conventional QNNs

Machine learning is a data-driven decision-making approach in which a program fits a mathematical model to data during the training phase and uses the trained model to derive decisions during the inference phase. Quantum Machine Learning (QML) is a promising application domain that is able to exploit quantum computing and offer acceptable results in the near future. Numerous QML models have been presented in the literature [63],[64]. The core of a QML is a QNN. QNNs have been employed in a variety of applications such as image classification [65], drug discovery [66], finance [67], etc. A typical QNN is composed of a data encoding circuit and a computational part, followed by a measurement operation (Figure 2.6).

QNN offer several potential advantages over classical neural networks due to the funda-

mental properties of quantum information. A key benefit is the ability of quantum circuits to represent data in exponentially large Hilbert spaces. With only n qubits, a QNN can encode and manipulate 2^n complex amplitudes, enabling a level of expressivity that would require exponentially more neurons or parameters in a classical network. This compact representation is especially powerful when dealing with high-dimensional datasets, structured quantum states, or problems that naturally map to quantum mechanics.

Another strength of QNNs lies in their ability to generate and utilize entanglement, which creates correlations among qubits that cannot be reproduced by classical systems. Entanglement allows QNNs to capture global relationships across features—even with shallow circuits—without requiring deep architectures or large parameter counts. This capacity to model globally correlated structure with minimal depth can lead to faster convergence and reduced model complexity compared to classical networks that must approximate these dependencies through many layers of computation.

QNNs also provide a natural advantage when the data itself originates from quantum processes. In fields such as quantum chemistry, materials science, or quantum simulation, classical preprocessing requires flattening quantum states into classical vectors, which may be exponentially costly or lossy. QNNs can circumvent this limitation by processing quantum-native data directly, maintaining the structure of the quantum system and avoiding expensive classical encoding steps. This makes QNNs particularly suitable for hybrid quantum-classical workflows.

Moreover, QNNs inherently operate within a probabilistic computational framework, where measurement outcomes follow quantum probability distributions. Unlike classical models that rely on explicit stochastic layers or regularizers, QNNs benefit from built-in randomness that can improve generalization and reduce overfitting. This probabilistic behavior, together with the limited number of trainable parameters and the shallow circuit

depths typical of the NISQ era, often leads to robust models that perform well even in the presence of hardware noise. Empirical results, including those reported in this work, demonstrate that shallow QNNs can achieve competitive accuracy and converge rapidly on real quantum devices.

Finally, QNNs have theoretical potential for computational advantages in tasks involving linear algebra, high-dimensional geometry, and structured data transformations. Quantum circuits can implement certain operations—such as large-dimensional rotations or state transformations—with resource scaling that may outperform classical methods under specific conditions. Although full quantum advantage in machine learning remains an open research question, early demonstrations provide strong motivation to consider QNNs as promising candidates for future large-scale learning systems

2.7.1 Encoding Data

Mapping classical data into qubits is a non-trivial task. A critical example for machine learning applications is the processing of big data. While from a theoretical point of view, there is no obstruction to uploading a large amount of data into qubits, it is not clear how to do it on NISQ devices while keeping the noise of quantum circuits under control.

One approach to encode classical data is amplitude encoding [68], [69]. This approach requires an efficient way to prepare and access these amplitudes. Each qubit can be in a superposition of states $|0\rangle$ and $|1\rangle$, with coefficients represented by complex numbers with real and imaginary parts. Thus, two classical values can be encoded using a qubit. This process can be extended by creating tensor products of multiple qubits, which is able to cover an exponentially-sized space in the classical domain. QNNs equipped with amplitude encoding require non-trivial quantum components that lead to resource-intensive circuits,

which cannot be implemented on small-scale devices [70]. State preparation is not a trivial procedure and requires costly quantum circuits with a high number of quantum gates as well as large circuit depth. Access to the states that encode classical data can be done using a quantum random access memory [71]. However, this is experimentally challenging as the construction of a QRAM is still under development.

An alternative approach for data encoding is rotation or angle encoding [72]. In the simplest case, classical data are uploaded as a rotation of qubits in Hilbert space. A single qubit is insufficient to create a universal data encoding. A single qubit has two degrees of freedom, thus allowing for the representation of data in two-dimensional space. To support classical data with higher dimensions, two approaches can be used. One approach would be to use n qubits to encode n classical features. For example, Rz(θ) on a qubit in superposition is used to encode a classical feature (Figure 2.7). The Hadamard gate in Figure 2.7 is used to put the qubit in a superposition state. As the state generated by a rotation gate along any axis repeats itself every 2π, features are generally scaled within 0 to 2π (or π to $-\pi$) during the data pre-processing step. Another approach would be applying back-to-back quantum rotation gates to a single qubit. For each feature, an angle is computed and used within a rotation gate. The downside of this approach is that the depth of the circuit increases with the number of features. The two approaches can be mixed to create an angle encoding scheme where n_1 qubits are rotated n_2 times each, to model n features in the classical domain ($n = n_1 \times n_2$).

2.7.2 Processing of Quantum States

After quantum encoding, the quantum states are transformed by the computational part of the QNN. The task of a QNN that we use in this work is to solve a supervised pattern

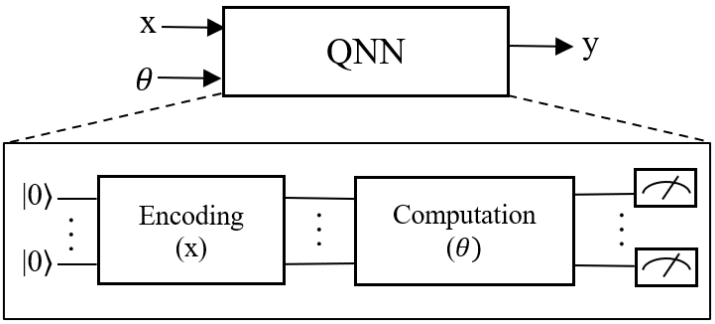


Figure 2.6: Structure of a QNN

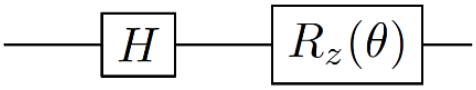


Figure 2.7: A feature in the classical domain is encoded by a qubit.

recognition problem, which is a standard problem in machine learning with applications in image recognition, fraud detection, healthcare, etc. A quantum classifier receives a quantum register as input and classifies the corresponding classical data. If the circuit of the quantum classifier includes only non-controlled gates, then it can exhaust only a small subset of the Hilbert space. In other words, the set of quantum states that the quantum classifier can reach is limited. Similar to a classical NN, the challenge of finding a generic QNN architecture is therefore to engineer a circuit that creates a powerful classifier for a classification problem. One approach for the implementation of the computational part in an NISQ device is using multiplication-and-accumulation (MAC) of inputs and network parameters, similar to classical NNs. However, as we will show later in this work, MAC operations are not practical on NISQ devices as they require a large number of quantum gates, which increases the depth of circuits and causes too much noise at the outputs.

A natural method to the problem of QNN design is to consider circuits that exploit

entanglement among quantum states. Such a circuit is a Parameterized Quantum Circuit (PQC) that is based on quantum rotation gates and can reach a wide corner of the Hilbert space using entanglement. Entanglement in quantum circuits is similar to correlation in classical circuits. A neuron that is connected to two inputs exploits the correlation between the two and increases the accuracy of classification. If no correlation exists between the two, then the weight corresponding to the two-input links is zero or close to zero. They can be removed safely without impacting the accuracy of the classifier. Similarly, in the quantum domain, if two features do not correlate, then removing entanglement between the two has no impact on the accuracy of the corresponding quantum circuit.

Entanglement in a quantum circuit can be implemented using controlled gates such as CNOT. Entanglement can be considered as a solution for the limitation established by the no-cloning theorem. Quantum circuits cannot copy qubits, but classical computers can copy bits. In a classical NN, a feature can be sent to multiple neurons to consider its correlation with others. However, a QNN can only use a qubit once. Using CNOTs, for example, a QNN can consider the correlation between a qubit and others and bypass this limitation in quantum circuits. The entanglement circuit is followed by a parametric circuit that consists of single-qubit gates and searches through the solution space. An effective method for building a parametric circuit is using rotation gates [69], [73]. The angle of rotation is determined through training of the QNN, similar to weights and biases in classical NNs. The combination of entanglement and parametric gates is referred to as a PQC. PQC is more convenient for NISQ devices than MAC-based QNNs since, in general, a PQC requires a short-depth circuit. Also, its variational core makes it more resilient to experimental errors. Figure 2.8 shows the PQC used in this work. Each layer can have multiple PQCs. There is no mathematical formula that specifies the optimal number of PQCs per layer. One way to decide on the structure of a layer is the try-and-error

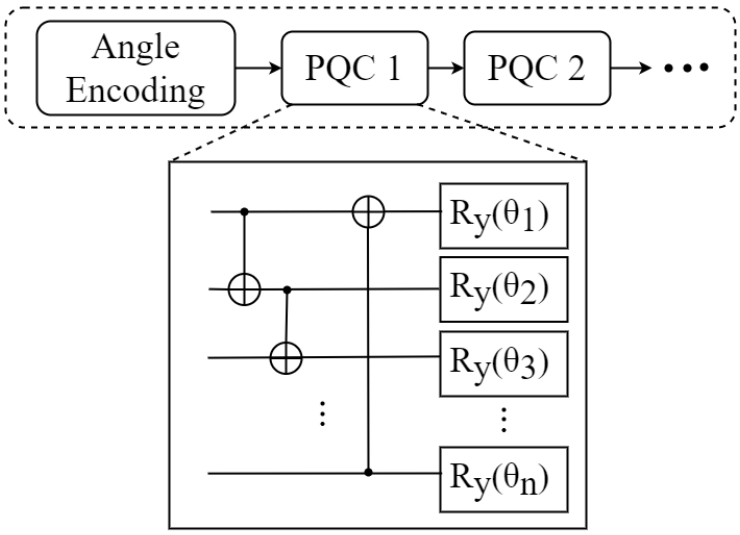


Figure 2.8: Architecture of a quantum layer. Each layer consists of one or more PQCs. Each PQC uses CNOTs to entangle the qubits. The CNOTs are followed by single-qubit Ry(θ) gates.

approach. Through simulation, the accuracy of classifiers with different numbers of PQCs is calculated, and a classifier with the highest accuracy is selected.

2.7.3 Measurment

In a Quantum Neural Network (QNN), measurement is the final stage that converts quantum information into classical data that can be used for learning or prediction. After quantum operations have been applied to encode and process information through parameterized quantum gates, measurement collapses the quantum state into a set of classical outcomes based on a chosen observable—typically the Pauli-Z operator for each qubit. The expected values of these measurements serve as the network's output features, which can then be compared with target values to compute a loss function and guide parameter

optimization via gradient-based training. Measurement plays a crucial role in bridging the quantum and classical components of a QNN, determining how quantum states are interpreted and how effectively the quantum circuit contributes to the learning process. The details about the measurement operation are explained in section 2.1.4.

2.7.4 Training of a QNN

We use the backpropagation algorithm for training a QNN. A cost function is needed for training. The cost function indicates the error of QNN. We used mean square error as the cost function:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \text{QNN}(X_i)\right)^2 \tag{2.9}$$

where N is the size of training set, X_i is the input, Y_i is the correct label associated with X_i, and $\text{QNN}(X_i)$ is prediction of the QNN for X_i. We use the Adam gradient descent algorithm for training. A gradient descent algorithm computes the derivative of the cost function with regard to network parameters. The parameter shift [74] is a popular method for computing quantum gradients. The gradient is computed by evaluating the cost function at two data points. The two data points do not need to be close to each other, which increases the resiliency of this technique to noise [74]. The backpropagation algorithm based on parameter shift gradient is implemented in PennyLane [75], and we use it to build and train QNNs used in this work.

2.8 Related Works on QNNs

Data re-uploading classifier [72] is a quantum circuit that can model an arbitrary classifier using a single qubit. Input data are uploaded sequentially into the qubit through angle

rotations. Each set of input data is followed by PQC layers. The combination of data uploading and PQCs repeats many times. The angle of rotation, as well as parameters of PQCs, should be optimized using a classical minimization algorithm such as the steepest descent algorithm. This classifier is not scalable as the depth of the circuits increases linearly with the number of features in the input dataset. As an example, a dataset with 100 features requires 100 sequential single-qubit operations.

Circuit-Centric Quantum Classifier (CCQC) [69] exploits amplitude encoding to reduce trainable parameters in PQCs. Once state preparation is completed, the prediction of the model is performed by applying a set of one- and two-qubit quantum gates. In contrast to some other QNN designs [76], [72], where the number of learnable parameters is proportional to the number of features, CCQC allows for exponentially fewer learnable parameters. To overcome overfitting, CCQC randomly selects a qubit and sets it aside for a certain number of training epochs. Then, the qubit is re-added to the circuit, and another qubit is randomly dropped. Similar to data-re-uploading [72], CCQC relies on measurement to add non-linearity flavor to QNNs.

QNet [77] breaks down a large QNN into several smaller QNNs where each QNN can be executed on a quantum computer. By carefully deciding on the size of each of those small QNNs, QNet can run machine learning algorithms of any size. Since those small QNNs that are independent can run in parallel, QNet enables heterogeneous technology integration within an application. QNet relies on classical computers for activation functions. The output of a quantum computer is fed to a classical computer to support non-linearity in QNNs.

Quantum computing promises computational advantages for some applications [78], [79]. In the absence of quantum error correction on contemporary quantum computers, optimization techniques play a vital role in closing the gap between the device and the

application. Thus, mitigation of hardware errors through the design of highly optimized quantum circuits is an active area of research.

A large body of prior work exists on quantum circuits optimization to reduce the total number of gates or number of layers [80], [81]. Maslov et al. [80] optimize a quantum circuit without considering hardware constraints. A quantum circuit is broken into subcircuits, and then each subcircuit is processed to be replaced by an equivalent circuit that has a lower cost. In the next step, a greedy algorithm is employed to compact quantum layers. A layer is defined as a set of gates that can operate in parallel. Layers in a circuit are formed one by one, starting from the primary inputs. Each time a new layer is created, it is analyzed to determine whether it can be merged with existing layers. The shortcoming of this technique is that it assumes that the error across qubits and gates is the same.

Qubit placement [81] is an NP-complete problem and requires an effective heuristic-based algorithm to map a quantum circuit to hardware. The algorithm proposed in [81] starts with a non-optimized basic mapping where a quantum circuit can operate on hardware. In the next step, a hill-climbing algorithm tries to match a qubit with the optimum physical gate. Each time, a new placement assignment is created by mapping a qubit to a new physical gate and is compared with the existing mapping. If it is less costly, then the new placement replaces the old one; otherwise, it moves on to the next qubit. Our optimization techniques are different as we focus on parametric circuits where the operation of the circuits can be adjusted by training parameters of the circuits. On the contrary, the above optimization techniques are designed primarily for circuits with fixed and non-trainable gates.

Murali et al. [82] mitigate the impact of crosstalk noises in quantum circuits through software techniques. Crosstalk occurs when multiple quantum gates operate simultaneously, corrupting the state of qubits. Crosstalk arises from fundamental challenges in

quantum hardware, such as leakage of control signals (needed for gate operations) onto qubits that are not part of the intended hardware operation. An approach to prevent interference between two high crosstalk components is to schedule them serially by using control instructions, such as a barrier. However, due to the fragile nature of qubits, serialization of quantum operations causes loss of quantum information. Murali et al. [82] develop a software module that serializes some of the crosstalk operations but also balances the need to avoid exponential decoherence error due to serialization. As an example, only those gate pairs that are separated by one hob are serialized as crosstalk talk noise, for gates with a distance of more than one hob is negligible. As a result, when two pairs are separated by two or more hobs, they can be executed in parallel to reduce the impact of serialization on coherence time.

Ding et al. [83] propose a frequency-aware software technique to reduce crosstalk noise via dynamic frequency tuning. In superconducting quantum computers, two qubits interact with each other through resonance of qubit frequencies. There are two main approaches to preventing accidental interference of resonance frequencies. The first one exploits tunable qubits where the frequencies of qubits are set to be apart so that the probability of frequency interference reduces. The second method uses tunable couplers and temporarily disables connections that cause interference.

IBM Q system uses fixed frequencies for all qubits and fixed couplers across all links and leaves it to a scheduler to avoid crosstalk noises [82]. On the contrary, Google uses tunable qubits with either fixed or tunable couplers [84]. Higher tunability offers more flexible hardware and provides more control over devices. However, it induces higher hardware overhead and causes sensitivity to control noise.

Ding et al. [83] introduce a balanced design, i.e., qubit frequencies are tunable, but couplers are fixed. This approach offers a high program success rate via software that

maps frequency decision to the coloring of the crosstalk graph. A vertex in the crosstalk graph represents a qubit, and an edge represents a link, i.e, a capacitor in the frequency-tunable architecture. When the qubits are idle, we would like to have different frequencies for every pair of connected qubits. In the context of graph theory, it is equivalent to coloring the connectivity graph where endpoints of an edge have different colors. If the graph can be colored with C colors, then the qubits require C different frequencies to avoid interference. Crosstalk mitigation techniques can be combined with our optimization techniques to reduce noise in QNNs further.

There have been numerous efforts to build tools for the implementation of quantum circuits on hardware so that the noise level is contained. As an example, Variation-aware Qubit Allocation (VQA) [85] takes into account the error rate of links to map logical to physical qubits. This is in contrast to some other mapping techniques, which are oblivious to the variation in the link reliability [14]. VQA starts with an initial mapping and then tries to converge to a configuration with minimum inter-qubit errors. To do so, it estimates the most frequently used qubits by analyzing the first n quantum gates and calculating the frequency of accesses to qubits. Then, it maps the most frequently used qubits to the physical qubits with the most reliable links to reduce noise in quantum circuits. The downside of VQA is that it assumes that qubits are equally important for outputs. However, in QNNs, the sensitivity of an output varies from one input qubit to another. We exploit this variability and propose a mapping scheme that uses more reliable hardware resources for qubits with higher sensitivities. As such, we were able to remove some of the hardware noises that could not be eliminated by VQA.

QuantumNAS [86] is an evolutionary-based technique to reduce noise in variational quantum circuits. QuantumNAS first constructs a SuperCircuit by grouping a sufficient number of quantum layers to cover a large fraction of a quantum circuit. Then, the Su-

perCircuit is broken into small sections called SubCircuits. QuantumNAS uses hardware noise information and relies on a genetic algorithm to find the most robust quantum circuits. A gene vector encodes a SubCircuit. Each element in the sub-gene represents the circuit width in a layer. The evolution engine searches for reliable circuits using a combination of mutation and crossover. A mutation randomly changes a subset of genes with a pre-determined probability. Crossover first picks two parents and then generates a new sample with genes that are sampled randomly from the parents. Thus, the new population is generated based on the parent population, mutation, and crossover. Then, the most reliable circuit is selected from the new population. QuantumNAS can be integrated with our optimization techniques to reduce noise in QNNs further.

Quantum on-chip (QOC) training [87] uses real quantum computers to accelerate training of PQCs. QOC uses parameter shift to obtain PQC gradients. Parameter shift calculates the gradient of a parameter by simply shifting the parameter and calculating the difference between the corresponding outputs. Due to hardware noises, gradients obtained through parameter shift have low fidelity and thus reduce the accuracy of training. QOC exploits pruning to mitigate the impact of noise on training. When the gradient magnitude is small, noises have a more detrimental impact on signals. The unreliable gradients have a harmful impact on training convergence as well as the accuracy of the trained network. Skipping those unreliable gradients can mitigate the impact of noise on the training of PQCs. Our approach differs as we focus on inference of QNNs.

2.9 Related Works on QRAM

QRAM is a memory element analogous to classical Random Access Memory (RAM) that can store data in a quantum format. Similar to RAM, QRAM has three components: the

input (or address) register, the output (or data) register, and the memory arrays. There are several approaches to implementing QRAM, with the earliest proposals being Fanout QRAM and Bucket-Brigade Quantum Random Access Memory (BB-QRAM)) [88], [89]. Fanout QRAM, inspired partly by classical addressing methods, is organized in a binary tree structure. It operates in two stages: address loading, where the address is entangled with the routers in the tree, and data retrieval, where a quantum bus traverses the marked path, allowing classically controlled gates to copy the data for output. The routers are then reset to a default state, typically $|0\rangle$ [90].

Giovannetti et al. [71] propose a bifurcation graph-based classical RAM structure into the quantum realm using qutrits. A qutrit, the ternary equivalent of a qubit, can represent three states: $|left\rangle$, $|right\rangle$, and $|wait\rangle$. Each leaf node in the QRAM acts as a memory cell for data storage. Initially, all qutrits are set to the —wait$\rangle$ state. Starting from the root, the address input states are sequentially sent to each qutrit, changing their states accordingly.

A quantum circuit-based implementation of bucket-brigade QRAM was proposed in [88]. This design requires $\mathcal{O}(n)$ qubits for addresses, $\mathcal{O}(2^n)$ ancillary qubits for quantum switches, $\mathcal{O}(2^n)$ memory cell qubits, and one qubit for readout. The process involves using address qubits to sequentially activate ancillary qubits, routing the path to the appropriate memory cell. This paper also addressed the robustness of bucket-brigade QRAM, revealing that when quantum error correction is applied to the circuit, the circuit loses the advantage of having a small number of active gates as the error correction operates on all of its components.

Giovannetti et al. [90] extended the original bucket-brigade QRAM concept [71] by introducing fan-out QRAM, a variant inspired by classical fan-out RAM. In fan-out RAM, each k^{th} address bit controls 2^k switches, and this concept is applied to the quantum

version, where each k^{th} address qubit controls 2^k quantum switches. In fan-out QRAM [90], the quantum switches are initially set to the $|0\rangle$ state, and the address qubits in the input register modify the switches' states: those connected to a $|0\rangle$ qubit remain in $|0\rangle$, while those connected to a $|1\rangle$ qubit change to $|1\rangle$, enabling the routing of data. The key difference between the two QRAM types is that bucket-brigade QRAM uses qutrits (three-level systems) for its switches, while fan-out QRAM uses qubits (two-level systems). Bucket-brigade QRAM is resilient to noise but suffers from limitations related to the use of qutrits and the complexity of the circuit [91].

Another design, Flip-Flop QRAM (FF-QRAM) [92], is a circuit-based architecture tailored for classical data. The overall circuit features exponential depth and linear width relative to the number of address qubits. Specifically, for n address lines and m bits of data per address, the circuit depth is $\mathcal{O}(2^n)$ and the width is $\mathcal{O}(n+m)$. Storing a single data point in Flip-Flop QRAM involves three stages: the flip stage, the register stage, and the flop stage. In the flip stage, a 'compute' operation matches the address and data qubits to the $|1\rangle$ state for storage. The register stage uses a multi-controlled rotation gate to store the data in a register qubit, and the flop stage, an 'uncompute' operation, reverses the compute operation on the address and data qubits [92]. Another approach employs a teleportation-based method, which has demonstrated improved success rates on photonic-integrated-circuit platforms compared to bucket-brigade QRAM [93], [94].

Recent innovations include using an Entangling Quantum Generative Adversarial Network (EQGAN) to approximate QRAM [95]. EQGAN is a GAN model built on quantum entanglement principles, employing PQC. It features a quantum generator and discriminator that are jointly trained using a minimax optimization approach. This design allows the QRAM to efficiently encode data using a constant $\mathcal{O}(1)$ number of gates.

PQCs are also gaining attention as a viable option for implementing QRAM in smaller-

scale quantum algorithms, including QML applications [96]. The trainable PQC-based QRAM proposed in [97], [98] enables data storage in the quantum Hilbert space by training PQC. This QRAM approach facilitates faster convergence for classification tasks when paired with QNNs and achieves error-free storage of 4-bit binary data.

All the aforementioned QRAM implementations are too complex to run on NISQ devices. Their excessive number of gates and circuit depth introduces significant noise on quantum computers, making them impractical for use on NISQ devices.

Chapter 3

Quantum Arithmetic Circuits

n this chapter, we begin by presenting the quantum circuit design for an exact adder, constructed using the 2-input AND gate [26] introduced in the previous chapter. We describe the logical structure of the adder, the required ancilla qubits, and the sequence of quantum operations necessary to perform accurate bit-wise addition. A detailed analysis of its depth, gate count, and qubit resources is also provided to establish a baseline for subsequent optimizations.

Next, we introduce an approximate-computing approach to reduce the implementation cost of the exact adder. By selectively relaxing precision in noncritical arithmetic components, we demonstrate how circuit depth, T-count, and the number of two-qubit gates can be significantly reduced while maintaining acceptable output accuracy. The design principles and error characteristics of the proposed approximate adder are discussed, along with a comparison to the exact version.

Building on this foundation, we then employ the approximate adder to construct efficient quantum circuits for multiplication, division, and square-root operations. For each

arithmetic unit, we describe the architectural modifications, the interaction between subcomponents, and the impact of approximation on performance and resource usage. Furthermore, we highlight how the reduced-complexity arithmetic units enable more scalable and hardware-friendly quantum designs, especially in the context of NISQ-era devices.

Overall, this chapter establishes a unified framework for designing cost-effective quantum arithmetic circuits, showing how approximate computing can be systematically leveraged to improve scalability without substantially compromising computational accuracy.

3.1 Quantum Exact Adder

Figure 3.1 shows the block diagram for a cost-efficient exact quantum-controlled adder [26]. The adder includes both compute and uncommuted parts and only relies on CNOT gates and 2-input AND gates to compute the sum and carry [26]. Consider the addition of two n-bit numbers a and b; Quantum registers $|A\rangle$ and $|B\rangle$ accommodate numbers a and b, respectively. The addition of $|A_i\rangle$ and $|B_i\rangle$ is conditioned on the value of one bit stored in a quantum register $|Ctrl\rangle$. At the end of computation, quantum register $|B\rangle$ holds the sum of $|A\rangle$ and $|B\rangle$. Quantum register $|A\rangle$ remains unchanged at the end of the computation to satisfy the reversibility of quantum circuits.

Each carry-bit C_{i+1} (where $0 \leq i \leq n-1$) is produced based on inputs at quantum register locations $|A_i\rangle$ and $|B_i\rangle$, as well as the carry bit C_i, generated in the previous stage (with C_0 initialized to zero). All carry bits are stored in ancillas. The equation for carry bits is as follows:

$$|C_{i+1}\rangle = \Big| (C_i \oplus A_i) \wedge (C_i \oplus B_i) \oplus C_i \Big\rangle \tag{3.1}$$

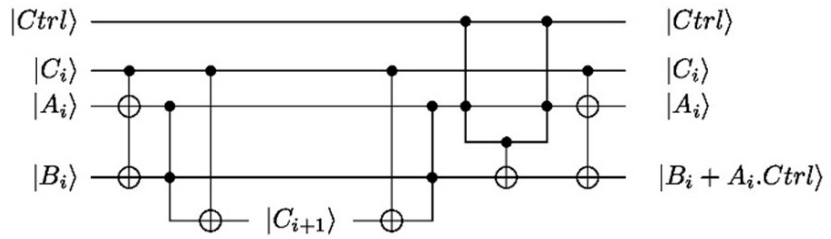


Figure 3.1: A controlled exact adder [26].

where $\oplus$ is the XOR operation applied using CNOT gates and $\wedge$ is the logical AND operation. When quantum register $|Ctrl\rangle = 1$, then qubit $|B_i\rangle$ is converted to Sum_i:

$$Sum_i = |C_i \oplus B_i\rangle \oplus |C_i \oplus A_i\rangle = |C_i \oplus A_i \oplus B_i\rangle \tag{3.2}$$

The circuit in Figure 3.1 is general and can be used to implement an adder of any size. However, as we will show, the adder with size of 4 ($n = 4$) is too noisy on a NISQ device. In particular, the carry-path increases with n, and this prevents the implementation on a real quantum computer from generating any meaningful result. This is primarily due to the ripple-carry structure of the adder, in which each carry bit must propagate sequentially through all preceding qubits. As n increases, the depth of the carry-path grows linearly, leading to a larger number of gates and longer circuit execution times. On real quantum hardware, this accumulation of operations exacerbates decoherence and gate errors, preventing the circuit from producing meaningful results.

3.2 Approximate Computing

Many applications, such as image processing and machine learning, can tolerate imprecision in computation and still generate acceptable and meaningful results. Accurate and precise circuits are not always needed for these applications. Thus, this characterization provides an opportunity to utilize inexact computing and simplify the circuits in terms of the number of gates to overcome noise in real quantum computers. Approximate computing is not a new topic. Over the past few years, it has been used to compromise accuracy for energy and/or performance in error-resilient applications. The paradigm of inexact computing relies on relaxing fully precise and deterministic circuit blocks to increase the energy efficiency of computing systems. This allows imprecise computing to redirect the design of exact digital systems by taking advantage of the error-tolerance property of some applications and reducing the complexity and cost of implementation. Since approximate computing introduces errors, metrics are required to measure the accuracy of approximate circuits. We evaluate the error in approximate circuits by Normalized Mean Error Distance (NMED). NMED is the Mean Error Distance (MED) normalized to the maximum value of the output. MED is defined as the average of Error Distance (ED), which is the absolute difference between the output of approximate and exact circuits across all combinations of inputs [99]. The definitions of ED, MED, and NMED are as follows:

$$ED = |out_{exact} - out_{approx}| \tag{3.3}$$

$$\text{MED} = \frac{1}{N}\sum_{i=1}^{N} ED \tag{3.4}$$

Table 3.1: Unified notation for approximate quantum arithmetic circuits.

Acronym	**Operation Type**	**Carry Mode**
C_AQANC	Controlled Approximate Quantum Adder	No Carry
C_AQAFC	Controlled Approximate Quantum Adder	Full Carry
C_AQAHC	Controlled Approximate Quantum Adder	Half Carry
C_AQAOC	Controlled Approximate Quantum Adder	One Carry
NC_AQANC	Non-Controlled Approximate Quantum Adder	No Carry
NC_AQAFC	Non-Controlled Approximate Quantum Adder	Full Carry
NC_AQAHC	Non-Controlled Approximate Quantum Adder	Half Carry
NC_AQAOC	Non-Controlled Approximate Quantum Adder	One Carry
AQMNC	Approximate Quantum Multiplier	No Carry
AQMFC	Approximate Quantum Multiplier	Full Carry
AQMHC	Approximate Quantum Multiplier	Half Carry
AQMOC	Approximate Quantum Multiplier	One Carry
AQDNC	Approximate Quantum Divider	No Carry
AQDFC	Approximate Quantum Divider	Full Carry
AQDHC	Approximate Quantum Divider	Half Carry
AQDOC	Approximate Quantum Divider	One Carry

$$NMED = \frac{MED}{MAX} \tag{3.5}$$

where out_{exact}, out_{approx}, N, and MAX are the accurate result, approximate result, number of all combinations of inputs, and the maximum value of the result, respectively.

3.3 Quantum Approximate Adder

The depth of a quantum circuit directly impacts the noise level on a real quantum computer. Since the carry generated for bit i of an adder depends on the carry generated in the previous stage, the quantum gates used for the generation of carry qubits are connected back-to-back. This causes the carry path to become the critical path in an adder. As a

result, it is not feasible to implement such an adder on a NISQ device. As we will show, the output of a 4-bit exact adder is too noisy on an IBM quantum computer. Therefore, in this section, we propose four approximate quantum adders to reduce the cost of implementation on NISQ devices. Each of these adders exploits a different level of approximation on the carry-path that affects the complexity of the quantum circuit. To enhance clarity in the sections that follow, Table 3.1 provides a comprehensive summary of all proposed approximate adders and their associated multipliers and dividers.

3.3.1 Controlled Approximate Quantum Adder with No-Carry

To eliminate the detrimental impact of the carry-path on noise level, we propose an Controlled Approximate Quantum Adder with No-Carry (C-AQANC) where all gates associated with the carry-path are eliminated. In other words, qubit i of the output solely depends on qubit i of the inputs. Figure 3.2 depicts the quantum circuit for a 4-qubit C-AQANC. For each output qubit, we use an AND gate to compute the bitwise-AND of control and a qubit from $|A\rangle$. The second AND gate is used for uncomputation, which is necessary to make the entire circuit reversible. At the end of the computation, register $|A\rangle$ remains unchanged. The result of the approximate adder is stored in register $|B\rangle$. If the control bit is zero, then the outputs of the AND gates are zero. As a result, the CNOT gates do not change register $|B\rangle$. On the contrary, if the control bit is one, then qubit i of register $|B\rangle$ is replaced by $|S_i\rangle = |A_i \oplus B_i\rangle$.

Each bit of C-AQANC requires an AND gate, which has a T-count of 4. We follow the circuit proposed by [26] to uncommute an AND gate. The T-count of an uncommuted AND gate is zero. Thus, the T-count of an n-qubit C-AQANC is $4n$. For the exact adder (section 3.1), the T-count is $8n$. As a result, C-AQANC cuts the cost of an exact adder

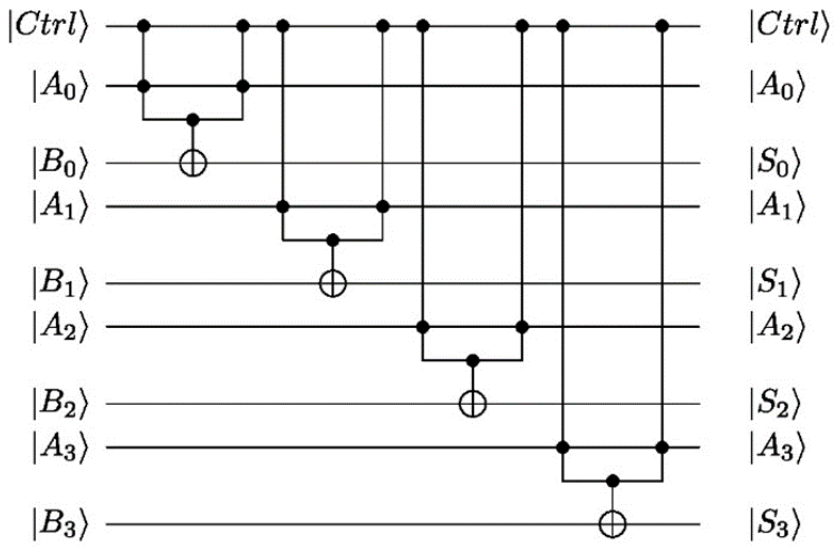


Figure 3.2: Quantum circuit for a 4-qubit C-AQANC.

by half. In addition to T-count, the other factor that impacts the noise level on a NISQ device is the depth of a circuit. The depth of a quantum circuit is the maximum number of stages, where each stage is composed of a quantum gate, or a series of quantum gates connected back-to-back. The depth of C-AQANC makes it scalable. Regardless of the size of an adder, the depth of C-AQANC remains the same. On the contrary, the depth of the exact adder is $O(n)$ as the delay of quantum gates used to compute carry qubits accumulates. As a result, an exact adder is not a viable solution for NISQ devices. Table 3.2 compares the error of C-AQANC with the error of exact and other approximate adders.

3.3.2 Controlled Approximate Quantum Adder with Full Carry

C-AQANC overcomes the cost of the carry-path by removing carry qubits from the adder. While this technique reduces the depth of the circuit, it causes significant error (Table

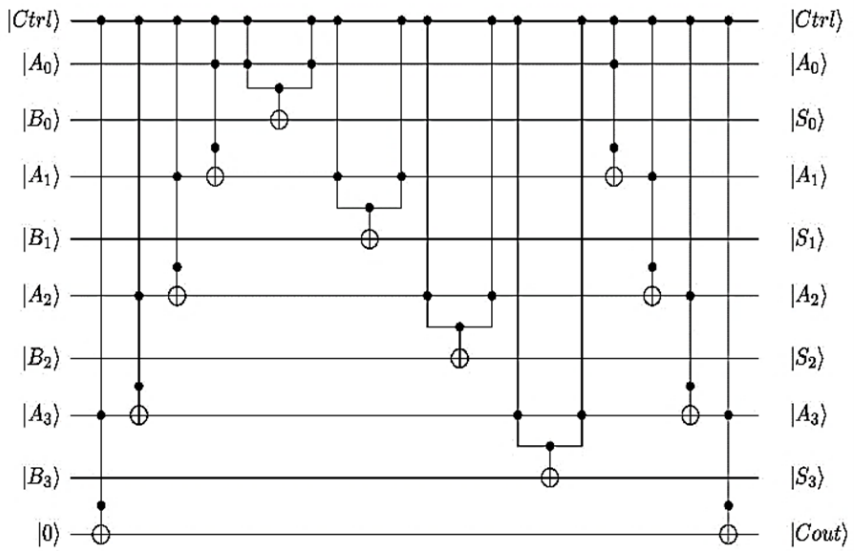


Figure 3.3: Quantum circuit for a 4-qubit C-AQAFC.

3.2). Between a complete carry-path in the exact adder and no carry-path in C-AQANC, we propose a Controlled Approximate Quantum Adder with Full Carry (C-AQAFC) that uses a simplified version of the carry-path. Certainly, we cannot allow a carry-path span from Least Significant Bit (LSB) all the way to the Most Significant Bit (MSB) as it has a detrimental impact on the depth of the circuit. C-AQAFC uses one of the two qubits from the previous stage to generate carry. Figure 3.3 shows the circuit block diagram for C-AQAFC. We use a bit from register $|A\rangle$ to generate carry. Since carry logic is confined between two consecutive qubits, the quantum gates needed for the generation of carry are localized. As a result, when the circuit in Figure 3.3 is synthesized into quantum gates of a real quantum computer, the noise level is small.

C-AQAFC requires two extra AND gates per qubit compared with C-AQANC. The extra AND gates are in charge of the computation/uncomputation of carry qubits. Thus, the T-count for an n-qubit C-AQAFC is $8n$ (C_0 is zero). It is important to note that while

T-count in C-AQAFC and the exact adder are the same, the depth of the two circuits is different. In the exact adder, the AND gate for the generation of C_i drives the quantum gates of the next stage. As a result, the depth of the carry path in an exact adder increases linearly with the size of the adder. However, in C-AQAFC, the AND gate used in $|C_i\rangle$ comes from the primary input of the circuit ($|A_{i-1}\rangle$). As a result, the depth of the carry-path is independent of the size of the adder, and this makes C-AQAFC scalable. Similar to C-AQANC, uncomputation logic for an AND gate does not require any T gate [26]. Fourth row in Table 3.2 shows NMED in C-AQAFC. Due to support of approximate carry, the error in C-AQAFC drops by 65% compared with C-AQANC.

3.3.3 Controlled Approximate Quantum Adder with Half Carry

Given that quantum computers are in the early stages, there are restrictions in terms of the size of a circuit that can be implemented on a real quantum computer. Although C-AQAFC simplifies the carry-path, if the size of the adder increases more than a certain threshold, C-AQAFC exceeds the capacity of quantum computers. Thus, we propose an Controlled Approximate Quantum Adder with Half Carry (C-AQAHC) which drops carry for the lower half of the adder and uses the same carry logic as C-AQAFC for the upper half. We choose the lower half to skip carry generation as the error of the adder is less sensitive to the lower half qubits than the upper half. Figure 3.4 shows a block diagram for C-AQAHC.

The T-Count in C-AQAHC drops to $6n + 4$ from $8n$ in C-AQAFC. The depth stays the same. Table 3.2 shows that errors in C-AQAHC and C-AQAFC are similar. Thus, C-AQAHC reduces the cost of the circuit while maintaining a similar level of accuracy.

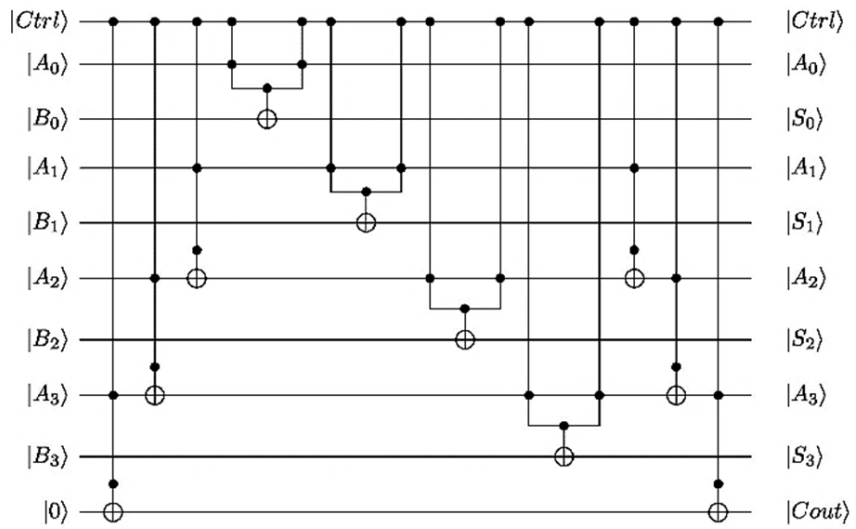


Figure 3.4: Quantum circuit for a 4-qubit C-AQAHC.

3.3.4 Controlled Approximate Quantum Adder with One Carry

The weight of qubits in an n-qubit register increases by a factor of two. As we move from LSB towards MSB, an error in a qubit has a higher impact on the final output of an approximate circuit. As an example, in a 4-qubit adder, flipping MSB can cause maximum ED of 8 while flipping the rest of the qubits can cause maximum ED of 7. Thus, we propose an Controlled Approximate Quantum Adder with One Carry (C-AQAOC) where carry is generated only for MSB. Note that in C-AQAFC, C-AQAHC, and C-AQAOC, in addition to sum, we generate an additional output: $|Cout\rangle$. This is needed to ensure that an n-qubit multiplier circuit (Section 3.4) generates a 2n-qubit product. For C-AQANC, $|Cout\rangle$ is always $|0\rangle$. Figure 3.5 depicts a circuit for C-AQAOC. The T-count in C-AQAOC is $4n+8$ and the depth is 3. Table 3.2 shows NMED for C-AQAOC. The error of C-AQAOC is close to C-AQAHC.

The results in Table 3.2 compare the numerical accuracy of the exact 8-qubit adder with

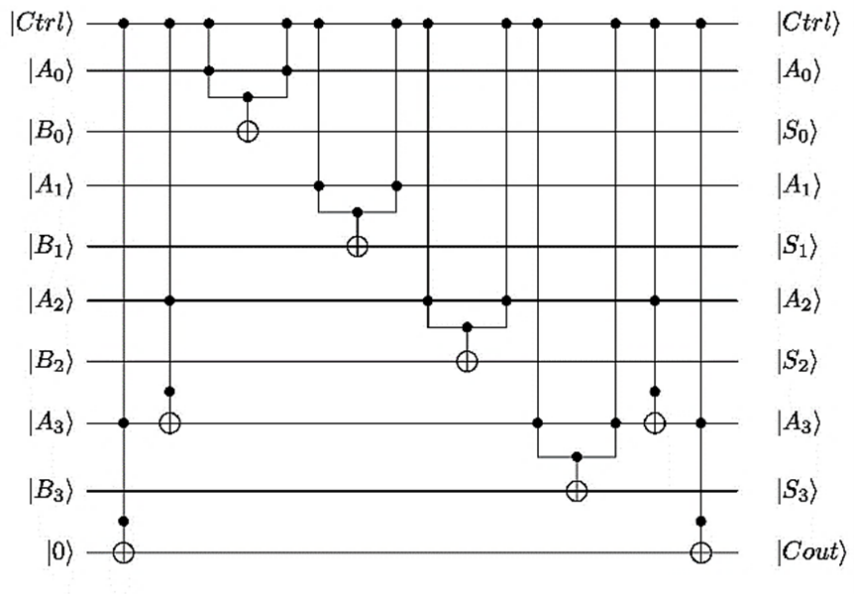


Figure 3.5: Quantum circuit for a 4-qubit C-AQAOC.

Table 3.2: NMED results for exact and approximate 8-qubit adder circuits.

Circuit	**NMED**
Exact	0
Proposed Approximate Adders	
C-AQANC	0.50000
C-AQAFC	0.17529
C-AQAHC	0.17683
C-AQAOC	0.18738

four proposed approximate quantum adder circuits using the Normalized NMED metric. As expected, the exact adder produces zero error, serving as a reference baseline. Among the approximate designs, C-AQANC exhibits the highest NMED (0.50000), indicating that this configuration introduces the largest deviation from the correct arithmetic results. This is consistent with its more aggressive gate and carry simplifications, which reduce circuit depth but at the cost of accuracy. On the other hand, C-AQAFC, C-AQAHC, and

C-AQAOC achieve significantly lower NMED values (approximately 0.17–0.19). These three designs strike a more balanced trade-off between circuit simplification and accuracy by retaining more of the carry-propagation structure. Among them, C-AQAFC shows the lowest error (0.17529), making it the most accurate approximate design. C-AQAHC follows closely with a comparable error level, while C-AQAOC exhibits slightly higher deviation but remains substantially more accurate than C-AQANC.

3.4 Quantum Integer Multiplication

3.4.1 Shift-and-Add Multiplication

A quantum multiplier computes the multiplication of two n-qubit registers $|A\rangle$ and $|B\rangle$ and generates a 2n-qubit product register $|P\rangle$. Due to the popularity of multiplication in digital hardware, researchers have developed many algorithms for multiplication [100], such as shift-and-add, Booth's algorithm, etc. In this work, we focus on the shift-and-add algorithm for multiplication of two numbers.

In each step of multiplication of two numbers, a bit from the multiplier is selected, starting from LSB. If it is one, then the multiplicand is added to the partial product. If it is zero, then the add operation is not needed. At the end of each step, the multiplicand is shifted to the left. Thus, the problem of multiplication is reduced to a series of conditional add operations. Algorithm 1 shows an algorithm for shift-and-add multiplication.

Algorithm 1 Shift-and-Add Multiplication ($|P\rangle = |A\rangle \times |B\rangle$).

Input: Multiplicand A, Multiplier B
Output: Product P
Initialize $P \leftarrow 0$
n: number of bits in B
for $i = 0$ to $n - 1$ **do**
  if $B_i = 1$ **then**
    $P \leftarrow P + A$
  end if
  Shift A one bit to the left ($A \leftarrow A \ll 1$)
end for

3.4.2 Proposed Approximate Quantum Multiplier

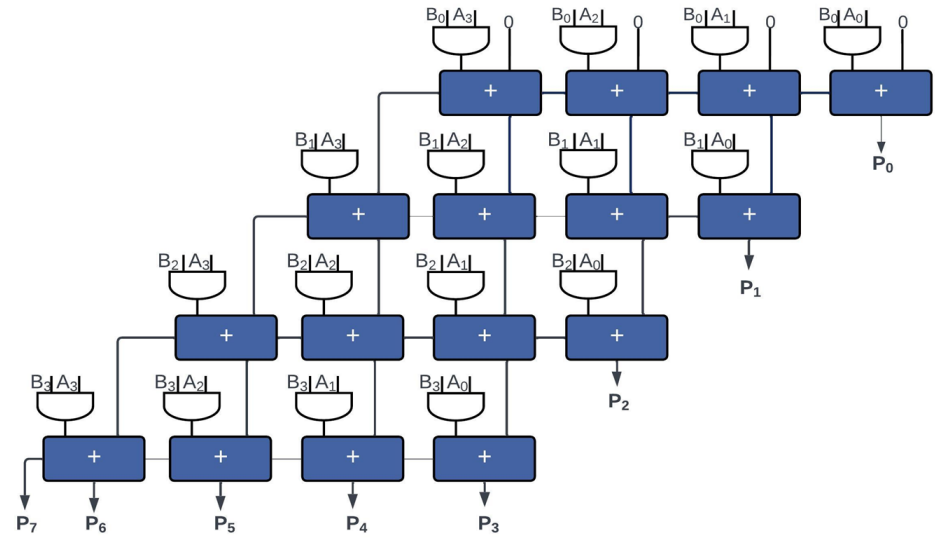
In a conventional ripple-carry adder, each qubit must wait for the carry from the previous stage before its sum can be computed, which forces the circuit to execute sequentially. This sequential dependency creates a long carry chain, increasing the circuit depth linearly with the number of qubits. By removing or approximating carries in the early stages of the adder, the sequential dependency is partially broken. The sum of each qubit in the early stages can then be computed independently, allowing gates to be applied in parallel rather than strictly sequentially. This parallelization dramatically reduces the overall circuit depth, decreases the number of gate operations, and consequently lowers the impact of decoherence and noise on NISQ devices, even though it introduces a controlled approximation in the final result. Moreover, this approximation of eliminating carry paths is particularly beneficial for designing multipliers, as it reduces the cumulative depth and complexity of the partial-product additions, leading to more noise-resilient quantum multiplier circuits.

Consider the multiplication of two n-bit numbers a and b stored in quantum registers $|A\rangle$ and $|B\rangle$, respectively. Further, consider a quantum register $|P\rangle$ of size $2 \times n$ consisting

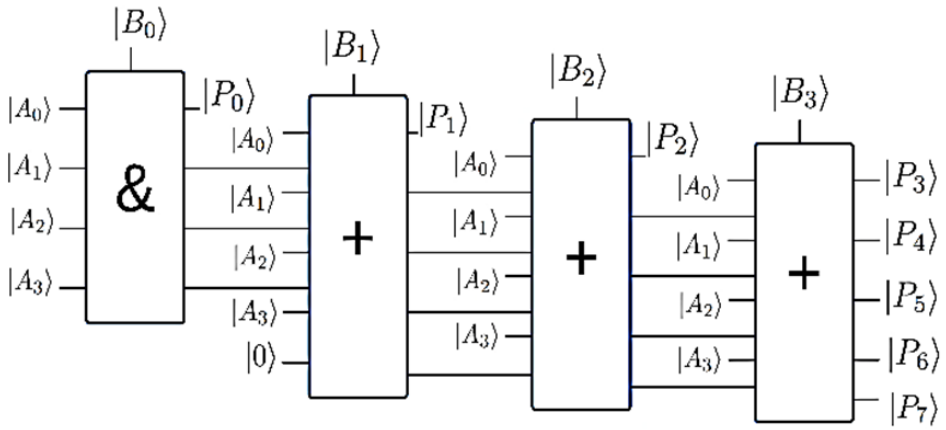
of ancilla qubits initialized to zero. At the end of the computation, the quantum registers $|A\rangle$ and $|B\rangle$ hold the values of a and b, respectively, while the result of the multiplication is stored in the quantum register $|P\rangle$. Figure 3.6 shows the approximate quantum multiplier circuit. We assume that the input registers of the multiplier in Figure 3.6 are 4-qubit registers. However, the proposed circuit is generic and can be applied to a multiplier of any size.

Each of the approximate adders described in Section 3.3 can be integrated into the approximate multiplier, with the choice of adder directly affecting both the accuracy and resource cost of the multiplier. The first stage of the multiplier circuit computes the bitwise AND between $|B_0\rangle$ and $|A\rangle$. This is implemented using four 2-input AND gates, where the first inputs are connected to $|A_0\rangle$ through $|A_3\rangle$ and the second inputs are all connected to $|B_0\rangle$. The outputs of these gates are stored in ancilla qubits $|P_0\rangle$ through $|P_3\rangle$. At the conclusion of this step, $|P_0\rangle$ of the quantum register $|P\rangle$ contains $|A_0 \cdot B_0\rangle$, representing the LSB of the final product.

For the second step, the multiplier computes the sum of the partial results generated in the previous step and the quantum register $|A\rangle$, conditioned on the value of $|B_1\rangle$. If $|B_1\rangle$ is zero, the adder in this step skips the addition; otherwise, it computes the sum of its inputs. All approximate adders proposed in Section 3.3 are controlled adders. The rationale for using controlled quantum adders is that our proposed quantum multiplier requires such adders in its structure. A $|Ctrl\rangle$ input determines whether the addition is performed or skipped. In the second step, the $|Ctrl\rangle$ input of the approximate adder is connected to $|B_1\rangle$. The two 4-qubit inputs of the adder are connected to $|A\rangle$ and $|0P_3P_2P_1\rangle$, respectively. At the end of this step, the states of the qubits are as follows:



(a)



(b)

Figure 3.6: Circuit for an approximate quantum multiplier based on controlled approximate adders.

$$\left(\bigotimes_{i=0}^{n-1}|A_i\rangle\right)\left(\bigotimes_{i=0}^{n-1}|B_i\rangle\right)|P_0\rangle\,|P_1\rangle\left(\bigotimes_{i=2}^{4}|P_i\rangle\right)\left(\bigotimes_{i=5}^{2n-1}|0\rangle\right) \tag{3.6}$$

The next $(n-2)$ steps are similar to the second step. In step j, $|B_{j-1}\rangle$ controls the

approximate adder. The first operand of the adder comes from $|A\rangle$, and the second operand comes from the partial results generated in step $(j-1)$. Note that for C-AQANC, the last output ($|Cout\rangle$) is always $|0\rangle$. In step j, the first j qubits of the final product are computed. After applying $(n-1)$ approximate adders to the partial results, the product of $|A\rangle$ and $|B\rangle$ is ready in the quantum register $|P\rangle$.

Table 3.3 shows the errors of the approximate multipliers. Approximate Quantum Multiplier with No-Carry (AQMNC), Approximate Quantum Multiplier with Full Carry (AQMFC), Approximate Quantum Multiplier with Half Carry (AQMHC), and Approximate Quantum Multiplier with One Carry (AQMOC) correspond to approximate multipliers based on C-AQANC, C-AQAFC, C-AQAHC, and C-AQAOC, respectively. AQMFC reduces the error of AQMNC by half. Eliminating the carry for half of the qubits in AQMHC increases the error by 6% compared with AQMFC. This gap rises to 25% in AQMOC, where the carry is used only in the MSB. Among the approximate quantum multipliers, using AQMFC significantly reduces error compared to AQMOC. Partial carry strategies (AQMHC and AQMNC) increase error, with minimal carry in AQMOC causing the largest error, highlighting the trade-off between carry complexity and accuracy.

The T-count of the proposed quantum multipliers depends on the type of adder used in each step. We assume that the same adder is used in all steps. The total T-count for an approximate quantum multiplier is calculated by summing the T-count of each step of the proposed circuit:

- **Step 1:** The first step consists of n AND gates. Thus, the T-count for step 1 is $4n$.
- **Steps 2–n:** From step 2 onward, each step uses a controlled approximate adder.

Assuming that the T-count of the adder is V, the T-count of the remaining steps is $(n-$

Table 3.3: Error of 8×8 Approximate Multipliers

Circuit	**NMED**
Exact	0
Proposed Approximate Multiplier	
AQMNC	0.162844
AQMFC	0.088077
AQMHC	0.093019
AQMOC	0.110338

1) $\times V$. Therefore, the total T-count of the proposed approximate quantum multiplier is $T_{\text{total}} = 4n + (n-1) \times V$.

3.5 Quantum Division

There are two main algorithms for the computation of the division of two numbers: restoring and non-restoring [100]. The main difference between the two algorithms is related to the remainder generated in each step. In the restoring algorithm, if the remainder is negative, its value is restored before the next step. On the contrary, the non-restoring algorithm continues to the next step without restoring the negative remainder to its positive value. In this section, we explain the two algorithms and propose quantum circuits based on the dynamic circuit to reuse qubits and reduce the cost of the circuits.

Algorithm 2 Algorithm for restoring quantum division

Require: $a = a_{n-1}a_{n-2}\ldots a_0$, $b = b_{n-1}\ldots b_0$
Ensure: Quotient Q and Remainder R of a/b

1: $Q \leftarrow 0_n$
2: $R \leftarrow a_{n-1}a_{n-2}\ldots a_0$
3: **for** $i = 0$ **to** $n-2$ **do**
4: $\quad Y \leftarrow Q_{n-2-i}Q_{n-3-i}\ldots Q_1Q_0R_{n-1}\ldots R_{n-1-i}$
5: $\quad Y \leftarrow Y - b$
6: $\quad$ **if** $Y < 0$ **then**
7: $\quad\quad Y \leftarrow Y + b$
8: $\quad\quad Q_{n-1-i} \leftarrow 0$
9: $\quad$ **else**
10: $\quad\quad Q_{n-1-i} \leftarrow 1$
11: $\quad$ **end if**
12: $\quad R_{n-1}\ldots R_{n-i-1} \leftarrow Y_i\ldots Y_0$
13: **end for**
14: $R \leftarrow R - b$
15: **if** $R < 0$ **then**
16: $\quad R \leftarrow R + b$
17: $\quad Q_0 \leftarrow 0$
18: **else**
19: $\quad Q_0 \leftarrow 1$
20: **end if**
21: **return** Q, R

3.5.1 Restoring Division

Algorithm 2 shows an algorithm for restoring division [101]. The algorithm computes the division of two operands, a and b, where each operand is an n-bit positive number represented in 2's complement format. At the end of the algorithm, Q holds the quotient and R holds the remainder. At the start of the algorithm, Q and R are initialized with zeros and a, respectively (lines 1–2). Then, in each iteration of the **for** loop, b is subtracted from Y, where Y is built through concatenation of Q and R (lines 3–13). In iteration i,

Table 3.4: An example for the restoring division algorithm

Step	Operation	Intermediate Values	Notes
Initialization	—	$R = a = 01111_2$, $b = 00011_2$, $Q = 00000_2$	—
$i = 0$	Subtraction	$Y = Q_3Q_2Q_1Q_0R_4 = 00000$	$Q_4 = 0$
		$Y - b = 00000 - 00011 = 11101$ $(Y < 0)$	
	Addition	$Y + b = 11101 + 00011 = 00000$	
		$Q_3Q_2Q_1Q_0R_4 = Y = 00000$	
$i = 1$	Subtraction	$Y = Q_2Q_1Q_0R_4R_3 = 00001$	$Q_3 = 0$
		$Y - b = 00001 - 00011 = 11110$ $(Y < 0)$	
	Addition	$Y + b = 11110 + 00011 = 00001$	
		$Q_2Q_1Q_0R_4R_3 = Y = 00001$	
$i = 2$	Subtraction	$Y = Q_1Q_0R_4R_3R_2 = 00011$	$Q_2 = 1$
		$Y - b = 00011 - 00011 = 00000$ $(Y >= 0)$	
		$Q_1Q_0R_4R_3R_2 = Y = 00000$	
$i = 3$	Subtraction	$Y = Q_0R_4R_3R_2R_1 = 00001$	$Q_1 = 0$
		$Y - b = 00001 - 00011 = 11110$ $(Y < 0)$	
	Addition	$Y + b = 11110 + 00011 = 00001$	
		$Q_0R_4R_3R_2R_1 = 00001$	
After For Loop	Subtraction	$R = R_4R_3R_2R_1R_0 = 00011$	$Q_0 = 1$
		$R - b = 00011 - 00011 = 00000$	
Final		$R = 00000_2$, $Q = 00101_2$	

the lowest $(n-1-i)$ bits of Q and the highest $(i+1)$ bits of R form Y. If the result of the subtraction is negative, then b is added to Y to offset the subtraction operation (lines 6–8), and the quotient bit is set to zero. However, if the subtraction leads to a positive number, Y remains unchanged, and the quotient bit is set to one.

Once the For loop terminates, b is subtracted from R. If the result is negative, then b is added to R and the MSB of the quotient is set to zero (lines 15-17); otherwise, the LSB of the quotient is set to one (line 19). Table 3.4 shows an example of Algorithm 2. The order of qubits appears from the MSB to the LSB.

Figure 3.7(a) shows a quantum circuit for the restoring algorithm. In this work, we

use Clifford+T gates to realize quantum division as these gates are universal and can be made fault-tolerant [1], [1]. We assume that n is equal to 5 in figure 3.7(a). However, the proposed circuit is general and can be used for restoring the division operation of any size.

Figure 3.7(b) shows the internal structure of iteration zero. Each iteration of the algorithm is implemented using a Quantum Subtractor (QSUB) and a Quantum-Controlled Adder (QCA). This makes the quantum division circuit modular, which simplifies coding as well as the realization of the circuit on hardware. If the control input of the QCA is in the state of $|1\rangle$, then the adder computes the sum of its inputs; otherwise, it does not change the inputs.

The circuit is composed of three main registers: $|B\rangle$, $|Q\rangle$, and $|R\rangle$. Registers $|B\rangle$ and $|Q\rangle$ are n-qubit each, while register $|R\rangle$ is a single qubit. Thus, the total number of qubits is $2n + 1$. Register $|B\rangle$ is initialized with the operand b, each qubit in register $|Q\rangle$ is initialized to $|0\rangle$, and $|R\rangle$ is initialized with a_{n-1}.

The quantum division circuit uses the dynamic circuit $(n-1)$ times. Each time a qubit is measured through the dynamic circuit, a qubit of the quotient is generated, starting from the Most Significant Qubit (MSQ). At the end of the algorithm, $|Q_0\rangle$ holds the Least Significant Qubit (LSQ) of the quotient. The circuit generates the remainder through the register combination $|R\rangle\,|Q_{n-1}\rangle\,|Q_{n-2}\rangle \ldots |Q_1\rangle$.

It is important to note that R in Algorithm 2 is implemented by the single qubit $|R\rangle$ and $(n-1)$ reused qubits, as illustrated in Figure 3.7(a). To simplify the explanation of the circuit, we show the mapping between the algorithm and the circuit in three sections:

a) Iteration Zero

Iteration zero in figure 3.7(a) can be decomposed into six steps:

- **Step 1:** Treat qubits $|Q_{n-2}\rangle$ to $|Q_0\rangle$ from quantum register $|Q\rangle$ and qubit $|R\rangle$ as an n-qubit register $|Y\rangle$ (line 4), such that qubits $|Q_{n-2}\rangle$ to $|Q_0\rangle$ are mapped to $|Y_{n-1}\rangle$ to $|Y_1\rangle$, and qubit $|R\rangle$ is mapped to $|Y_0\rangle$.

- **Step 2:** Apply quantum registers $|Y\rangle$ and $|B\rangle$ to an n-qubit QSUB so that quantum register $|B\rangle$ is unchanged and $|Y\rangle$ holds $|Y - B\rangle$ at the end of the computation (line 5).

- **Step 3:** Apply a CNOT gate to locations $|Q_{n-1}\rangle$ and $|Q_{n-2}\rangle$ so that qubit $|Q_{n-2}\rangle$ does not change, and the state of $|Q_{n-1}\rangle$ changes to $|Q_{n-1} \oplus Q_{n-2}\rangle$. This step sets $|Q_{n-1}\rangle$ to $|1\rangle$ if the result of the subtraction is negative; otherwise, $|Q_{n-1}\rangle$ stays at $|0\rangle$.

- **Step 4:** Apply quantum registers $|Y\rangle$ and $|B\rangle$ to an n-qubit QCA so that the state of $|B\rangle$ is unchanged and $|Y\rangle$ holds the output of the QCA. The control terminal of the QCA is derived from $|Q_{n-1}\rangle$. If $|Q_{n-1}\rangle$ is in the state $|1\rangle$ (i.e., the output of the QSUB is negative), then the QCA computes $|Y + B\rangle$ (line 7); otherwise, $|Y\rangle$ remains the same.

- **Step 5:** Apply a quantum NOT gate to location $|Q_{n-1}\rangle$ to complement the state of the qubit. If the output of the QSUB is negative, this step sets $|Q_{n-1}\rangle = |0\rangle$ (line 8); otherwise, it sets $|Q_{n-1}\rangle = |1\rangle$ (line 10).

- **Step 6:** Through the dynamic circuit, $|Q_{n-1}\rangle$ is measured. This generates the MSQ of the quotient. Then, $|Q_{n-1}\rangle$ is reset to be used for the next iteration. The quantum NOT gate at the input of Iteration 1 in figure 3.7(a) initializes the reused qubit with a_{n-2}.

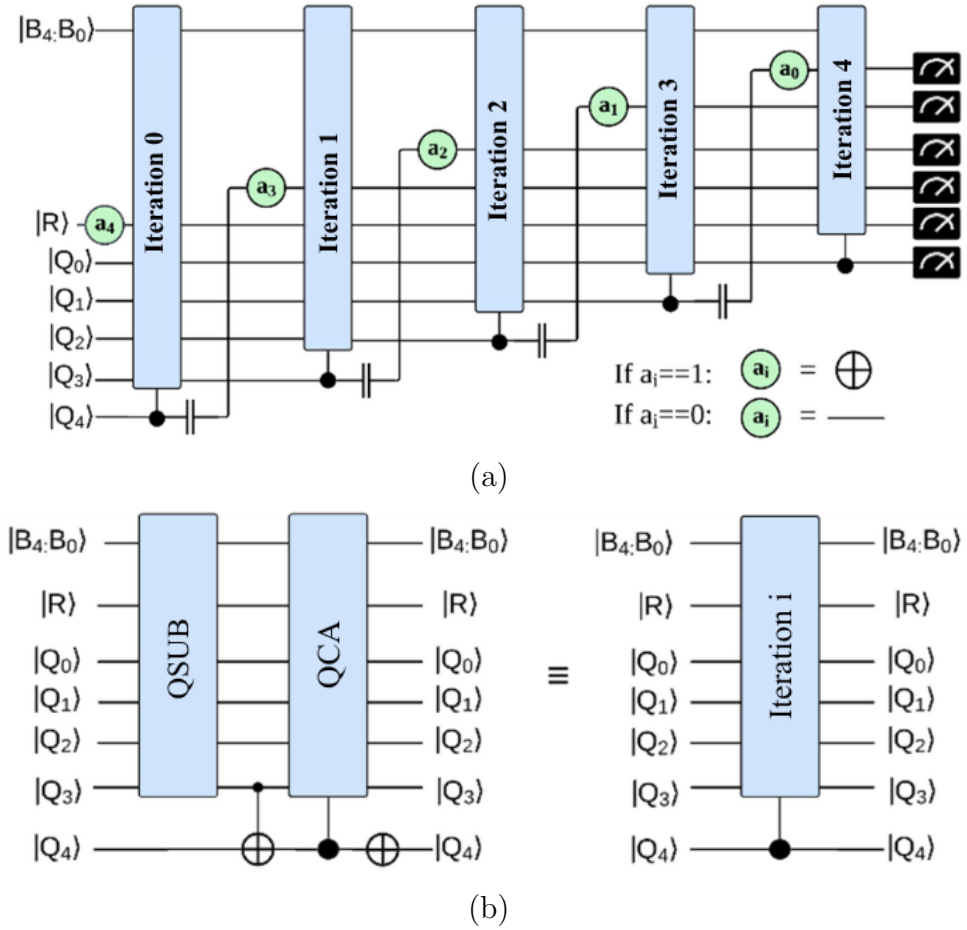


Figure 3.7: (a) Quantum circuit for the restoring algorithm [101] using dynamic circuit. (b) Architecture of iteration zero.

b) Iteration $i = 1$ to $n - 2$:

Iteration i in figure 3.7(a) subtracts $|B\rangle$ from $|Q_{n-2-i}\rangle \ldots |Q_0\rangle\, |R\rangle\, |Q_{n-1}\rangle \ldots |Q_{n-i}\rangle$ (lines 4-5). Then, a CNOT gate is applied to locations $|Q_{n-1-i}\rangle$ and the MSQ of the QSUB, so that $|Q_{n-1-i}\rangle$ is the target qubit and the MSQ of the QSUB is the control qubit. If the result of the subtraction is negative (line 6), then $|Q_{n-1-i}\rangle$ is set to $|1\rangle$; otherwise, it stays in the state $|0\rangle$.

$|Q_{n-1-i}\rangle$ is used to control the QCA in Iteration i. If $|Q_{n-1-i}\rangle = |1\rangle$ (which indicates the output of the QSUB is negative), the QCA adds $|B\rangle$ to the output of the QSUB to cancel the subtraction (line 7). Otherwise, if $|Q_{n-1-i}\rangle = |0\rangle$, the QCA does not modify the output of the QSUB.

A NOT gate is applied to $|Q_{n-1-i}\rangle$ to generate a qubit of the quotient. If $|Q_{n-1-i}\rangle = |1\rangle$, the NOT gate flips it to $|0\rangle$ (line 8); otherwise, it is set to $|1\rangle$ (line 10).

$|Q_{n-1-i}\rangle$ holds qubit $(n-1-i)$ of the quotient. The dynamic circuit connected to $|Q_{n-1-i}\rangle$ first measures the corresponding quotient qubit, then resets $|Q_{n-1-i}\rangle$. In preparation for the next iteration, a_{n-2-i} is loaded into $|Q_{n-1-i}\rangle$ so that it can be used for the next block. If $a_{n-2-i} = 1$, a NOT gate is needed; otherwise, no quantum gate is required as $|Q_{n-1-i}\rangle$ is already reset to $|0\rangle$.

c) After the For Loop

The last block in figure 3.7(a) (Iteration 4) implements the rest of the algorithm (lines 14–20). First, register $|B\rangle$ and qubits $|R\rangle\,|Q_{n-1}\rangle \ldots |Q_1\rangle$ are applied to an n-qubit QSUB so that register $|B\rangle$ remains unchanged and qubits $|R\rangle\,|Q_{n-1}\rangle \ldots |Q_1\rangle$ hold the outcome of the computation (line 14).

Next, $|R\rangle$ and $|Q_0\rangle$ are applied to the control and target terminals of a CNOT gate, respectively. Then, $|Q_0\rangle$ is used as the control qubit of a QCA. Register $|B\rangle$ and qubits $|R\rangle\,|Q_{n-1}\rangle \ldots |Q_1\rangle$ are used as input operands of the QCA. If the output of the QSUB is negative, $|Q_0\rangle$ is in the state $|1\rangle$, and the QCA performs addition to nullify the subtraction (line 16).

At the end, a quantum NOT gate flips $|Q_0\rangle$ to generate the LSQ of the quotient. Measurements of the outputs of the QCA ($|R\rangle\,|Q_{n-1}\rangle \ldots |Q_1\rangle$) generate the remainder of

the division.

3.5.2 Non-Restoring Division

Algorithm 3 shows an algorithm for non-restoring division [102]. The algorithm receives two positive numbers, a and b, each of size n bits and represented in 2's complement format. At the end of the algorithm, the quotient and remainder are returned through Q and R, respectively. The size of R is $(n-1)$ bits, since the remainder from the division of two n-bit numbers can have at most $(n-1)$ bits.

Algorithm 3 Non-Restoring Division

Require: Dividend $a = a_{n-1} \dots a_0$, Divisor $b = b_{n-1} \dots b_0$
Ensure: Quotient Q and Remainder R of a/b

1: $R = a_{n-2} \dots a_1 a_0$
2: $Q = 0^{n-1} a_{n-1}$
3: $Y = Q = Q - b$
4: **for** $i = 1$ **to** $n-1$ **do**
5: $\quad \{Q_{n-i}, Y\} = Y_{n-1} \dots Y_0 R_{n-i-1}$
6: $\quad$ **if** $Q_{n-i} = 0$ **then**
7: $\quad\quad Q_{n-i} = 1$
8: $\quad\quad Y = Y - b$
9: $\quad$ **else**
10: $\quad\quad Q_{n-i} = 0$
11: $\quad\quad Y = Y + b$
12: $\quad$ **end if**
13: **end for**
14: $\{Q_0, R\} = Y$
15: **if** $Q_0 = 1$ **then**
16: $\quad R = R + b$
17: **end if**
18: $Q_0 = \sim Q_0$
19: **return** Q, R

In Algorithm 3, R is loaded with the lowest $(n-1)$ bits of a. The LSB of Q is loaded with the MSB of a, and the remaining bits of Q are initialized with zeros. To reduce the number of iterations in the `For` loop, b is subtracted from Q before the loop (line 3).

Then, in each iteration of the loop, Y is built from the previous value of Y and R (line 5). If $Q_{n-i} = 0$, the previous subtraction generated a positive value; thus, the quotient bit is set to 1, and b is subtracted from Y (lines 7-8). Otherwise, the quotient bit is set to 0, and b is added to Y (lines 10-11).

After the `For` loop terminates, b is added to R if the result of the previous subtraction is negative (line 16). Finally, the LSB of Q is flipped (line 18). Table 3.5 shows an example for Algorithm 3.

To make the paper self-explanatory, we prove the correctness of Algorithm 3. To distinguish values of Y across iterations of the loop, we use Y_i to represent the new value of Y set in iteration i (line 5). Y_i is an n-bit number where the LSB is equal to a_{n-i-1}.

The proof for the case when the subtraction in line 8 generates a positive value is trivial. Therefore, we focus on proving the correctness of the algorithm when a subtraction generates a negative value. Assume that in iteration i, $Q_{n-i} = 0$ (i.e., the subtraction in iteration $(i-1)$ generated a positive result). Then, b is subtracted from Y_i (line 8).

For iteration $(i+1)$, assume that $Q_{n-i-1} = 1$, which indicates that the subtraction in iteration i generated a negative value. We expect that the subtraction in iteration i is canceled, and then b is subtracted from $\{Y_i, a_{n-i-2}\}$.

Since $Q_{n-i-1} = 1$, b is added to Y_{i+1} (line 11):

$$Y_{i+1} + b = \{(Y_i - b) \ll 1, a_{n-i-2}\} + b = \{Y_i, a_{n-i-2}\} - 2 \times b + b = \{Y_i, a_{n-i-2}\} - b$$

Table 3.5: Step-by-Step Non-Restoring Division Example

Step	**Operations**	**Updated R, Q**
Initialization	$a = (01111)_2$, $b = (00011)_2$	$R = (1111)_2$, $Q = (00000)_2$
Subtraction	$Q = Q - b = 00000 - 00011 = 11101$	$R = (1111)_2$, $Q = (11101)_2$
$i = 1$	$Y = Q_3Q_2Q_1Q_0R_3 = 11011$ $Q_4 = 1 : Q_4 = 0$ $Y = Y + b = 11011 + 00011 = 11110$ $Q_3Q_2Q_1Q_0R_3 = Y = 11110$	$R = (0111)_2$, $Q = (01111)_2$
$i = 2$	$Y = Q_2Q_1Q_0R_3R_2 = 11101$ $Q_3 = 1 : Q_3 = 0$ $Y = Y + b = 11101 + 00011 = 00000$ $Q_2Q_1Q_0R_3R_2 = Y = 00000$	$R = (0011)_2$, $Q = (00000)_2$
$i = 3$	$Y = Q_1Q_0R_3R_2R_1 = 00001$ $Q_2 = 0 : Q_2 = 1$ $Y = Y - b = 00001 - 00011 = 11110$ $Q_1Q_0R_3R_2R_1 = Y = 11110$	$R = (1101)_2$, $Q = (00111)_2$
$i = 4$	$Y = Q_0R_3R_2R_1R_0 = 11101$ $Q_1 = 1 : Q_1 = 0$ $Y = Y + b = 11101 + 00011 = 00000$ $Q_0R_3R_2R_1R_0 = Y = 00000$	$R = (0000)_2$, $Q = (00100)_2$
After For Loop	$R = 0000 \Rightarrow Q_0 = \sim Q_0 = \sim 0 = 1$	$R = (0000)_2$, $Q = (00101)_2$

This proves the correctness of the algorithm.

Figure 3.8 shows a quantum circuit for the non-restoring division algorithm. The circuit uses registers $|Q\rangle$, $|B\rangle$, and $|R\rangle$ with n-, n-, and 1-qubit, respectively. The LSQ of $|Q\rangle$ is loaded with a_{n-1}, and the remaining qubits are initialized to $|0\rangle$. Quantum register $|B\rangle$ is loaded with b. Qubit $|R\rangle$ is initialized with a_{n-2}.

The register R in Algorithm 3 is implemented by the single qubit $|R\rangle$ and $(n-2)$ reused qubits in Figure 3.8. The building blocks of the circuit are an n-qubit QSUB, an n-qubit Quantum-Controlled Adder/Subtractor (QCAS), and an n-qubit QCA. If the control qubit of the QCAS is in the state $|1\rangle$, the module subtracts its inputs; otherwise, it computes

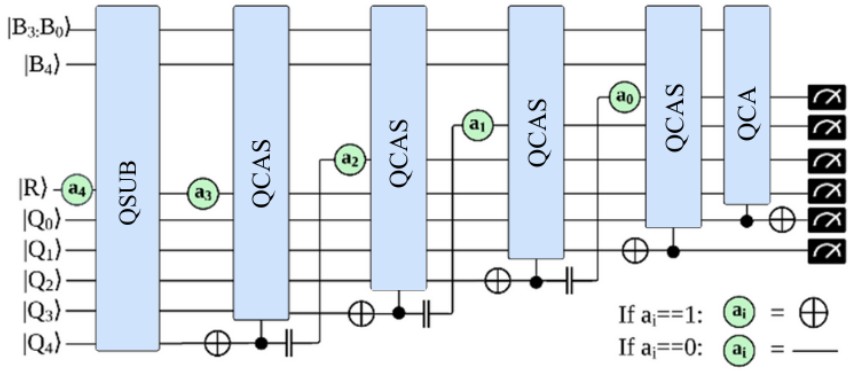


Figure 3.8: Quantum circuit for the non-restoring algorithm using a dynamic circuit.

the sum of its inputs.

While the circuit in Figure 3.8 has 5 qubits, the architecture is general and can be used to design a quantum division circuit of any size. We explain the operation of the circuit in three sections:

a) Before the For Loop

An n-qubit QSUB is used to implement line 3 of Algorithm 3. The quantum registers $|B\rangle$ and $|Q\rangle$ are applied as inputs to the QSUB such that, at the end of the computation, the register $|B\rangle$ remains unchanged, while the register $|Q\rangle$ contains the result of the subtraction.

b) Iteration 1 (n-1)

The main operation in the **for** loop is the conditional add/subtract operation. The output of the previous step determines whether the operands in the current iteration should be added or subtracted. The circuit uses a subset of the quantum register $|Q\rangle$ to form the operand $|Y\rangle$ in Algorithm 3 (line 5). In iteration i, the quantum register $|Y\rangle$ is formed by

concatenating

$$\{|Q_{n-i-1}\rangle \ldots |Q_0\rangle \ |R\rangle \ |Q_{n-1}\rangle \ldots |Q_{n-i+1}\rangle\}.$$

Then, a quantum NOT gate is applied to the MSQ of the previous step, $|Q_{n-i}\rangle$. Thus, if the result of the previous step is positive (line 6), the NOT gate changes the state of $|Q_{n-i}\rangle$ to $|1\rangle$; otherwise, it sets the state to $|0\rangle$.

If $|Q_{n-i}\rangle$ is in the state $|1\rangle$, the QCAS operation computes the subtraction of its inputs (line 8); otherwise, it computes the sum (line 11). The output of the NOT gate holds the quotient corresponding to iteration i (lines 7 and 10).

In the dynamic circuit at iteration i, the control qubit of the QCAS, $|Q_{n-i}\rangle$, is measured and then reset, allowing it to be used as the LSQ of $|Y\rangle$ in the next iteration.

c) After the For Loop

A QCA with $(n-1)$ qubits implements lines 15–17 of Algorithm 3 . The two inputs of the QCA are $|B\rangle$ and

$$\{|R\rangle \ |Q_{n-1}\rangle \ldots |Q_2\rangle\}.$$

The MSB of the QCAS in the last iteration of the loop, $|Q_0\rangle$, is connected to the control terminal of the QCA. If $|Q_0\rangle$ is in the state $|1\rangle$ (indicating a negative value), the QCA computes the sum of the input operands (line 16); otherwise, it leaves its inputs unchanged.

The control qubit $|Q_0\rangle$ is then flipped by a NOT gate to generate the LSB of the quotient. At the end of the circuit, the outputs of the QCA are measured to obtain the remainder of the division operation.

3.5.3 Approximate Quantum Division with Dynamic Circuit

The restoring and non-restoring quantum division circuits are not practical for implementation on a NISQ device, as such implementations result in excessive noise. This noise arises from both the quantum circuit depth and the number of quantum gates. The basic building block of the proposed circuits is an adder. Even a subtracter can be implemented using an adder, since

$$A - B = \overline{\overline{A} + B}.$$

Therefore, to simplify the circuit architecture, the focus should be on optimizing the adder, aiming to reduce both its depth and the number of gates.

An effective approach to simplify an adder is *approximate computing*. An n-qubit adder consists of n 1-qubit Quantum Full-Adders (QFA), where each QFA receives three qubits, $|a_i\rangle$, $|b_i\rangle$, and $|\text{carry-in}_i\rangle$, and produces two outputs, $|\text{sum}_i\rangle$ and $|\text{carry-out}_i\rangle$. The $|\text{carry-out}\rangle$ generated by a QFA serves as the $|\text{carry-in}\rangle$ of the next QFA.

Thus, the carry path in an n-qubit adder forms the critical path, which complicates the implementation of any circuit built upon the adder in a quantum computer. There are four approaches to break down the critical path in an n-qubit adder using approximate computing. Since we exploit these approximate adders for both restoring and non-restoring division circuits, we briefly explain the approximate adders in this section (Figure 3.9). These proposed adders are different from the approximate controlled adder explained in section 3.3 since their operation is not controlled by an additional qubit as a controller.

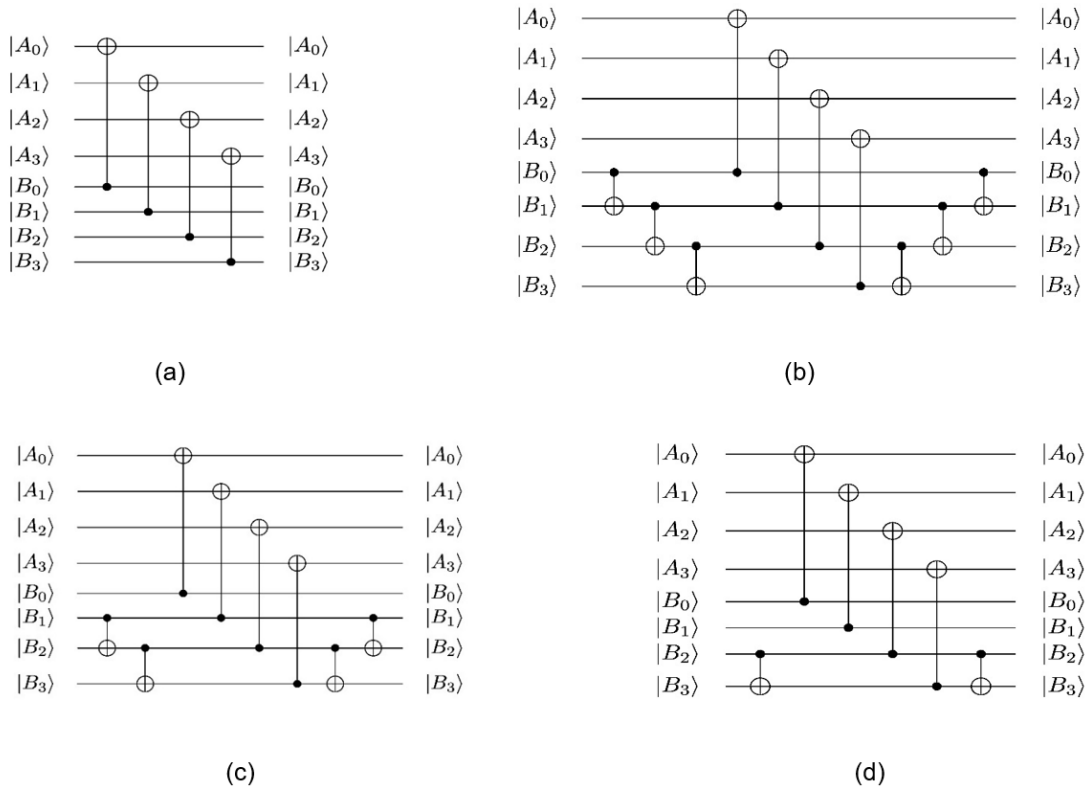


Figure 3.9: 4-qubit approximate quantum adders: (a) NC-AQANC, (b) NC-AQAFC, (c) NC-AQAHC, and (d) NC-AQAOC.

a) Non-Controlled Approximate Quantum Adder With No-Carry

An Non-Controlled Approximate Quantum Adder with No-Carry (NC-AQANC) removes the entire carry-path in an n-qubit adder (Figure 3.9(a)). In other words, each 1-qubit adder receives two inputs $|a_i\rangle$ and $|b_i\rangle$, and produces one output $|\text{sum}_i\rangle$. While such a simple structure is ideal for implementation of the circuit on a NISQ device, it causes significant error as all carry-out signals are removed.

b) Non-Controlled Approximate Quantum Adder With Full Carry

In contrast to the NC-AQANC, an Non-Controlled Approximate Quantum Adder with Full Carry (NC-AQAFC) utilizes carry signals for all QFAs. The |carry-out⟩ of a QFA in the NC-AQAFC is different from that in an exact adder. In an exact QFA, the |carry-out⟩ depends on all input qubits. However, such a structure is not practical on a NISQ device, as the depths of the carry-path across QFAs are accumulated. The NC-AQAFC uses only one of the input qubits from the previous step as |carry-in⟩. Figure 3.9(b) shows the structure of a 4-qubit NC-AQAFC, where QFA_i uses $|B_{i-1}\rangle$ as $|\text{carry-in}_i\rangle$.

c) Non-Controlled Approximate Quantum Adder With Half Carry

An Non-Controlled Approximate Quantum Adder with Half Carry (NC-AQAHC) (Figure 3.9(c)) drops the carry signal for the lower half of the QFAs, as the impact of carry signals in the lower half is less significant than in the higher half with respect to error.

d) Non-Controlled Approximate Quantum Adder With One Carry

An Non-Controlled Approximate Quantum Adder with One Carry (NC-AQAOC) (Figure 3.9(d)) exploits the carry signal only for the MSQ. The NC-AQAOC is based on the observation that the weight of the MSQ in an n-qubit number is greater than the sum of the weights of all other qubits. As a result, the NC-AQAOC achieves an accuracy comparable to the NC-AQAHC, while significantly reducing the cost associated with the carry-path.

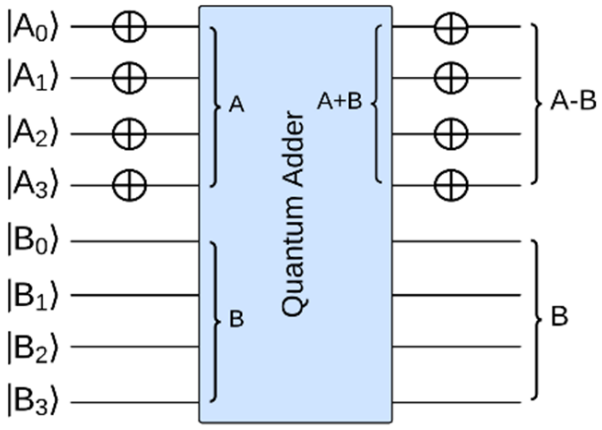


Figure 3.10: A 4-qubit QSUB based on a quantum adder ($A - B = \overline{\overline{A} + B}$).

3.5.3.1 Restoring Approximate Quantum Division with Dynamic Circuit

The restoring division circuit relies on two major components: a QSUB and a QCA. A QSUB is built out of an adder using quantum NOT gates at one of its input operands, as well as the output. Figure 3.10 shows the architecture of a QSUB. The QSUB does not require any T-gate outside of the adder circuit. Thus, the T-count and the T-depth of the QSUB are the same as the adder. The adder block in Figure 3.10 can be replaced by NC-AQANC, NC-AQAFC, NC-AQAHC, or NC-AQAOC represented in Figure 3.9.

Figure 3.11 showed a 4-qubit QCA based on the C-AQAFC architecture that is similar to the C-AQAFC represented in section 3.3. If the $|\text{Ctrl}\rangle$ qubit is in the state $|1\rangle$, then the circuit computes $|S\rangle = |A\rangle + |B\rangle$; otherwise, the register $|B\rangle$ remains unchanged. Quantum AND gates are employed to enable or disable the addition operation. The AND gate labeled "1" in Figure 3.11 is used for the generation of the carry signal. The AND gate labeled "2" sets $|S_1\rangle = |A_1\rangle + |B_1\rangle$ if the $|\text{Ctrl}\rangle$ qubit is in the state $|1\rangle$; otherwise, $|S_1\rangle = |B_1\rangle$. Similarly, other approximate adders can also be extended to QCA implementations.

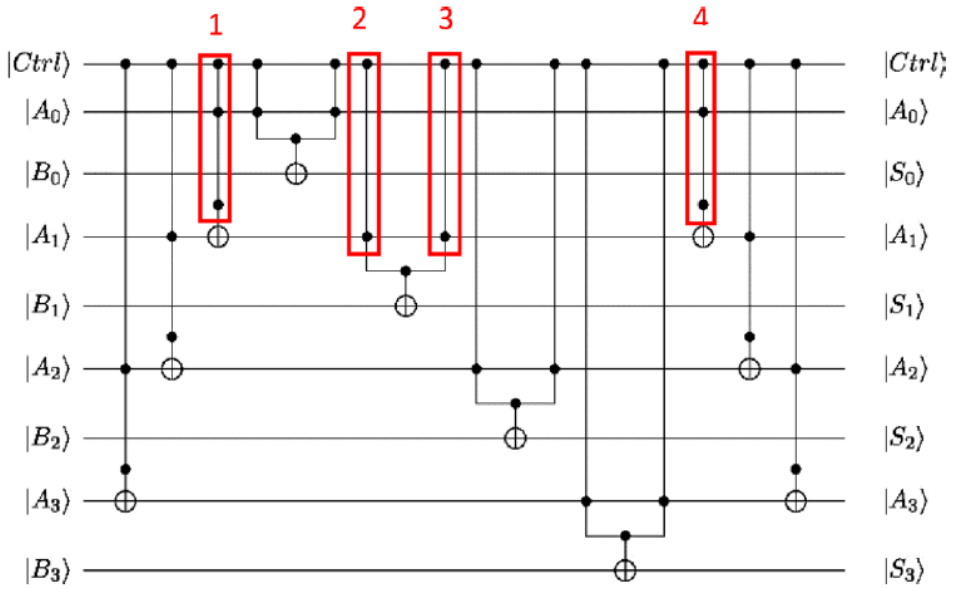


Figure 3.11: A 4-qubit C-AQAFC.

3.5.3.2 Nonrestoring Approximate Quantum Division with Dynamic Circuit

The non-restoring division circuit consists of three main blocks: a QSUB, a QCAS, and a QCA. The approximate QSUB and QCA were explained in previous sections; therefore, in this section, we focus only on the approximate QCAS module. Figure 3.12 shows a 4-qubit QCAS circuit. The $|\text{Ctrl}\rangle$ qubit determines whether the circuit performs addition or subtraction. If the $|\text{Ctrl}\rangle$ qubit is in the state $|1\rangle$, the CNOT gates before the adder compute $\overline{|A\rangle}$, and the CNOT gates after the adder complement the output of the adder, resulting in $|S\rangle = |A\rangle - |B\rangle$. Conversely, if $|\text{Ctrl}\rangle = |0\rangle$, the CNOT gates leave the target states unchanged, and the circuit computes $|S\rangle = |A\rangle + |B\rangle$.

The QCAS circuit does not require any additional T-gates beyond those potentially used in the adder itself. Consequently, the T-count and T-depth of the circuit in Figure 3.12 are the same as those of the adder. Furthermore, the adder block in Figure 3.12

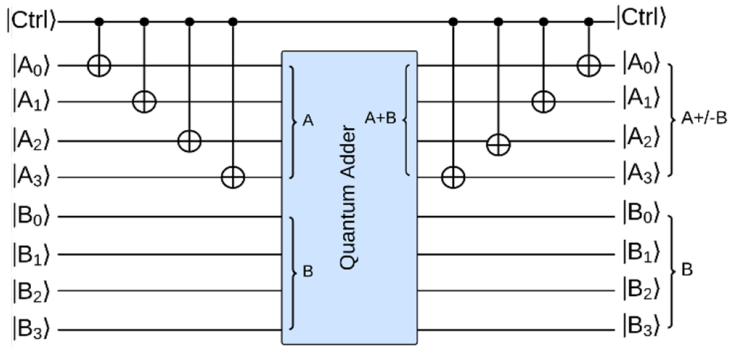


Figure 3.12: A 4-qubit QCAS.

can be replaced by any of the four approximate adders shown in Figure 3.9. We named each approximate divider according to the type of adders employed in their QSUB, QCAS, and QCA blocks and defined Approximate Quantum Divider with No-Carry (AQDNC), Approximate Quantum Divider with Full Carry (AQDFC), Approximate Quantum Divider with Half Carry (AQDHC), and Approximate Quantum Divider with One Carry (AQDOC) accordingly.

Table 3.6 shows NMED for restoring and non-restoring approximate quantum circuits when n is equal to 8. We report errors in two configurations. In the first configuration, all components within the quantum division circuits use approximate adders. In the second configuration, only the first block is exact, and the rest are approximate. Ideally, we would like to have all components approximate to simplify circuits as much as possible. However, as the table shows, approximating all components causes too much error in the outputs of the circuits. As an example, the errors in both restoring and non-restoring circuits based on AQDNC and AQDOC are over 66%. To mitigate the impact of approximation on error, we modeled the second configuration where the first component is exact, and the rest are approximate. The first component generates the MSQ of the result, and thus its impact on the accuracy is higher than the rest of the components. That is why we decided to use the

Table 3.6: Performance comparison of restoring and non-restoring division with different approximate quantum adders.

Adder Type	**Restoring Division**		**Non-Restoring Division**	
	All Approx.	**1st Exact rest Approx.**	**All Approx.**	**1st Exact rest Approx.**
AQDNC	0.663699	0.652422	0.661999	0.372309
AQDFC	0.422062	0.412383	0.516853	0.035224
AQDHC	0.420479	0.410836	0.529830	0.045586
AQDOC	0.663699	0.652422	0.661999	0.372309

exact structure only for the first component. Across all configurations for approximation, AQDFC and AQDHC for the non-restoring division result in an acceptable level of error. Between AQDFC and AQDHC, we select AQDHC for the rest of the paper as it cuts the size of the carry-path by half.

3.6 Quantum Square Root

We propose a new non-restoring quantum square root circuit that reduces the cost of implementation compared with previous circuits [51],[44],[43],[47]. The implementation cost is evaluated in terms of the number of T gates, T-depth, and the total qubits required for the algorithm. Additionally, we have optimized the design by reducing the number of two-qubit gates, including CNOT and SWAP gates. We also exploit approximate computing to simplify the quantum square root circuit further and implement it on an IBM quantum computer.

3.6.1 Exact Quantum Square Root

There are various methods for calculating the square root of a number [103],[104],[52]. These methods can be categorized into two distinct groups: The first method estimates the Square Root (SQR) of a number and is iterative in nature. An example is the Newton-Raphson [104] method, in which the SQR operation starts with an initial estimation of the result. The initial value is then revised through a series of recursive operations. Finally, the algorithm generates the square root of the number. The accuracy of the result depends on the number of iterations. If the algorithm does not run for a sufficient number of iterations, this approach causes a significant error. The second approach computes the square root of a number step by step and generates a qubit of the result in each step. The total number of steps is equal to the size of the result. An example of this approach is restoring [52] SQR algorithms. The precision of this technique is fixed as the number of steps is predetermined. In this work, we focus on the second method for computing the SQR of a number, as the precision of the result is predictable.

There are two main methods for computing the square root of a number with deterministic precision: restoring [52] and non-restoring [103]. The restoring algorithm for the SQR operation guarantees a qubit for the result in each step. It then verifies whether the guess was correct. If it was incorrect, it nullifies the arithmetic operations performed with the incorrect qubit and corrects the guess. Let's assume that $A = \mathrm{A_7A_6}\cdots\ \mathrm{A_1A_0}$. For every pair of qubits in A, the integer part of SQR has one qubit. Thus, the integer part of the result contains four qubits: $Q = \mathrm{Q_3Q_2Q_1Q_0}$. At the start of the algorithm, the result is initialized to 0 (Q=0). The algorithm then iterates four times and generates a qubit of Q in each iteration, starting from Q_3. In iteration i, Q_i is set to 1 and then $Q \times Q$ is subtracted from A. If the result is negative, setting Q_i to 1 made Q too large. Thus, Q_i is

Table 3.7: An Example of Restoring SQR Algorithm

Initialization: $A = 28_d = (011100)_2$, $Q = (000)_2$
$i = 2$ $Q = 100 \Rightarrow A - Q \times Q = (001100)_2$ positive
$i = 1$ $Q = 110 \Rightarrow A - Q \times Q = (111000)_2$ negative, reset Q_1 to 0
$i = 0$ $Q = 101 \Rightarrow A - Q \times Q = (000011)_2$ positive
Outputs $\Rightarrow R = 3_d = (11)_2$, $Q = 5_d = (101)_2$.

reset to 0. This algorithm modifies a qubit twice if the guess is incorrect. Consequently, this approach is called the restoring SQR algorithm. Table 3.7 presents an example of the restoring algorithm when A has six qubits.

The second method, which offers deterministic precision for SQR, is called the non-restoring algorithm. Algorithm 4 shows the details of the algorithm. The algorithm computes the SQR for an n-qubit number ($|A\rangle$). $|A\rangle$ is a positive number, represented in 2's complement format, and has an even number of qubits. At the end of the algorithm, quantum register $|Q\rangle$ with $\frac{n}{2}$ qubits holds the result ($|Q\rangle = \sqrt{|A\rangle}$), and quantum register $|R\rangle$ with n qubits stores the remainder of the computation so that $|A\rangle = |Q\rangle^2 + |R\rangle$. In each iteration, the algorithm generates a qubit of $|Q\rangle$. The partial remainder generated in each iteration guides the next iteration of the algorithm. If the partial remainder is positive or zero, $|Q_i\rangle$ is set to $|1\rangle$ (line 14); otherwise, it is set to $|0\rangle$ (line 16). This is in contrast to the restoring algorithm [52], where an additional step is needed for a negative partial remainder to revert to its value in the previous step. Once the loop terminates, if the remainder is not negative, it is the final remainder. However, if it is negative, an additional step is required to generate the correct remainder (line 20).

Algorithm 4 Non-Restoring Square Root Algorithm

1: **Input:** $A = A_{n-1}A_{n-2}\dots A_1A_0$
2: **Output:** $Q = \sqrt{A}$, $R = A - Q^2$
3: $Q \leftarrow 0^{n/2}$ $\{n/2$ zeros$\}$
4: $R \leftarrow 0^n$ {Initialize remainder to zero}
5: **for** $i = n/2 - 1$ **downto** 0 **do**
6: **if** $R \geq 0$ **then**
7: $T \leftarrow R_{n-2i-3}\dots R_0a_{2i+1}a_{2i}$
8: $R \leftarrow T - 0^{n/2-i-1}Q_{n/2-1}Q_{n/2-2}\dots Q_{i+1}01$
9: **else**
10: $T \leftarrow R_{n-2i-3}\dots R_0a_{2i+1}a_{2i}$
11: $R \leftarrow T + 0^{n/2-i-1}Q_{n/2-1}Q_{n/2-2}\dots Q_{i+1}11$
12: **end if**
13: **if** $R \geq 0$ **then**
14: $Q_i \leftarrow 1$
15: **else**
16: $Q_i \leftarrow 0$
17: **end if**
18: **end for**
19: **if** $R < 0$ **then**
20: $R \leftarrow R + 0^{n/2-1}Q_{n/2-1}Q_{n/2-2}\dots Q_01$
21: **end if**
22: **return** Q, R

In this work, we use the non-restoring algorithm presented in the algorithm 4 as the baseline scheme [103] because it does not need to nullify arithmetic operations in the event of a negative remainder. In addition, it utilizes operations such as add and subtract, which are less costly and more efficient than some other SQR algorithms [44].

Table 3.8 presents an example of how the algorithm calculates the SQR of a number. In this example, the input is 28 with a qubit length of 6 (n=6). The first iteration is slightly simpler than the other iterations. In the first iteration, R is not negative (line 6) and so lines 10-11 are never executed. We use this property of the first iteration of the algorithm

Table 3.8: Non-Restoring SQR of $28_d = (011100)_2$ Based on Algorithm 4

R	Q
Initial 0	0
$i = 2 \Rightarrow (01)_2 - (01)_2 = (00)_2$	,$Q_2 = 1$
$i = 1 \Rightarrow (0011)_2 - (0101)_2 = (1110)_2$	,$Q_2 = 1$, $Q_1 = 0$
$i = 0 \Rightarrow (111000)_2 + (001011)_2 = (000011)_2$	,$Q_2 = 1$, $Q_1 = 0$, $Q_0 = 1$
Outputs $\Rightarrow R = 3_d = (11)_2$, $Q = 5_d = (101)_2$.	

to simplify the circuit of SQR. From the second iteration, the algorithm shifts R to the left and inserts two new qubits into the LSB, taken from input A. Then, based on the sign of R, it assigns either 1 or 0 to Q_i. Once the algorithm exits the loop, R is positive and thus it skips adjusting R (line 20).

To make this paper self-explanatory, we prove the correctness of the algorithm in this section. Let's assume that in step k (please note that step k is equal to iteration $\frac{n}{2} - k$ where $k = 1, 2, 3, \cdots$), r_k (generated in line 8 or 11) represents the remainder, and $q_k = \mathrm{Q}_{\frac{n}{2}-1}\mathrm{Q}_{\frac{n}{2}-2}\cdots\ \mathrm{Q}_{\frac{n}{2}-k}$. We prove the correctness of the algorithm through induction over k.

The first iteration of the loop in Algorithm 4 is trivial because A is a positive number in 2's complement format. Assuming that the algorithm works for $k = j$, we should prove that it works for $k = j + 1$.

In step j, let's assume that the two qubits taken from A and concatenated with r_{j-1} are ab ($a = A_{n-2j+1}, b = A_{n-2j}$).

In step $k = j + 1$, If $r_j \geq 0$, then the next two qubits of A (let's say cd, $c = A_{n-2j-1}$, $d = A_{n-2j-2}$) are concatenated with r_j and r_{j+1}, is computed as follows (lines 5-6):

$$r_{j+1} = r_j cd - q_j 01 \tag{3.7}$$

if $r_{j+1} \geq 0$, then $q_{j+1} = \mathrm{q}_j 1$ (line 14). However, if $r_{j+1} < 0$, then $q_{j+1} = \mathrm{q}_j 0$. If $r_j < 0$, then r_j should be adjusted, and the subtract operation in the previous iteration should be canceled. Thus:

$$r_{j+1} = r_{j-1}abcd - q_j 01 \tag{3.8}$$

From iteration $k = j$, we have:

$$\begin{aligned} r_j cd &= 4 \times r_j + cd = 4 \times (r_{j-1}ab - q_{j-1}01) + cd \\ &= r_{j-1}abcd - q_{j-1}0100 \end{aligned} \tag{3.9}$$

From (3):

$$r_{j-1}abcd = r_j cd + q_{j-1}0100 \tag{3.10}$$

We can rewrite (2) by substituting r_{j-1}abcd from (4):

$$\begin{aligned} r_{j+1} &= r_j cd + q_{j-1}0100 - q_j 01 \\ &= r_j cd + q_j 100 - q_j 01 \\ &= r_j cd + (q_j \times 8 + 100) - (q_j \times 4 + 01) \\ &= r_j cd + q_j \times 4 + 11 = r_j cd + q_j 11 \end{aligned} \tag{3.11}$$

Equation (5) is the same as lines 8-9 in the algorithm. Thus, we proved the correctness of the algorithm for iteration $j + 1$.

3.6.2 Proposed Enhanced Exact SQR Circuit

Algorithm 4 processes a radicand from the MSQ and generates a qubit for the result per iteration. In step k, the algorithm either subtracts or adds two numbers. The size of the

two numbers is equal to $2k$, which requires a $2k$-qubit adder/subtractor. However, some of these qubits are not needed because the range of the numbers added or subtracted in step k is small, and it is feasible to generate correct results using adders/subtractors smaller than $2k$-qubit. We mathematically prove this:

Algorithm 4 generates correct results in each iteration. In other words, in step k: $R_k = A - {Q_k}^2$, where R_k and Q_k are the remainder and SQR in step k, respectively. For Q_k to be an SQR in step k, $(Q_k + 1)^2$ should be greater than A:

$$\begin{aligned} (Q_k + 1)^2 > A &=> {Q_k}^2 + 2Q_k + 1 > A \\ &=> 2Q_k + 1 > A - {Q_k}^2 \end{aligned} \tag{3.12}$$

However, $R_k = A - {Q_k}^2$, thus:

$$R_k < 2Q_k + 1 \tag{3.13}$$

Because R_k is an integer, we have the equation: $R_k \leq 2Q_k$. Thus, the maximum value for the remainder is twice the SQR obtained in step k. In a non-restoring SQR algorithm, the remainder of step k can be a negative number. This occurs when the computed SQR is one unit greater than the actual SQR. Assuming that a negative remainder in step k is represented by R'_k, then:

$$R'_k = A - (Q_k + 1)^2 = R_k - (2Q_k + 1) \tag{3.14}$$

From this equation, we can extract the number of qubits required to hold a negative remainder. Since Algorithm 4 generates one qubit per iteration, the size of Q_k is k-qubit.

Given that R'_k is a negative number, the smallest value for R'_k occurs when $R_k = 0$:

$$R'_{\text{kmin}} = -(2Q_k + 1) \tag{3.15}$$

From (9), we can compute the number of qubits needed for a negative remainder in the worst-case scenario. In step k, $(k + 2)$ qubits are required for a negative remainder. Thus, the non-restoring SQR algorithm requires $(k+2)$-qubit adder/subtractor in step k to compute the correct results. This implies that there is room for improvement of the original algorithm. The number of adder/subtractor qubits can be reduced from $(\sum_{k=2}^{n/2} 2k = \frac{n^2}{4} + \frac{n}{2} - 2)$ to $(\sum_{k=2}^{n/2}(k + 2) = \frac{n^2}{8} + \frac{5n}{4} - 3)$.

Quantum Circuit for the Enhanced Exact SQR Algorithm

In this section, we propose a quantum circuit for the enhanced exact QSQR algorithm. The proposed circuit is generic and can be used to implement a QSQR operator of any input size. For the sake of discussion, we show a 6-qubit QSQR circuit in Figure 3.13. This circuit is scalable and can be generalized for larger numbers. Three registers derive the inputs of the circuit: quantum register $|A\rangle$ with n qubits, quantum register $|B\rangle$ with $(\frac{n}{2} + 2)$ qubits, and quantum register $|\text{Ctrl}\rangle$ with one qubit. For the example presented in Table 3.8, the quantum register $|A\rangle$ is initialized to 28. The other two registers are initialized with zeros. At the end of the algorithm, $|A\rangle$ holds the final remainder (R in Algorithm 4) and the upper $\frac{n}{2}$ qubits of $|B\rangle$ hold $\sqrt{|A\rangle}$ (Q in Algorithm 4). Overall, the circuit requires $n + \frac{n}{2} + 2 + 1 = \frac{3n}{2} + 3$ qubits to compute the QSQR of an n-qubit number.

The circuit is composed of quantum gates such as CNOT and blocks such as QCAS and QCA. A QCAS adds or subtracts its inputs based on the value of the control signal. If the control signal is in the state $|0\rangle$, then the block performs an add operation; otherwise,

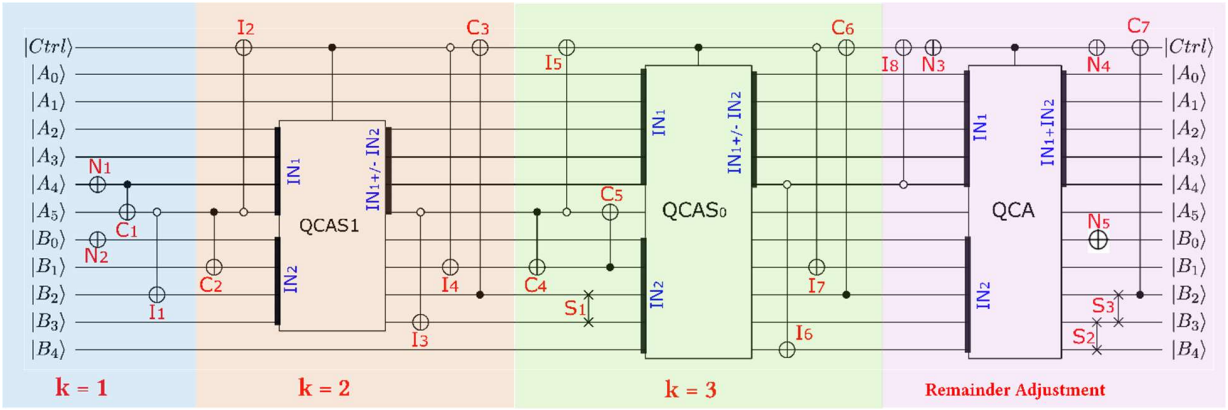


Figure 3.13: Enhanced exact QSQR circuit.

it subtracts the inputs. A QCA computes the sum of the inputs if its control signal is in the state $|1\rangle$; otherwise, it does not change its inputs.

The proposed QSQR circuit is divided into three parts:

1. **Part 1, k=1**: This part executes the first iteration ($i = \frac{n}{2} - 1$) of the loop and generates the MSQ of Q. Because R is initialized to zero in Algorithm 4, the first iteration of the loop never executes the Else section of the If-statement (lines 8-9). Thus, the first iteration requires a simpler circuit to realize than the other iterations do.

2. **Part 2, k=2 to n/2**: This part executes the remaining $\frac{n}{2} - 1$ iterations of the loop. The two main operations in this part of the circuit are subtraction and addition, which correspond to lines 6 and 9 of Algorithm 4, respectively.

3. **Part 3, Remainder adjustment**: This part of the circuit corresponds to the if statement that follows the loop in Algorithm 4.

The detailed quantum circuit design of parts 1, 2, and 3 required to implement Algo-

rithm 4 is explained as follows:

Part 1 (k=1): This part is composed of four quantum gates. A quantum NOT gate is applied to $|A_{n-2}\rangle$ (N_1) and another quantum NOT gate is applied to $|B_0\rangle$ (N_2). A CNOT gate (C_1) is applied to locations $|A_{n-2}\rangle$ and $|A_{n-1}\rangle$ so that the location $|A_{n-2}\rangle$ is not changed while location $|A_{n-1}\rangle$ now has the value $|A_{n-2} \oplus A_{n-1}\rangle$. An inverted CNOT gate (I_1) is applied to locations $|A_{n-1}\rangle$ and $|B_2\rangle$ so that the state of $|A_{n-1}\rangle$ remains the same but the state of $|B_2\rangle$ changes to $|\overline{A_{n-1}} \oplus B_2\rangle$.

Since the input is a positive number, $|A_{n-1}\rangle$ is in the state of $|0\rangle$ ($|A_5\rangle = |0\rangle$ in Figure 3.13). However, $|A_{n-2}\rangle$ can be either $|0\rangle$ or $|1\rangle$. For $n = 6$, if $|A_5A_4\rangle$=$|01\rangle$, then N_1 and C_1 change states of $|A_5A_4\rangle$ to $|00\rangle$. However, if $|A_5A_4\rangle$=$|00\rangle$, then $|A_5A_4\rangle$=$|11\rangle$. This is expected as in the first iteration of the algorithm, $|01\rangle$ is subtracted from R (line 8). At the end of the first iteration (lines 10-13), a qubit of the result is generated. In Figure 3.13, I_1 generates the qubit and stores it into $|B_2\rangle$. Later, the state of $|B_2\rangle$ is transferred into $|B_4\rangle$ through S_1 and S_2. N_2 sets $|B_0\rangle$ to $|1\rangle$ and its state remains the same for the rest of the algorithm (lines 6 and 9).

Part 2 (k=2 to n/2): The building block in the second step of the algorithm is the QCAS. Before each QCAS, a few quantum gates set up the QCAS inputs. After each QCAS, there are additional quantum gates to set Q_i in Algorithm 4 and restore the state of $|\text{Ctrl}\rangle$ and some other qubits. A quantum CNOT gate is applied to MSQ of the remainder generated in the previous step and $|B_1\rangle$, where the target terminal of the CNOT gate is connected to $|B_1\rangle$. An inverted CNOT gate is applied to the MSQ of the remainder from the previous step and the $|\text{Ctrl}\rangle$. $|\text{Ctrl}\rangle$ qubit is the target terminal of the inverted CNOT gate. From k=3, a series of SWAP gates moves the high-order qubits of Q downward (S_1). In addition, a CONT gate (C_5) is applied to $|B_1\rangle$ and MSQ of the remainder generated in the previous step so that $|B_1\rangle$ is connected to the control terminal of the CNOT.

Once the inputs of a QCAS are assigned, the QCAS is applied to:

$A_{n-(k-1)} \sim A_{n-(k-1)-(k+1)}$ and $B_{k+1} \sim B_0$.

At the output of the QCAS, an inverted CNOT is applied to MSQ of the output of the QCAS and $|B_{k+1}\rangle$, where the target terminal of the gate is applied to $|B_{k+1}\rangle$. An inverted CNOT is connected to $|\text{Ctrl}\rangle$ and $|B_1\rangle$ where $|\text{Ctrl}\rangle$ is connected to the control terminal. A CNOT gate is applied to locations $|B_2\rangle$ and $|\text{Ctrl}\rangle$ where the target terminal is connected to $|\text{Ctrl}\rangle$.

To clarify part 2 of the circuit presented in Figure 3.13, we explain $k = 2$ and $k = 3$ separately. For $k = 2$, the sign of R determines whether B should be added/subtracted to/from A (line 6). The MSQ of A indicates the sign of A. If it is in state $|0\rangle$, A is positive and the state of $|\text{Ctrl}\rangle$ changes to $|1\rangle$ (QCAS_1 performs subtract operation). In addition, C_2 sets $|B_1\rangle$=$|0\rangle$ (line 8). However, if it is in state $|1\rangle$, A is negative, I_2 sets $|\text{Ctrl}\rangle$=$|0\rangle$, and C_2 sets $|B_1\rangle$=$|1\rangle$. The MSQ of the QCAS_1's output indicates the sign of the remainder in the second iteration of the loop (lines 10-13). I_3 sets $|B_3\rangle$ based on MSQ. At the end of the algorithm, $|B_3\rangle$ holds the second MSQ of the result (Q_1). Before starting the next iteration, I_4 and C_3 restore $|B_1\rangle$ and $|\text{Ctrl}\rangle$ to their initial states, respectively.

For the third iteration ($k = 3$), C_4 (similar to C_2) sets $|B_1\rangle$ to $|0\rangle$ or $|1\rangle$ based on the MSQ of the QCAS_1's output. I_5 determines the control signal for the QCAS_0. The SWAP gate S_1 moves $|B_2\rangle$ towards the bottom of the circuit as $|B_2\rangle$ holds MSQ of Q. C_5 restores $|A_5\rangle$ to its value right before QCAS_1. QCAS_0 computes either the subtraction or the sum of its inputs based on $|\text{Ctrl}\rangle$. I_6 generates the LSQ of the final result (Q_0) and stores it into $|B_4\rangle$. I_7 and C_6 restore the state of $|B_1\rangle$ and $|\text{Ctrl}\rangle$ to their initial values. Eventually, S_2 and S_3 transfer $|B_4\rangle$ to $|B_2\rangle$.

Part 3 (Remainder Adjustment): Once the loop is terminated, an additional step

is required if the remainder is negative. This part occurs only once. An inverted CNOT gate is applied to the MSQ of the output of the previous QCAS and $|\text{Ctrl}\rangle$, where $|\text{Ctrl}\rangle$ is connected to the target terminal of the inverted CNOT. A NOT gate is applied to the $|\text{Ctrl}\rangle$. A QCA is applied to the lower $\frac{n}{2}+2$ qubits of A and B. A NOT gate is applied to $|\text{Ctrl}\rangle$ and another one is applied to $|B_0\rangle$. A series of SWAP gates moves Q_0 generated by the last QCAS to $|B_2\rangle$. A CNOT gate is applied to $|B_2\rangle$ and $|\text{Ctrl}\rangle$ where $|\text{Ctrl}\rangle$ is connected to the target terminal.

For the circuit in Figure 3.13, I_8 and N_3 together determine the control signal for QCA based on the sign of the QCAS_0 output. If the output is negative, the control signal is set to $|1\rangle$ so that QCA adjusts R (line 20). N_4 and C_7 restore $|\text{Ctrl}\rangle$ back to $|0\rangle$. Similarly, N_5 restores the state of $|B_0\rangle$.

3.6.3 Proposed Approximate QSQR Circuit

The exact QSQR circuits described in Sections 3.6.1 and 3.6.2 are not practical because their implementation on an NISQ device leads to excessive noise. Noise is the result of the number of quantum gates as well as the depth of the circuit. One approach to reduce the complexity of the circuit is approximate computing. Among the different components depicted in Figure 3.13, QCAS and QCA are amenable to approximations. The building block of the QCAS and QCA is an adder. Even the core of a subtractor is an adder ($A-B=(\overline{\overline{A}+B}$). Thus, to simplify the QSQR circuit, we need to approximate the adder circuit.

An n-qubit adder is composed of n 1-qubit adders, where the 1-qubit adders are connected back-to-back. Each 1-qubit adder generates a carry-out that derives the carry-in of the next 1-qubit adder. Because the carry path spans from the LSQ to the MSQ of the

n-qubit adder, it is the longest path in the adder and thus forms the critical path. The number of approximate QCAS blocks in an QSQR circuit is a trade-off between the accuracy and depth of the circuit. In the extreme case where all QCAS blocks are approximate, the depth of the circuit is minimal; however, such an approximate circuit causes the highest amount of error. We evaluated different levels of approximation on the carry-path and found that approximating the most significant QCAS ($QCAS_1$ in Figure 3.13) causes too much error. This is expected because the weight of Q_i generated by the most significant QCAS is higher than that of all other qubits generated by other QCAS blocks. Thus, we use an exact circuit only for the most significant QCAS to avoid excessive error. However, for the other QCAS blocks, we use approximate circuits.

Different types of approximate adders have been reported in the section 3.3. All these approximate adders change the carry-path to trade off the depth of the circuit in exchange for accuracy. We evaluated these different approximate adders in the context of the QSQR circuit and found that the error injected into the approximate QSQR circuit is very similar. Regardless of the type of approximate adder used, the accuracy of the corresponding QSQR circuits remains almost the same. This is mostly due to the masking property of digital circuits. Thus, we select the simplest approximate adders, which are C-AQANC and NC-AQANC, to reduce the depth of the QSQR circuit.

Figure 3.14 shows the structure of a QCAS based on NC-AQANC. In a QCAS block, if the $|\text{Ctrl}\rangle$ signal is in the state of one, the block computes $(|A\rangle - |B\rangle)$ and stores the result into $|A\rangle$; otherwise, it computes $|A\rangle + |B\rangle$ and stores the result into $|A\rangle$. Since there is no need to calculate the carry propagated through the circuit, the sum of $|A\rangle$ and $|B\rangle$ is computed by the exclusive-or of $|A\rangle$ and $|B\rangle$ (the box labeled 2 in Figure 3.14). Subtract of $|B\rangle$ from $|A\rangle$ $(|A\rangle - |B\rangle)$ is the same as $\overline{\overline{|A\rangle} + |B\rangle}$. The first box in Figure 3.14 computes the complement of $|A\rangle$ and the last box computes the complement of $(\overline{|A\rangle} + |B\rangle)$.

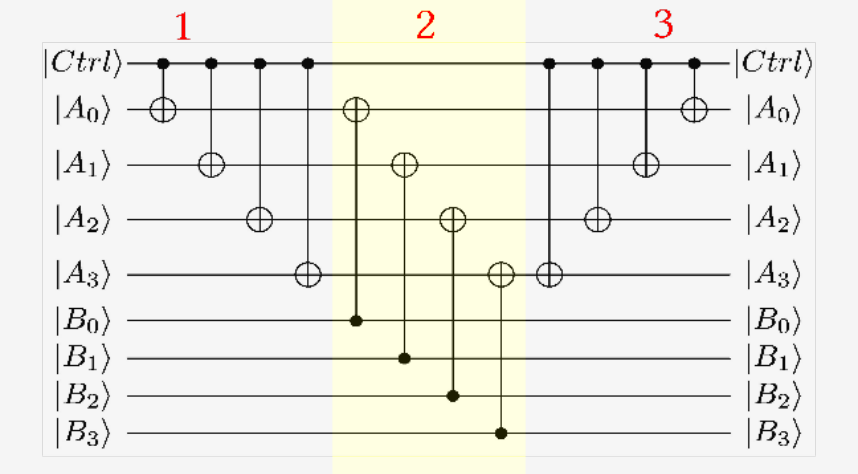


Figure 3.14: Structure of a 4-qubit approximate QCAS

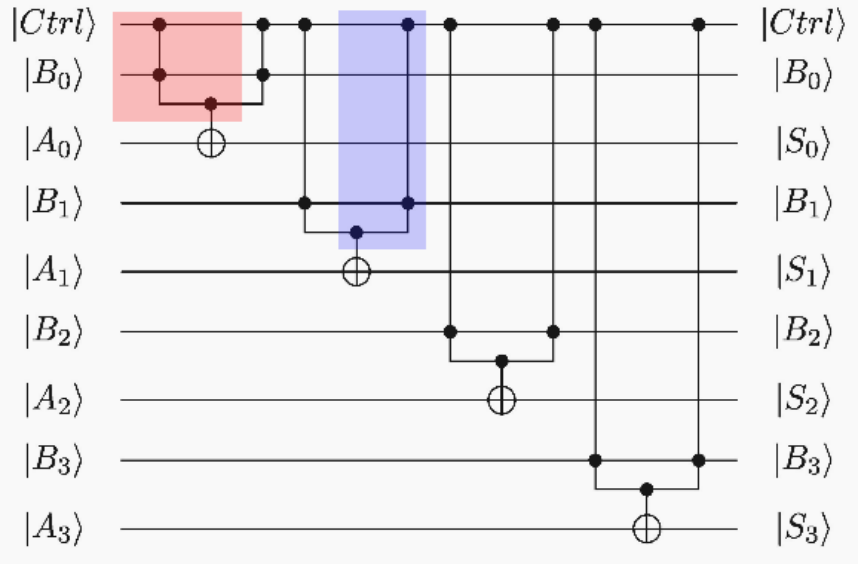


Figure 3.15: A 4-qubit approximate QCA.

The proposed approximate QCAS in Figure 3.14 is T gate-free and is composed of only CNOT gates. This reduces the complexity of the QCAS circuit and renders it suitable for implementation on NISQ devices.

Figure 3.15 shows a 4-qubit approximate QCA based on C-AQANC design. If the $|\text{Ctrl}\rangle$ signal is in the state of one, then $|S\rangle = |A\rangle + |B\rangle$; otherwise, $|S\rangle = |A\rangle$. The CNOT gates compute $|A\rangle + |B\rangle$ if the $|\text{Ctrl}\rangle$ signal is in the state of one. The blue box in Figure 3.15 is used for the uncomputation of the corresponding AND gate to ensure the reversibility of the QCA. The only component in Figure 3.15 that uses the T gate is the logical AND gates. Thus, an n-qubit QCA has a T-count of $4n$. Since C-AQANC eliminates the carry-path, calculating the sum of $|A_i\rangle$ and $|B_i\rangle$ is performed in parallel. Consequently, the depth of the circuit is $O(1)$, which is much smaller than the depth of an exact adder [51],[44], [43], [47].

We compare the accuracy of the proposed approximate QSQR with an exact QSQR using two error metrics: MED and NMED [99]. Table 3.9 lists the MED and NMED for a 10-qubit approximate QSQR circuit. The details about the calculation of MED and NMED are explained in section 3.2.

Table 3.9: Error of Approximate QSQR Circuit

Circuit Type	MED	NMED
Approximate QSQR Circuit	1.300781	0.086719

Chapter 4

Quantum Neural Network

In this chapter, we first describe the architecture of the QNN employed in this work, detailing its circuit structure, parameterization strategy, and the role of variational layers in learning complex quantum representations. We outline how data is encoded into quantum states, the sequence of single- and two-qubit operations used, and the measurement scheme adopted to extract classical outputs. Following this, we introduce the proposed Noise Resilient Quantum Neural Network (NR-QNN), explaining its motivation, key design components, and how it differs from conventional QNN architectures. In particular, we discuss the incorporation of noise-resilience mechanisms, circuit-level optimizations, and pruning strategies that enable the model to maintain high performance even under realistic hardware noise. Together, these sections provide the necessary foundation for understanding the methodological contributions and experimental results presented in the subsequent chapters.

Next, we present the approximate QRAM module incorporated into our design. We discuss its role in efficiently encoding classical data into addressable quantum memory

while reducing circuit depth and gate overhead compared to full QRAM constructions. The approximate QRAM enables scalable data access for downstream QNN operations by relaxing exact amplitude encoding requirements and introducing controlled approximations that preserve relevant information while improving hardware feasibility.

4.1 Architecure of Noise Resilient Quantum Neural Network

A QNN consists of parametric quantum gates that should be trained to generate a desired input-output relationship. The QNN is modeled by $\Phi(x, \theta)$ where x is the input data and θ is a set of parameters for adaptive optimization. Since on IBM quantum computers, the default state for a qubit is $|0\rangle$, a QNN that is run on an IBM quantum computer maps $|0...0\rangle$ to $\Psi(x, \theta)$ where $\Psi(x, \theta) = \Phi(x, \theta)\,|0\ldots0\rangle$.

Figure 4.1 shows the structure of a QNN, and a layer of PQC in QNN. The first step in the QNN is encoding, where classical data are converted to quantum states. For a continuous variable, a popular technique is angle encoding, where the continuous variable is encoded as an angle of rotation along the desired axis (X/Y/Z). As a state produced by a quantum rotation gate around any axis will repeat itself in 2π intervals, classical data are scaled to 0 to 2π range in a data preprocessing step. At the beginning of the encoding circuit, a Hadamard gate changes the state of an input qubit to a superposition of $|0\rangle$ and $|1\rangle$. This allows the encoding circuit to cover a larger subset of the Hilbert space.

An effective approach to designing a low-depth and accurate QNN is to consider circuits that prepare strongly entangled quantum states. An entangled circuit is capable of reaching wide corners in the Hilbert space. In other words, they have a better chance to project

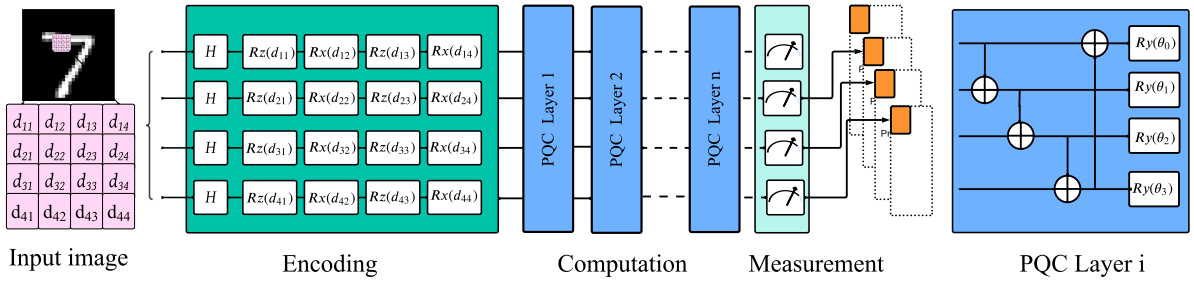


Figure 4.1: Architecture of a QNN.

classical input x to state $\Psi(x, \theta)$ which generates the correct label y when it is measured. From a theoretical point of view, a classifier should be able to capture both short- and long-term correlations. There is clear evidence that a shallow circuit is suitable for this purpose if it is strongly entangled [105]. Figure 4.1 shows an example of an entangled QNN implemented using a PQC. The entanglement part is realized using a set of multi-qubit operations between the qubits, such as CNOT, to generate correlated states. The rotation gates in the PQC search through the solution space.

Once the outputs of a PQC are measured, they are fed to a dense layer [77]. The number of neurons in the dense layer is equal to the number of classes in the dataset. We consider a stochastic descent method for training the QNN. We choose a Multi-Class Cross-Entropy Loss function to evaluate the error of the QNN:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij}.log(\hat{y}_{ij}). \tag{4.1}$$

Where N denotes the number of samples in the input dataset, and C represents the total number of classes. For each sample i and class j, y_{ij} corresponds to the correct label, typically encoded as a one-hot vector, while $\hat{y}_{ij}$ represents the predicted probability

output of the QNN. The cross-entropy loss penalizes discrepancies between the predicted probabilities and the true labels, effectively guiding the training process. In the context of classification, the QNN transforms classical input data into a quantum state, processes it through a sequence of parameterized quantum gates, and outputs a probability distribution over the C classes. Minimizing this loss function encourages the network to assign higher probabilities to the correct class labels, improving the model's overall classification accuracy on unseen data.

The training is an iterative process and searches for the best parameters in $\Phi(x, \theta)$. We train the QNN using the Adagrad gradient-based optimization algorithm. To compute optimum values for PQC parameters, the derivative of the QNN outputs with regard to inputs and parameters should be computed. There are a variety of techniques to compute the gradients. However, not all of them are suitable for hardware. As an example, the adjoint method [106] requires circuit intermediate state values to compute gradients that are not accessible when the circuit runs on hardware. The parameter-shift rule [74] is employed in this work to compute gradients of parameterized quantum circuits. This method evaluates the target function at two analytically determined points, enabling the exact calculation of derivatives with respect to circuit parameters whose generators have two distinct eigenvalues. The parameter-shift approach is particularly suitable for quantum circuits because it provides exact gradients without relying on numerical approximations. In our implementation, the parameter-shift method is realized using the PennyLane framework [75], which offers efficient support for both simulation and hardware-based quantum backends. In this work, we use PyTorch [107] and PennyLane [75] frameworks to train QNNs.

4.1.1 Quantum Pruning

NISQ devices are vulnerable to quantum noise, which may compromise the fidelity of the computation results. One of the sources of errors in quantum computers is the coherence error. If the time that it takes for a quantum circuit to compute its output exceeds the coherence time of a qubit, then the output of the circuit is skewed with noise and is not reliable. The computation time of a quantum circuit is correlated with the depth of the circuit [108]. The depth of a circuit is determined by the critical path, which is the longest path composed of serially connected quantum gates in a quantum circuit. Thus, reducing the depth of a quantum circuit is an effective way to decrease computation time, which in turn reduces the noise level. The other source of error in quantum circuits is quantum gates. Each quantum gate adds noise to its qubit, reducing the fidelity of the qubit. Thus, a quantum circuit with a smaller number of gates is more reliable.

An effective way to reduce the depth and the number of gates in QNNs is quantum pruning. We observe that for most rotation gates in the PQC of a QNN, if the angle of rotation is far from zero for several iterations during the training phase of the QNN, it will likely stay far from zero in the next several iterations. Similarly, if the angle remains small for several iterations, it will likely stay small in the next several iterations. Thus, the angle of rotation is reliably predictable to some extent. We propose angle pruning to remove gates with a small angle of rotation. This method potentially reduces the depth of PQCs and reduces the probability of quantum computation time exceeding coherence time. In addition, quantum gate pruning reduces the number of gates, which in turn decreases the impact of gate errors on the fidelity of PQCs. It also decreases the number of parameters to be updated during the training phase and thus accelerates the training of QNNs.

A QNN is trained first to learn the parameters of the corresponding PQC. Next, we

prune gates with small angles: all rotation gates with angles less than a threshold are removed from the network. Finally, we retrain the network to adjust the remaining angles so that they can compensate for pruned gates and maintain the accuracy of the original network. This process is repeated several times to remove gates as much as possible. During the retraining, it is better to retrain the angles with initial values left from the previous round of training rather than reinitializing the surviving gates from scratch. NNs contain fragile co-adapted features [109]: gradient descent can find a decent solution when a network is initially trained, but not after reinitializing network parameters from scratch and retraining them.

To put it formally, let's assume $\Phi(x, \theta)$ represents a QNN where θ is the set of angles in rotation gates. Our objective is to prune the model $\Phi(x, \theta)$ so that rotation gates with small angles are removed. However, the accuracy of the original network is maintained:

$$\Phi(x, \theta) = \Phi(x, \theta_1) \tag{4.2}$$

where $\theta_1 \subset \theta$. The pruning strategy selects gates based on the magnitude of their angles. Mathematically:

$$f_{prune}(\theta_j; \lambda) = \begin{cases} 0 & \text{if } \theta_j < \lambda \\ \theta_j & \text{otherwise} \end{cases} \tag{4.3}$$

Where λ is a predefined threshold. Let's define the sparsity/pruning ratio S of the pruned network as the fraction of angles that are zero:

$$S = \frac{\text{Number of pruned gates}}{\text{Total number of gates}} = \frac{n_{\text{pruned}}}{n_{\text{total}}} \tag{4.4}$$

Where n_{pruned} is the number of pruned gates and n_{total} is the total number of rotation

gates in the original network. The goal is to increase S while maintaining the accuracy of the original network. After pruning a set of weights, the model often undergoes fine-tuning to recover from any potential loss in accuracy. This involves retraining the network with the pruned angles frozen to allow the remaining angles to compensate for the missing gates and connections. Mathematically, after pruning and fine-tuning, the loss function may converge to a new local minimum:

$$\mathcal{L}_{\text{fine-tuned}} = \mathcal{L}(\theta_{\text{pruned}}) + \epsilon \tag{4.5}$$

Where ϵ represents a small adjustment due to fine-tuning. The pruning process involves iterative rounds of pruning and fine-tuning, progressively increasing the sparsity S until no additional gates can be pruned.

Algorithm 5 describes quantum gate pruning for QNNs. We divide the training phase into n epochs and perform pruning periodically at the end of each epoch. First, the gradient of the cost function with regard to the parameters of the PQC is computed using the parameter shift technique [74] (line 11). Then, parameters are updated using the gradient descent method (line 12). Once a round of training with all inputs from the training dataset (S_{train}) is finished (lines 10-13), the pruning phase starts. The angle of each rotation gate in the PQC is compared with a predetermined threshold value (line 16). If the angle is less than the threshold, then the corresponding gate is removed from the circuit. Since pruning a subset of rotation gates from the quantum circuit may degrade accuracy, the network is retrained in the following epoch so that parameters of the remaining gates are updated to compensate for potential accuracy drop. Retraining the pruned network, starting with parameters generated in the previous epoch, requires less computation because it does not need to backpropagate through the entire network. The pruned network is smaller, which

Algorithm 5 Quantum Gate Pruning

```
Inputs:
S_train: training set
C: cost function
η: learning rate
n: number of epochs
θ: angle of rotation
f(θ): output of PQC
thld: Threshold for pruning
for epoch = 1, 2, ..., n do
    for a batch B from S_train do
        ∇_θ C_B(θ) = 0.5(∂f(θ)/∂θ)^T (∂C(θ)/f(θ))
        θ = θ − η∇_θ C_B(θ)
    end for
    //pruning phase
    for each rotation gate g(θ) in PQC layers do
        if θ < thld then
            remove g(θ) from PQC layers
        end If
    end for
end for
Output: Pruned QNN
```

accelerates the training process.

Pruning rotation gates with small angle values is an iterative procedure. One approach for pruning would be training a QNN for n epochs and then pruning the network only once. This method reduces the overhead of pruning but is not able to reach the full potential of pruning. It may generate a network that is smaller than the original network, but it may still contain a large number of unnecessary gates. Pruning a QNN after each epoch removes redundant gates gradually, enables finding a QNN with a smaller number of gates, and increases the success rate of the pruned network when it is deployed into an NISQ device. The iterative procedure described in Algorithm 5 is very similar to the

mammalian brain [110], where synapses are created in the first few months after a child is born. However, during the postnatal development of the child, synapses change based on their efficacy and activity. Synapses with no or little usage experience a postsynaptic gating mechanism, which is similar to the pruning of gates with small angles of rotation. On the contrary, those synapses that are used frequently remain in the nervous system, and their corresponding axon will carry larger electrical signals.

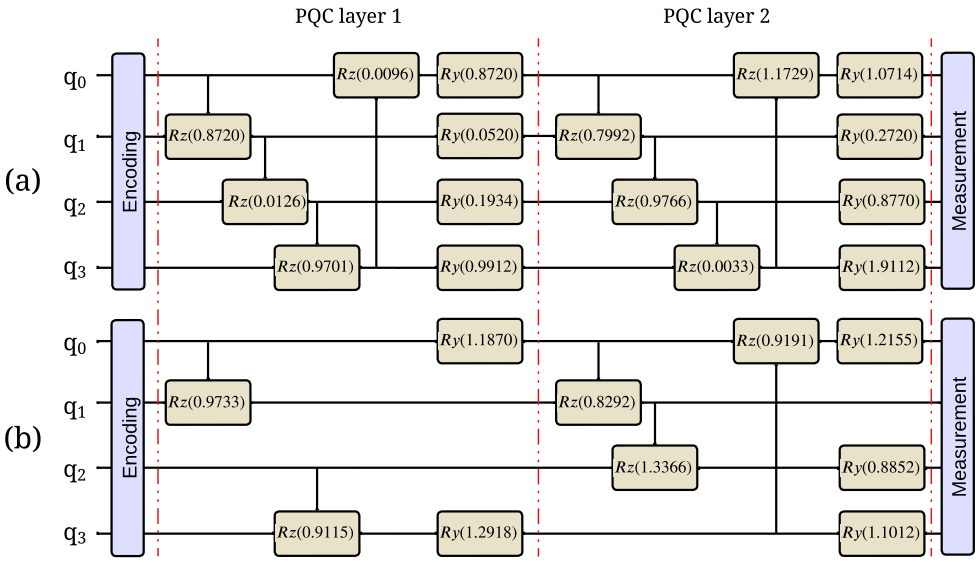


Figure 4.2: An example of QNN pruning for a circuit with two PQC layers where the threshold is $1\pi/4 = 0.7854$. (a) Base QNN circuit. (b) QNN circuit after pruning and fine-tuning.

Figure 4.2 shows an example of QNN pruning for a circuit with two PQC layers using Algorithm 5. Figure 4.2(a) is the original circuit, and Figure 4.2(b) is the pruned circuit. Pruning removes the rotation gates and controlled rotation gates with angles below a predefined threshold. In this example, the threshold is $1\pi/4 = 0.7854$. After each round of pruning, we retrain and fine-tune the circuit to recover accuracy. Pruning only affects the rotation gates in the computation part (PQC layers) of the circuit and does not impact

the rest of the circuit, such as the encoding part.

4.1.2 Sensitivity-Aware Qubit Mapping

It is well-known that the error rates of qubits in NISQ devices vary significantly across physical qubits. In IBM quantum computers, for example, the error rate of a qubit can fluctuate by a factor of up to seven due to process variations, temperature fluctuations, and environmental factors [108]. Consequently, when mapping a quantum circuit onto a NISQ device, it is advantageous to prioritize qubits with lower error rates to maximize computation fidelity. For instance, IBM Brisbane provides 127 physical qubits, but if a quantum circuit involves only 10 logical qubits, selecting the most reliable subset of qubits for execution can substantially reduce the impact of noise and improve the overall accuracy of the computation. Therefore, hardware-aware mapping strategies that account for qubit-specific error rates are essential for effectively deploying quantum algorithms on current NISQ devices.

A functional mapping of a quantum circuit onto NISQ hardware requires first an initial placement of the circuit qubits onto the hardware qubits in order to reduce error. Then, an effective strategy is needed to reduce the likelihood of decoherence and operational errors when the circuit runs on real quantum hardware. Our work performs mapping based on daily calibration data released by IBM in order to avoid using unrealistic qubit errors and to prioritize qubit positioning to reduce the likelihood of quantum errors. IBM calibrates quantum computers regularly to mitigate the impact of quantum errors on quantum circuits [108]. Daily calibrations include single- and two-qubit calibrations. Hourly calibrations deal with readout errors and stability checks.

Figure 4.3 shows T_1, T_2, and Readout errors for all 127 qubits in IBM-Brisbane. A

qubit in a high-energy state ($|1\rangle$) has a natural tendency to decay to a low-energy state ($|0\rangle$). The time that it takes for this transition is called the T_1 coherence time. This type of error is similar to retention errors in classical computers. However, classical computers are only subject to bit-flip errors, whereas quantum computers are also susceptible to phase change errors. The time constant associated with phase change is called the T_2 coherence time. Phase error is the result of the interaction between a qubit and the environment. Over the past decade, coherence time has improved drastically [111].

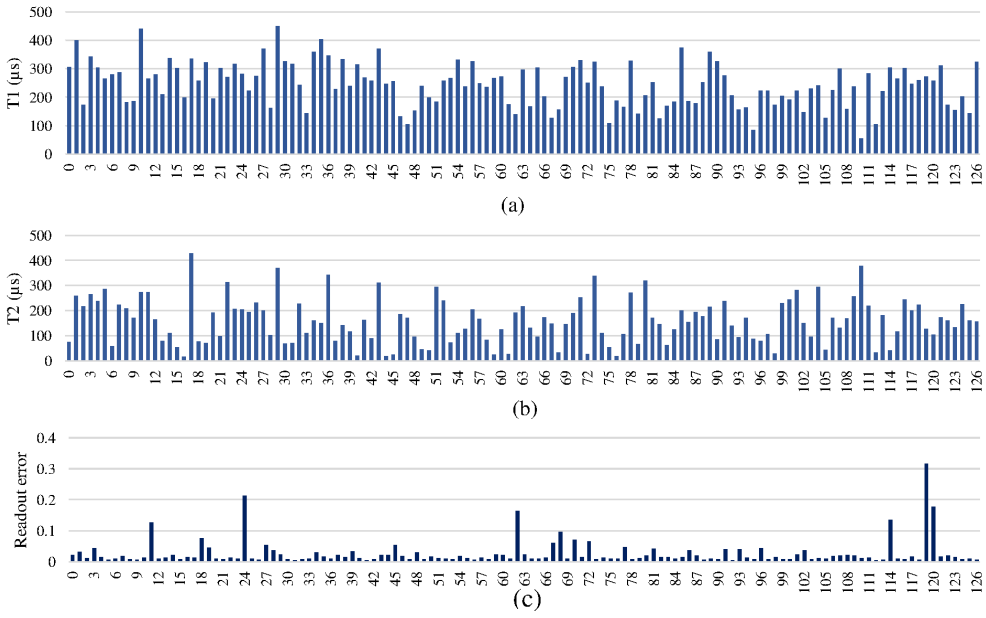


Figure 4.3: Variation of a) T_1, b) T_2, and c) read-out errors over 127 qubits for IBM-Brisbane [108].

From the daily calibration log, we observe that T_1 coherence time varies up to $8x$. The average and standard deviation for T_1 coherence time are $245\mu s$ and $76.5\mu s$, respectively. The T_2 coherence time varies up to $26.2x$ and the average and standard deviation for T_2 coherence time are $158\mu s$ and $88.83\mu s$, respectively.

Readout error on IBM-Brisbane quantum device varies from 0.41×10^{-2} to 0.316. These fluctuations stem from material defects caused by the lithographic process used to manufacture qubits and are expected to be present in future generations of NISQ devices [112].

We use QISKIT 0.45.0 [32] as the baseline compiler for qubit mapping. The compiler takes into account the variability in link errors and assigns logical qubits to physical qubits so that the number of SWAPs is minimized. The main shortcoming of the QISKIT compiler, as well as other qubit allocation techniques [14], [11], [12], is that none of them consider the variability of logical qubits. In other words, these mapping techniques are oblivious to the sensitivity of the output of a quantum circuit to input qubits. In the context of a QNN, not all input qubits are equally important. The sensitivity of the output of the QNN varies from one logical qubit to the other.

To be able to guide the mapping procedure based on the importance of logical qubits, we need a mechanism to calculate the sensitivity of the output to individual qubits. Formally, the sensitivity of the output of a QNN to input qubit q_i can be defined as $\frac{\partial C}{\partial \theta_i}$ where C is the cost function of the QNN and θ_i is the angle of rotation for q_i. We can use the chain rule to compute the derivative of the cost function with regard to input parameters. Let's assume $\{C \rightarrow g_1^p \rightarrow \cdots \rightarrow g_n^p\}$ is the path that emerges from output towards q_i and g_i^p represents a quantum gate. Since there may be more than one such path, we use superscript p to distinguish different paths. The sensitivity is given by:

$$\frac{\partial c}{\partial \theta_i} = \sum_{p=1}^{N_p} \frac{\partial c}{\partial g_1^p} \frac{\partial g_1^p}{\partial g_2^p} \cdots \frac{\partial g_n^p}{\partial \theta} \tag{4.6}$$

Instead of going through the burden of computing derivatives for individual gates, we use a profiling approach to extract the sensitivity of logical qubits. For qubit mapping, the

Algorithm 6 Logical to Physical Qubit Mapping

Steps:

1. Use IBM QISKIT compiler to generate initial mapping.
2. Use profiling to find sensitivity of individual qubits.
   - 2.1 For each input qubit, increase the angle of the corresponding rotation gates by Δ.
   - 2.2 if QNN predicts a different class, set sensitivity to $\frac{1}{n\Delta}$ (n is the number of times the angle is increased); otherwise, go to step 2.1.
3. Sort logical qubits in descending order of sensitivity.
4. Sort physical qubits in descending order of reliability extracted from IBM calibration data.
5. Map the sorted logical qubits to the sorted physical qubits one by one.

absolute value of the sensitivities is not important. We only need a mechanism that sorts logical qubits based on their relative sensitivity. Once a QNN is trained, we evaluate the sensitivity of output to individual inputs by increasing the angle of input rotation gates by a small value: Δ. It is important to note that the angles of all rotation gates used for the encoding of the qubit are increased by Δ. If the prediction made by the QNN changes, then we record $1/\Delta$ as sensitivity for input qubit q_i; otherwise, we increase the input angle by another Δ. We continue this procedure (each time, the angle is increased by Δ) until the prediction made by the QNN before and after the angle increase changes. Intuitively, Δ is a measure of the vulnerability of qubits to noise. An input qubit that requires a large increase in its angle to flip the prediction has lower sensitivity. On the contrary, if the QNN predicts a different class by a small increase in an input angle, then the sensitivity of the qubit is high.

Once we compute the sensitivity of all input qubits, we are ready to map logical qubits to physical qubits. Since the Qiskit compiler optimizes the number of SWAP gates, we use the same physical qubits selected by the compiler for the QNN. We only change the

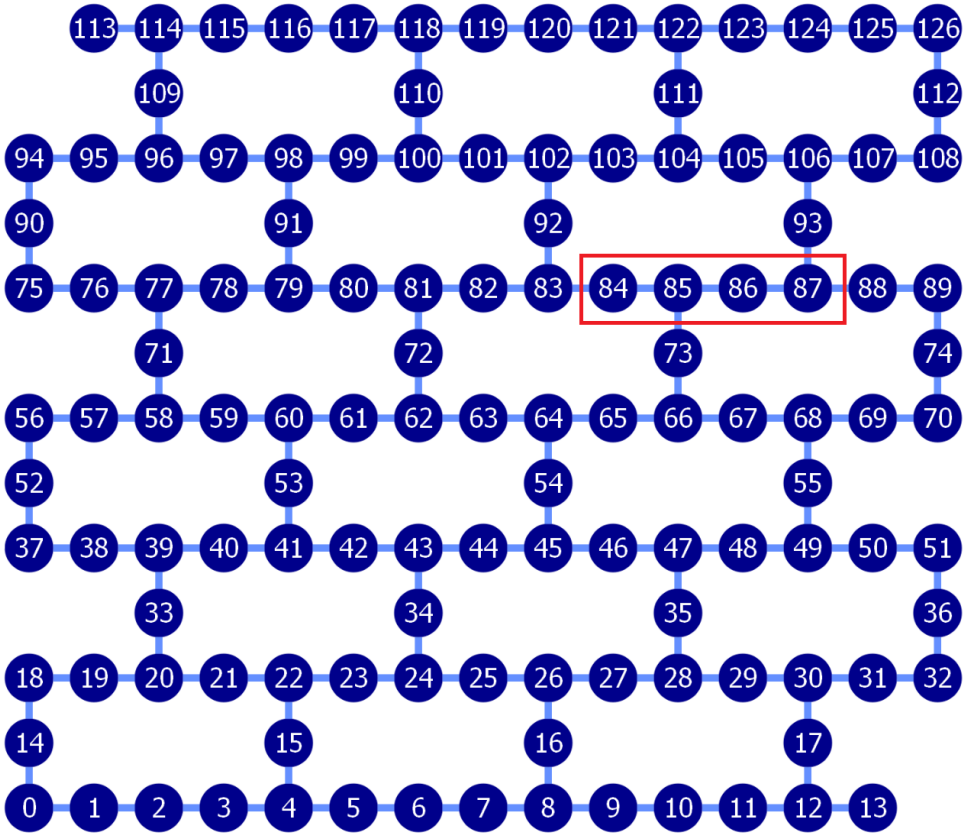


Figure 4.4: IBMQ-Brisbane Layout [108].

mapping of logical qubits to the selected physical qubits. For example, Figure 4.4 shows the topology of qubits in IBM-Brisbane. The four qubits contained in a red box are selected by the compiler for implementation of a QNN with 4 qubits. We use the same qubits but change the logical to physical mapping based on the sensitivities of the logical qubits. The reliability of the physical qubits is determined based on calibration data released by IBM and includes T_1 coherence, T_2 coherence, and measurement errors. A more sensitive logical qubit is mapped to a more reliable physical qubit. Algorithm 6 shows the steps in sensitivity-aware qubit mapping.

4.2 Approximate QRAM

In this section, we first explain the architecture of an approximate QRAM and how it is trained. Then, we elaborate on how we exploit pruning to simplify the circuit of the QRAM.

4.2.1 Architecture of an Approximate QRAM

The backbone of an approximate QRAM is a PQC-based QNN. The QRAM has n address lines and is able to accommodate up to 2^n datapoints. For a given datapoint, a unique address is provided, and an address-data pair is formed. Figure 4.5 depicts the architecture of a multi-layer QNN. The first stage of the QNN is encoding, where classical data are mapped to quantum states. Mapping classical data into quantum states is a non-trivial task, especially for machine learning applications that involve large datasets. From a theoretical perspective, there is no fundamental obstacle to uploading vast amounts of data into qubits. However, on NISQ devices, achieving this while keeping quantum circuit noise under control remains challenging.

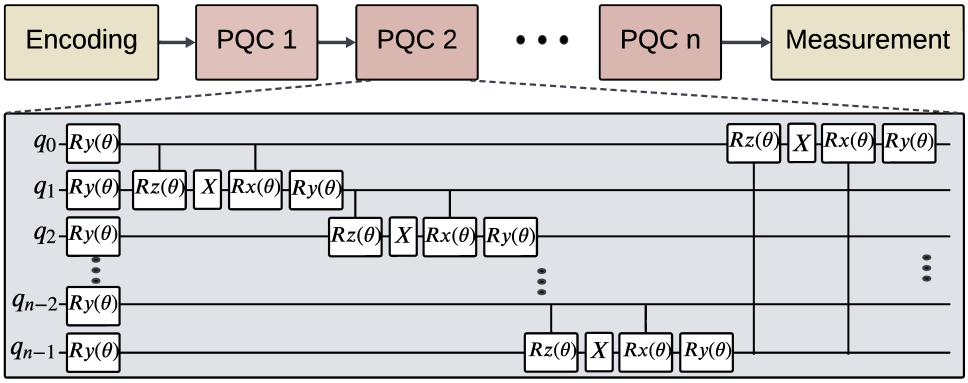


Figure 4.5: Architecture of a multi-layer QNN.

A natural approach to QNN design involves circuits that exploit entanglement among quantum states, allowing the classifier to access a broader region of the Hilbert space. Entanglement in quantum circuits is analogous to correlation in classical circuits. For instance, a neuron connected to two inputs in a classical NN exploits the correlation between them, improving classification accuracy. If no correlation exists, the weight of the corresponding input connections is negligible, allowing their removal without degrading accuracy. Similarly, in the quantum domain, if two features are uncorrelated, removing their entanglement has no impact on the circuit's accuracy.

Figure 4.5 illustrates the PQC used in this work. A QNN can consist of multiple PQCs. Determining the optimal number of PQCs lacks a strict mathematical formula and often relies on a trial-and-error approach. Through simulation, the accuracy of classifiers with different PQC configurations is evaluated, and the structure yielding the highest accuracy is selected. The outputs of the last PQC are measured to convert quantum states into classical data.

4.2.2 Training of Approximate QRAM

We use the backpropagation algorithm to train the QRAM. The training procedure is illustrated in Figure 4.6 (Step 1) and is conducted using a given training dataset. The dataset consists of images, each corresponding to an address. The QRAM receives an input address and generates the associated image. The parameters of PQCs in the QRAM are optimized so that the output of the QRAM closely matches the expected images.

For training, a cost function is required to quantify the error in classification. We use

the Mean Squared Error (MSE) as the cost function, defined as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \text{QRAM}(X_i)\right)^2 \tag{4.7}$$

where N is the size of the training set, X_i is the input, Y_i is the correct label associated with X_i, and $\text{QRAM}(X_i)$ is the prediction of the QRAM for X_i.

We use the Adam gradient descent algorithm for training. Gradient descent works by computing the derivative of the cost function with respect to the network parameters. In the quantum domain, the parameter-shift rule [74] is a widely used method for calculating quantum gradients. This technique computes the gradient by evaluating the cost function at two distinct data points, which do not need to be close to each other. This feature enhances the method's resilience to noise [74]. The backpropagation algorithm, leveraging the parameter-shift gradient method, is implemented in PennyLane [75], which we use to design and train the QRAMs in this work.

4.2.3 Simplifying Approximate QRAM

Contemporary NISQ devices suffer from quantum noise, which impacts the fidelity of quantum circuits. One of the factors that causes quantum noise is coherence error. If the time that it takes to compute the output of a circuit exceeds the coherence time, then the output of the circuit is skewed with noise, making it erroneous. Thus, the depth of a circuit directly impacts the fidelity of the output. A deeper circuit requires more time to generate output, which results in more quantum noise. As a result, optimizing a deep circuit and replacing it with a shallow one can reduce computation time, thereby mitigating noise levels. The other factor that impacts the noise of circuits is the number of quantum gates.

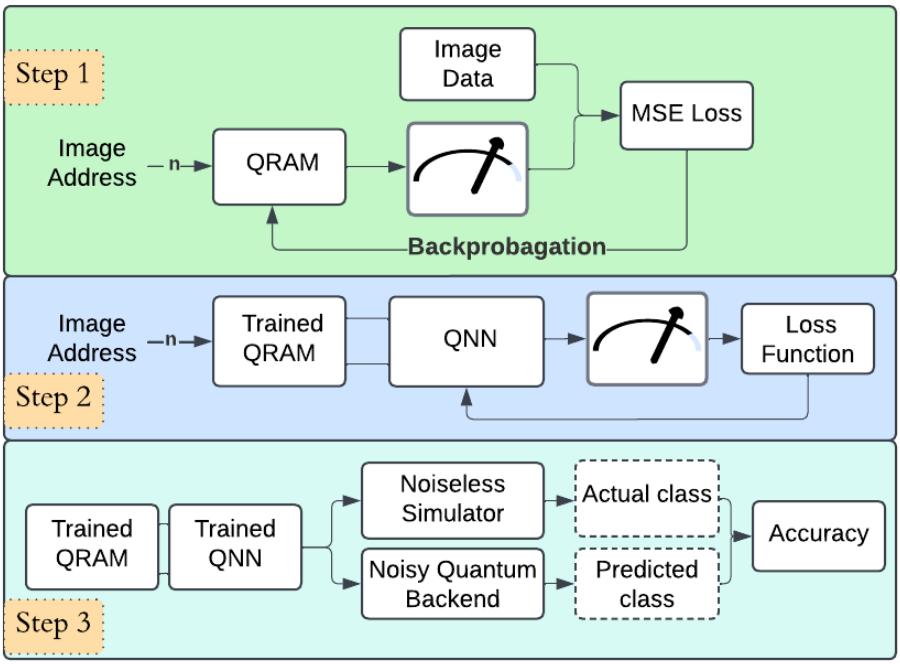


Figure 4.6: Training and inference of QRAM for a given dataset.

Each gate adds noise to the qubits it operates on. Thus, simplifying a circuit by reducing the number of quantum gates increases the success of quantum circuits on NISQ devices.

An effective method to reduce noise in approximate QRAMs is quantum pruning. Removing gates with a small angle of rotation can reduce the depth of circuits as well as the number of quantum gates. Our evaluations of PQCs in QRAMs indicate that the angle of rotation demonstrates predictable behavior during training. If an angle stays far from zero in consecutive training epochs, then it is likely to stay far from zero in subsequent training epochs. Similarly, if an angle stays close to zero for a few iterations, then it remains close to zero in the following iterations. This property helps us to exploit pruning and simplify approximate QRAMs without compromising the accuracy of the circuit.

Simplifying the architecture of approximate QRAM involves three parts. First, the approximate QRAM is trained following the procedure explained in section 4.2.2. Then, those rotation gates with angles less than a predetermined threshold are removed. Finally, the remaining gates are retrained to compensate for potential accuracy loss. This process is repeated iteratively to minimize the number of gates while maintaining the accuracy of the baseline circuit.

During retraining, it is preferable to initialize the surviving angles with the values learned in the previous training phase rather than resetting them to random values. Neural networks, including QNNs, contain fragile co-adapted features [109]. While gradient descent can effectively find a suitable solution during the initial training phase, reinitializing the parameters from scratch often hampers the network's ability to regain its original performance.

We utilized the Algorithm 5 to simplify approximate QRAM. The training of PQCs within the QRAM is performed over N_e epochs (Line 6). The parameters of the PQCs are computed using the gradient descent method [74]. At the end of each epoch, the angle of each rotation gate in the PQC is compared to a predetermined threshold value (Line 10). Gates with angles smaller than the threshold are removed from the circuit. Since removing ration gates from PQCs may reduce the accuracy of the circuit, the pruned networks are trained in the subsequent iteration. The other advantage of removing rotation gates is training time. As the number of gates reduces, the gradient descent method requires less time to optimize the network parameters, which accelerates the training process.

Pruning gates with small rotation angles is an iterative procedure. One possible approach involves training the QRAM for n epochs and performing pruning only once at the end. While this method minimizes the overhead of pruning, it fails to fully exploit the potential of the pruning strategy. Such a one-time pruning approach may produce a

smaller network compared to the original, but could still include many redundant gates. Conversely, performing pruning at the end of each epoch incrementally removes unnecessary gates, enabling the discovery of a QRAM with fewer gates. This gradual pruning process enhances the likelihood of deploying a compact and efficient QRAM on an NISQ device.

Chapter 5

Evaluations

This chapter presents a comprehensive evaluation of the approximate quantum arithmetic circuits and quantum machine-learning architectures developed in this work. Motivated by the limitations of current NISQ-era hardware—where depth, gate count, and susceptibility to noise remain major obstacles—this chapter investigates how approximate computation can significantly reduce resource requirements while maintaining high functional accuracy. The focus spans four major components of the proposed framework: approximate adders, approximate multipliers, approximate division circuits, and the QSQR, followed by evaluations of the proposed NR-QNN architecture and the approximate QRAM module. Together, these components demonstrate a unified approach for building noise-resilient and resource-efficient quantum circuits suitable for real quantum processors.

We begin with a detailed comparison of approximate quantum adders, examining their depth, gate count, scalability, and accuracy. This establishes the foundation upon which more complex arithmetic units are constructed. Next, the chapter evaluates the approximate quantum multipliers, analyzing their structure, error characteristics, and suitability

for practical applications. Special emphasis is placed on using these multipliers in real-world tasks such as image processing. Their performance is further validated through experiments conducted on IBM quantum hardware.

The discussion then extends to the approximate quantum division circuits, where we assess their computational efficiency and application potential. Similar to multipliers, these division circuits are tested on real quantum processors to demonstrate their practicality under hardware noise and decoherence. Following this, we evaluate the proposed approximate square root circuit based on the non-restoring algorithm. The QSQR design is analyzed in terms of approximation quality, resource savings, and application domains, and its correctness is further confirmed through implementation on actual NISQ devices.

In evaluating the performance of the proposed approximate arithmetic circuits, this chapter compares each design against the most optimized exact quantum arithmetic circuits available in the literature. Because this work introduces the first family of approximate designs for quantum adders, multipliers, dividers, and square root units, no prior approximate baselines exist for direct comparison. Therefore, to ensure fairness and context, each proposed circuit is benchmarked against state-of-the-art exact implementations that minimize depth, T-count, and qubit usage. This approach highlights the improvements achieved through approximation and demonstrates how significant approximation can be obtained while maintaining acceptable computational accuracy—an essential requirement for practical deployment on NISQ devices.

In addition, the chapter evaluates the NR-QNN. We examine the model's accuracy, convergence behavior, robustness against noise, and suitability for deployment on real quantum systems. Finally, the chapter concludes with an evaluation of the QRAM module, highlighting its ability to store and retrieve quantum data efficiently using approximate and noise-resilient operations.

In this work, we used several IBM Quantum backends to run our quantum circuits on real quantum devices, with IBM–Brisbane being the primary system used for most experiments. These backends are based on superconducting qubit technology, one of the most advanced and widely adopted approaches for constructing practical quantum processors. The selection of each backend was driven by two main factors: (1) the number of available qubits, ensuring that the chosen device could support the full algorithmic requirements—including address qubits, data qubits, and ancillary operations—and (2) the practical availability of the system at the time of execution. Since current quantum hardware is still in an early development stage, device availability and queue times fluctuate frequently, and the operational status of different systems can change over time. Therefore, backends such as IBM–Brisbane were chosen because they offered sufficient qubit resources, stable calibration metrics, and accessible execution windows, making them suitable for reliably evaluating the proposed quantum models on real superconducting hardware.

5.1 Comparison of Approximate Quantum Adders

In section 3.3, we explained the architecture of four approximate quantum adders. Table 5.1 compares the proposed approximate quantum adders with existing exact quantum adders [16], [17], [18]. For each adder, Table 5.1 reports the T-count and depth of the corresponding circuit. We only focus on T-count and depth of circuits, as the noise level in a quantum circuit depends on these two parameters. A quantum circuit with a higher number of T-counts is more likely to generate noisy outputs. Also, a deeper circuit is more susceptible to noise as errors of quantum gates that are chained together accumulate. The T-count in approximate and exact adders is $O(n)$. However, T-count in approximate adders grows at a much slower pace than exact quantum adders. All approximate adders

Table 5.1: Comparison of Quantum Adders

Exact Adder	T-count	Depth
Jayashree et al. [16]	$28n + 7$	$3n + 2$
Muñoz-Coreas et al. [17]	$21n + 14$	$5n + 6$
Chun Lin et al. [18]	$56n$	$6n + 1$
Proposed Circuits		
C-AQANC	$4n$	1
C-AQAFC	$8n$	3
C-AQAHC	$6n + 4$	3
C-AQAOC	$4n + 8$	3

have lower depth than exact adders. Regardless of the size of an approximate adder, the depth of the circuit is $O(1)$. This is the key characteristic of our proposed techniques, which enables the approximate adders to generate meaningful results on NISQ devices. On the contrary, all exact adders [16], [17], [18] exploit a chain of quantum gates to generate carry qubits. As a result, none of them is scalable and cannot be implemented on a real quantum computer.

5.2 Evaluation of Approximate Quantum Multipliers

Table 5.2 compares proposed approximate multipliers (section 3.4.2) with exact quantum multipliers in [16], [18], [17]. We do not consider those quantum multipliers that exploit the quantum Fourier transform for comparison, as they suffer from a high number of T gates. As an example, the T-count for the multipliers in [38] is $O(n^3)$, which makes it impossible to implement it on NISQ devices. The T-count in approximate and exact multipliers in Table 5.2 is $O(n^2)$. However, T-count increases with a much slower pace in approximate multipliers than in exact multipliers. The depth of approximate multipliers changes linearly with the size of the input operands. On the contrary, the order of growth of depth in exact

Table 5.2: Comparison of Quantum Multipliers

Adder	T-count	Depth
Jayashree et al. [16]	$28n^2 + 7n$	$3n^2 + 4n - 2$
Muñoz-Coreas et al. [17]	$21n^2 - 14$	$5n^2 + 2n - 6$
Chun Lin et al. [18]	$56n^2$	$7n^2 + n$
Proposed Circuits		
AQMNC	$4n^2$	$\mathcal{O}(1) + (n-1)$
AQMFC	$8n^2 - 4n$	$\mathcal{O}(1) + 3(n-1)$
AQMHC	$6n^2 + 2n - 4$	$\mathcal{O}(1) + 3(n-1)$
AQMOC	$4n^2 + 8n - 8$	$\mathcal{O}(1) + 3(n-1)$

multipliers [16], [18], [17] is quadratic. This prevents exact multipliers from being deployed into contemporary NISQ devices.

5.2.1 Application of Approximate Quantum Multipliers

The proposed approximate multipliers are applied to an image processing algorithm where two images are multiplied pixel by pixel and combined into a single output image. We selected standard images from [113] for evaluation of the multipliers. Peak signal-to-noise ratio is one of the parameters used to assess the proximity of an approximate image to an exact image [114]. This parameter does not necessarily reflect the quality of an image perceived by humans [114]. A better metric to measure the quality of an approximate image is the Structural Similarity Iindex Metric (SSIM), which works based on quantization of structural similarity of the exact and approximate images [114]. SSIM is based on the principle that human visual perception can extract information from the structure of images. Computation of SSIM is complex and is described in detail in [114]. Table 5.3 shows the SSIM of the multiplication of two images (moon and cameraman [113]) when approximate multipliers are employed. The maximum value for SSIM is one where an exact

Table 5.3: Accuracy for 8×8 Approximate Multipliers

Adder	SSIM
Liu *et al.*[36]	0.49
Akbari *et al.*[35]	0.77
Gupta *et al.*[34]	0.53
Proposed Circuits	
AQMNC	0.48
AQMFC	0.66
AQMHC	0.62
AQMOC	0.59

multiplier is used. In Table 5.3, we also report SSIM for approximate multipliers proposed for classical circuits in [34], [35], [36]. We select multipliers with the highest accuracy from [34], [35],[36]. AQMNC has the lowest accuracy. This is expected as the design of AQMNC considers only the input operands for the computation of the sum and ignores carry qubits. AQMFC recovers 18% of the error in AQMNC by dedicating quantum gates for the generation of carry. AQMHC drops half of the quantum gates used in AQMFC for the carry path and achieves an accuracy close to AQMFC. Even AQMOC with only one carry qubit is more accurate than approximate multipliers based on [34], [36]. The approximate multiplier proposed in [35] has the highest accuracy. However, this multiplier uses back-to-back gates for the generation of carry bits. The depth of the logic for the carry path is accumulated from the LSB all the way to the MSB. Such a circuit is not practical on NISQ devices and results in a significant noise level.

5.2.2 Experiment on a Real Quantum Computer

IBM has developed multiple quantum computers and a quantum simulator. We run our experiments on a 65-qubit quantum computer called IBMQ-Brooklyn [8]. We use QISKIT

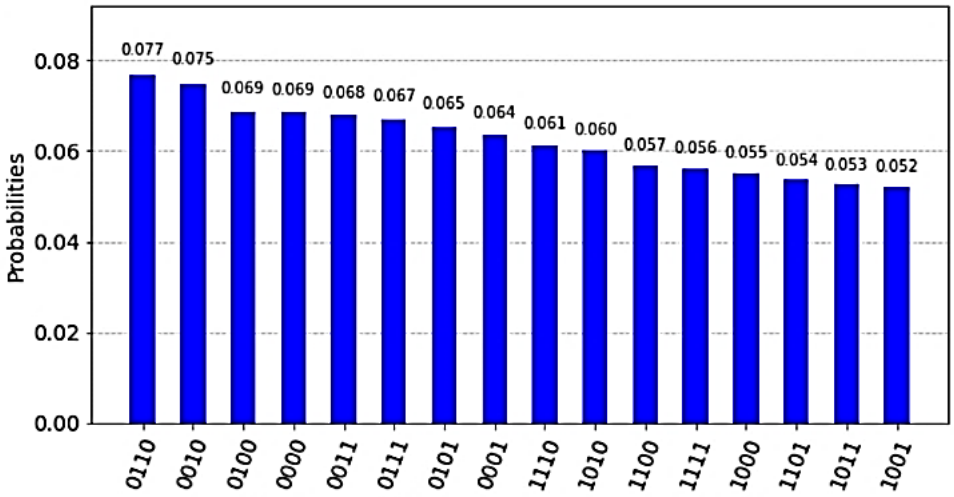


Figure 5.1: 4-qubit exact adder [26] on IBMQ-Brooklyn [8].

[32] to model our proposed quantum circuits.

Figure 5.1 shows the addition of $|1111\rangle$ and $|1111\rangle$ using the circuit proposed by [26] (Exact adder explained in section 3.1). It is clear that the 4-qubit adder is not practical on NISQ computers, and it produces wrong results due to excessive noise. The desired output state is $|0000\rangle$, and it is expected to have the highest probability when the output is measured. However, $|0000\rangle$ is the fourth most probable output. This is mainly due to the carry path in the exact adder, which increases the depth of the circuit and causes noisy outputs. As a result, when we measure the quantum state of output qubits, the outputs do not collapse to the correct state. Thus, building a quantum multiplier using such an adder is not feasible on contemporary quantum computers. The solution for this problem is to use approximate adders to shrink the critical path. By simplifying the adder circuit, it is feasible to overcome the limitation of contemporary NISQ devices. Figure 5.2 shows the multiplication of two 4-qubit numbers on IBMQ-Brooklyn [8] using the proposed approximate multipliers. All multipliers compute the product of $|1111\rangle$ and $|1111\rangle$. We set

all input qubits to $|1\rangle$ as it creates maximum delay in the circuits. Since the multipliers exploit approximation, the results generated by the multipliers are not the same as an exact multiplier. For example, AQMNC, which uses approximate adders with no carry, generates $|01010101\rangle$ while the output of an exact multiplier would be $|11100001\rangle$. Thus, the values with the highest probabilities in Figure 5.2 are approximations of an output generated by an exact multiplier. For all approximate quantum multipliers implemented on IBMQ-Brooklyn [8], the probability distribution of outputs generates the correct results. As an example, for AQMOC, the value with the highest probability is the same as the output generated by the corresponding classical approximate multiplier.

We define fidelity of results as the difference between the highest probability observed in the output and the "runner up", i.e., the probability with the second-highest value. AQMNC has the highest fidelity among all multipliers. This is due to the simplified circuit in C-AQANC, which computes only the sum and skips carry qubits. On the contrary, AQMFC falls behind the other circuits in terms of fidelity. This is expected as AQMFC is built out of approximate adders that compute carry for all qubits. The fidelity of the other two multipliers is between AQMNC and AQMFC.

5.3 Evaluation of Approximate Quantum Division

In this section, we compare our proposed restoring and non-restoring division circuits with existing designs. In particular, we compare T-count, T-depth, and the number of qubits of the division circuits. We do not focus on the number of ancillas and garbage as in IBM quantum computers, which can be reset and reused for other qubits.

The four approximate adders do not use any T-gates. As a result, the T-count and the T-depth of the approximate adders are zero. We use the exact adder proposed by [26]

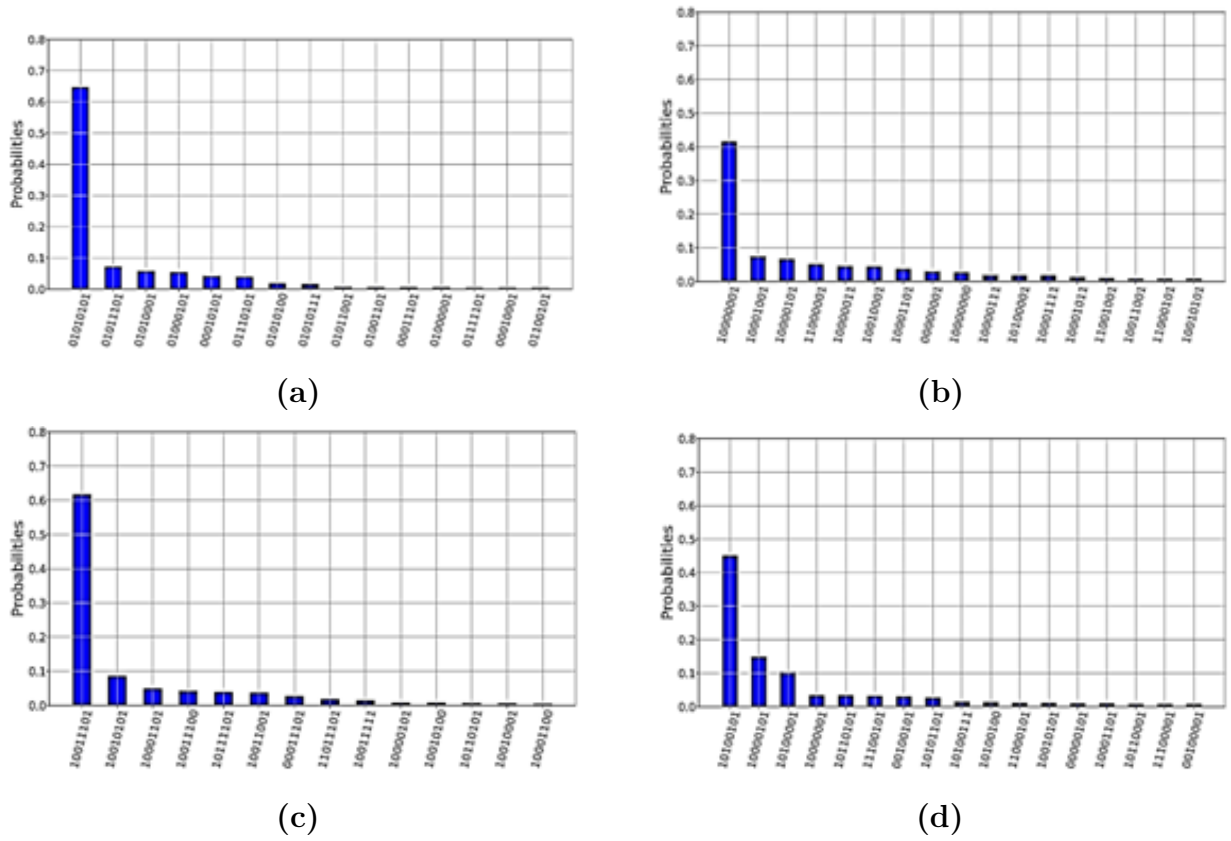


Figure 5.2: Results of 4-qubit approximate multipliers on a real quantum computer: (a) AQMNC, (b) AQMFC, (c) AQMHC, (d) AQMOC

in this work. The T-count and the T-depth of an n-qubit exact adder are 4n-4 and 2, respectively.

The building blocks of restoring and non-restoring division circuits are QSUB, QCA, and QCAS. Table 5.4 reports the cost of these components built out of approximate and exact adders.

Table 5.4: Cost of n-qubit QSUB, QCA, and QCAS.

Circuit	**QSUB**		**QCA**		**QCAS**	
	T-count	**T-depth**	**T-count**	**T-depth**	**T-count**	**T-depth**
Enhanced Exact	$4n-4$	2	$8n-4$	2	$4n-4$	2
AQMNC	0	0	$4n$	2	0	0
AQMFC	0	0	$8n-4$	2	0	0
AQMHC	0	0	$6n$	2	0	0
AQMOC	0	0	$4n+4$	2	0	0

Table 5.5: Comparison of Restoring Division Circuits

Restoring Division Circuits	**T-count**	**T-depth**	**# of Qubits**
Khosropour *et al.* [101]	$400n^2$	$130n$	$4n$
Thapliyal *et al.* [115]	$35n^2-28n$	$23n$	$3n$
Dibbo *et al.* [116]	$9n^3$	NA	$0.5n^3+4n$
Proposed Circuits			
Enhanced Exact	$12n^2-8n$	$2n$	$2n+1$
AQDNC	$4n^2$	$2n$	$2n+1$
AQDFC	$8n^2-4n$	$2n$	$2n+1$
AQDHC	$6n^2$	$2n$	$2n+1$
AQDOC	$4n^2+4n$	$2n$	$2n+1$

5.3.1 Cost Comparison for Restoring Division

a) T-count

The restoring quantum division circuit exploits a block per iteration of the algorithm 2 (Figure 3.7(a)). Each block consists of an n-qubit QSUB and an n-qubit QCA (Figure 3.7(b)). Thus, the T-count per block is $4n$, $8n-4$, $6n$, $4n+4$, and $12n-8$ for AQDNC, AQDFC, AQDHC, AQDOC, and Exact adder, respectively. Since each block is repeated n times, the T-count for the circuit is equal to n multiplied by the T-count per block. The second column in Table 5.5 shows the T-count of the proposed restoring division circuits.

b) T-depth

The T-depth of a circuit is computed by considering the number of T-gates seen by each input qubit, individually. The quantum register with the maximum number of T-gates determines the T-depth of the circuit. In the restoring quantum division circuit (Figure 3.7(a)), registers $|B\rangle$ and $|R\rangle$ see the maximum number of blocks. By considering how each register is applied to a block, we can determine that register $|B\rangle$ encounters the maximum number of T gates. Thus, we only consider the T-depth of the quantum register $|B\rangle$ for the T-depth calculation. By adding the T-depth of a QSUB and a QCA from Table 5.4, the T-depth of the restoring circuit can be computed as 2n for all restoring division circuits (the third column in Table 5.5).

c) Number of Qubits

Figure 3.7(a) has three input registers: $|Q\rangle$, $|B\rangle$, and $|R\rangle$. The sizes of $|Q\rangle$, $|B\rangle$, and $|R\rangle$ are n-, n-, and 1-qubit respectively. Thus, the total number of qubits in the restoring circuit is $2n + 1$ (the fourth column in Table 5.5). Table 5.5 compares the cost of the proposed restoring circuits with existing designs [101], [115], [116]. Our proposed approximate circuits are superior to all existing designs in terms of T-count, T-depth, and the number of qubits. Approximate blocks used in the design of quantum QSUB and QCA reduce the number of quantum gates as well as quantum depth, and thus, T-count and T-depth are lower in our proposed designs.

5.3.2 Cost Comparison for Non-Restoring Division

a) T-count

We divide the T-count computation for the non-restoring division circuit (Figure 3.8) into three parts:

- Part 1: This part corresponds to the part of the algorithm 3 which appears before the For loop. It consists of an n-qubit QSUB (section 3.5.3).
- Part 2: The for loop in algorithm 3 is implemented by (n-1) QCAS units (3.5.3).
- Part 3: The last part corresponds to the section of the algorithm 3 that appears after the For loop. This section is implemented by an (n-1) qubit QCA (section 3.5.3).

The total T-count is computed by summing the T-count of the three parts. As an example, if the non-restoring division circuit is implemented using AQDNC, then based on Table 5.4, the breakdown of T-count for the three parts is as follows:

- Part 1: 0
- Part 2: (n-1)×0=0
- Part 3: 4×(n-1) = 4n-4

The second column of Table 5.6 reports the total T-count of the proposed non-restoring division circuits.

Table 5.6: Cost of Non-Restoring Division Circuits

Restoring Division Circuits	**T-count**	**T-depth**	**# of Qubits**
Jamal and H. Babu [102]	$28n^2$	NA	$2n^2 + 5n - 1$
Thapliyal *et al.* [115]	$14n^2 + 7n - 21$	$10n + 13$	$3n - 1$
Dibbo *et al.* x11	$9n^3$	NA	$0.5n^3 + 4n$
Proposed Circuits			
Enhanced Exact	$4n^2 + 4n - 12$	$2n + 2$	$2n + 1$
AQDNC	$4n - 4$	2	$2n + 1$
AQDFC	$8n - 12$	2	$2n + 1$
AQDHC	$6n - 6$	2	$2n + 1$
AQDOC	$4n$	2	$2n + 1$

b) T-Depth

Among all inputs of the non-restoring circuit, quantum register $|B\rangle$ encounters the highest number of T-gates. Thus, the total T-depth is computed by counting the number of T-gates on $|B\rangle$ across the three zones of the circuit. The third column of Table 5.6 reports the T-depth of the proposed non-restoring division circuits.

c) Number of Qubits

The non-restoring circuit has three input registers: $|B\rangle$, $|R\rangle$, and $|Q\rangle$ with n-, 1-, and n-qubit, respectively. Thus, the total number of qubits in the circuits is $2n + 1$ (the fourth column in Table 5.6). Table 5.6 compares the cost of our proposed non-restoring circuit with the cost of existing designs [102], [115], [116].

5.3.3 Applications of Approximate Quantum Division

In this section, we assess the non-restoring approximate quantum division circuit based on an exact adder and AQDHC using two image processing applications: change detection

Table 5.7: SSIM for Image Processing Applications

Application	**SSIM**
Change detection	0.95
Foreground extraction	0.90

and foreground extraction.

In change detection, motion across a set of frames is detected by dividing two consecutive frames pixel-by-pixel. If there is no movement between the two frames, then dividing the two frames generates fixed values of one. However, if an object moves, then dividing back-to-back frames generates potentially variable values for those regions in which the intensity spatially changes. Table 5.7 shows the SSIM [117] for two standard images specifically tailored for change detection [118]. SSIM achieves its maximum value of one when exact and approximate images are indistinguishable.

The second application is foreground extraction. This application is used to extract foreground in an image with a visually disruptive background. By dividing an image by its background, the foreground object can be better viewed. Table 5.7 shows SSIM for two images designed for foreground extraction [118]. The evaluation of the approximate division circuit suggests that our proposed division circuit offers acceptable levels of accuracy in applications that are resilient to inexactness.

5.3.4 Experiments on a Real Quantum Computer

We use IBMQ-Brisbane [8] to run the non-restoring approximate quantum division circuit. Figure 5.3 shows the layout of the IBMQ-Brisbane, and Table 5.8 lists characteristics of the computer. We run the circuit for 10,000 shots and set the size of the circuit (n) to 3-qubit. We use QISKIT [32] to model the quantum circuits. Figure 5.4(a) shows the

Table 5.8: IBMQ-Brisbane Characteristics

Details	**Value**
# of Qubits	127
CLOPS	2.7K
EPLG	1.9×10^{-2}
Processor type	Eagle r3
Version	1.1.12
Features	OpenQASM 3
Basis gates	ECR, ID, RZ, SX

Qubit Metric	**Min**	**Max**	**Median**
T_1 (μs)	11.47	399.93	220.03
T_2 (μs)	16.25	389.43	132.19
Frequency (GHz)	4.61	5.118	4.906
Readout error	4.10×10^{-3}	3.72×10^{-1}	1.23×10^{-2}
Pauli-X error	9.47×10^{-5}	4.16×10^{-2}	2.33×10^{-4}
ECR error	2.85×10^{-3}	1.00×10^{0}	7.52×10^{-3}
Gate time (ns)	660	1100	660

probability distribution of the output in the exact quantum division circuit when a=3 and b =2. These input values cause maximum delay in the circuit. The circuit does not generate the correct result due to quantum noise. The desired state is $|00101\rangle$, and it is expected that it has the highest probability in the output. However, it does not exist among the top five most frequent outputs. This is mainly due to the complexity of the exact circuit, which makes it impractical for contemporary quantum computers. Figure 5.4(b) shows the probability of outputs when the same inputs are applied to our proposed approximate circuit. The combination of a dynamic circuit and approximation simplifies the circuit and makes it feasible to generate meaningful results on the NISQ device. The approximate circuit generates the expected result ($|01011\rangle$).

One of the advantages of the dynamic circuit is reducing the number of SWAP gates. In the IBMQ-Brisbane, similar to other NISQ devices, there is a limited number of links

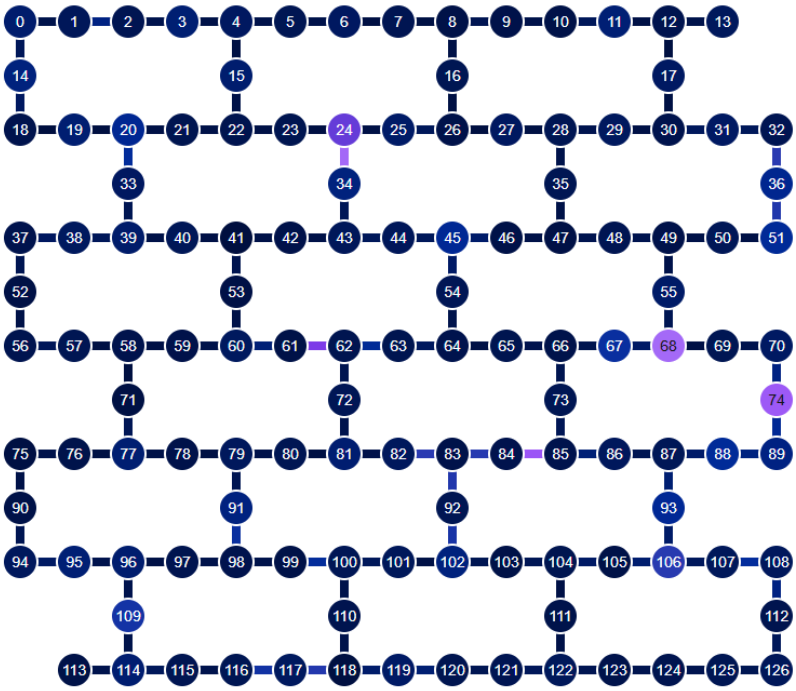


Figure 5.3: IBMQ-Brisbane layout [8].

between qubits, and quite often, a node is connected to its neighborhood qubits. These links enable entanglement between qubits through multi-qubit gates such as CNOT. If two qubits derive inputs of a CNOT gate but there is no direct link between the two, then a series of SWAP gates is needed to move the state of the two qubits towards each other so that they become adjacent. These SWAP gates prolong the duration of circuit execution and increase the susceptibility of quantum circuits to coherence errors. Dynamic circuits enable the reusability of qubits, which in turn reduces the number of SWAP gates needed for the realization of entanglement in quantum circuits. In addition, approximate computing reduces the number of SWAP gates as it decreases the number of quantum gates, which in turn reduces the distance between qubits. Table 5.9 shows the number of SWAP gates in the baseline and the proposed quantum division circuits. On average, the

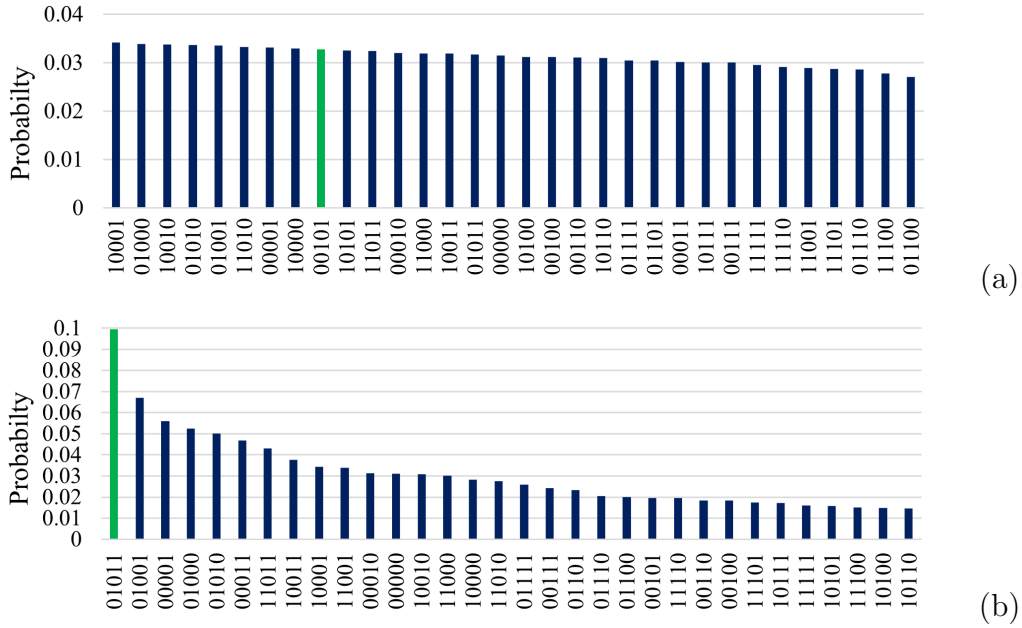


Figure 5.4: Outputs of quantum division circuits executed on a real quantum computer: (a) Exact (expected result= '00101', $Q = 001$, $R = 01$), (b) dynamic circuit+approximate units (expected result= '01011', $Q = 011$, $R = 01$).

Table 5.9: Number of SWAP and CNOT Gates in Quantum Division Circuits

n	**SWAP**		**CNOT**		**Total CNOT**	
	Baseline	**Proposed**	**Baseline**	**Proposed**	**Baseline**	**Proposed**
4	446	203	303	132	1641	741
6	1304	537	729	263	4641	1874
8	2326	1060	1323	426	8301	3606
10	4241	1762	2085	621	14808	5907
12	6189	2719	3015	848	21582	9005
14	7228	3133	3525	995	25209	10394
16	8670	3758	4203	1174	30213	12448
18	10112	4384	4881	1353	35217	14505
20	11555	5010	5559	1532	40224	16562

combination of a dynamic circuit and approximation reduces the number of SWAP gates by 57%.

The other factor that reduces the cost of the proposed quantum division circuit is the number of CNOT gates. Approximate arithmetic circuits such as QCA and QCAS require less number of CNOT gates than their exact counterparts. On average, the proposed approximate division circuit reduces the number of CNOT gates by 69% when the input size varies from four to 20 (4th and 5th columns in Table 5.9). Each SWAP gate is synthesized into three CNOT gates. If we take into account the SWAP gates, then our proposed circuit reduces the total number of CNOT gates by 58% on average when the input size varies from four to 20 (6th and 7th columns in Table 5.9). The combination of a dynamic circuit and approximation simplifies the exact circuit and enables the proposed quantum division circuit to run successfully on the NISQ device.

5.4 Evaluation of Approximate Square Root

The next arithmetic algorithm we optimized is the quantum square root algorithm. As we discussed in section 3.6, we first optimized the quantum circuit for the non-restoring square root algorithm. Then we exploit approximate computing to reduce the cost and complexity of the quantum circuit to propose a practical implementation of the circuit on real hardware. In this section, we compare the T-count and T-depth of the optimized square-root circuit and the approximate circuit. Then, we evaluate the proposed approximate quantum circuit using an application in the area of image processing.

5.4.1 Cost Analysis of Quantum Square Root Circuit

In this section, we compare our proposed circuits with other SQR circuits reported in the literature [51], [44], [43], [47]. In particular, we focus on T-count, T-depth, and the number

Table 5.10: T-Count and T-Depth for QCAS and QCA Blocks

Quantum Circuit	**T-Count**	**T-Depth**
Exact QCAS/QCA	$4n/8n$	2/2
Approx. QCAS/QCA	$0/4n$	0/2

of qubits. We do not consider the number of ancillas for comparison, as ancillas can be reused in NISQ devices by resetting their quantum states. Table 5.10 shows T-count and T-depth for the exact and approximate QCAS and QCA. The exact model is based on the design proposed in [26].

5.4.2 Cost Analysis of Enhanced Exact QSQR

T-Count In the enhanced exact QSQR circuit (Figure 3.13), the size of QCAS [26] in step j is $(j+2)$. Thus, the T-count for QCAS blocks is: $\sum_{j=2}^{n/2} 4(j+2) = \frac{n^2}{2} + 5n - 12$. It is important to note that for the first iteration, there is no QCAS block ($k = 1$ in Figure 3.13). There is only one QCA [26] block with $(\frac{n}{2}+2)$ qubits at the end. Therefore, T-count in the enhanced exact QSQR circuit is: $(\frac{n^2}{2} + 5n - 12) + 8(\frac{n}{2} + 2) = \frac{n^2}{2} + 9n + 4$.

T-Depth For the enhanced exact QSQR circuit, the T-depth corresponding to QCAS blocks is: $\frac{n}{2} - 1$. By adding the T-depth of the last QCA block, We can compute the T-depth of the circuit as: $(\frac{n}{2} - 1) + 2 = \frac{n}{2} + 1$.

Number of Qubits The circuit in Figure 3.13 has three inputs: $|A\rangle$, $|B\rangle$, and $|\text{Ctrl}\rangle$. The size of $|A\rangle$ and $|B\rangle$ are n and $\frac{n}{2} + 2$ qubits, respectively. Thus, the total number of qubits in the exact circuit, including the $|\text{Ctrl}\rangle$ qubit, is $\frac{3n}{2} + 3$.

5.4.3 Cost Analysis of Approximate QSQR

T-Count In the approximate QSQR circuit, the first QCAS is exact, whereas the rest are approximate. The number of qubits in the first QCAS is 4; thus, its T-count is 4×4=16. The T-count for the approximate QCAS is zero. The size of the approximate QCA is $\frac{n}{2} + 2$ qubits. Thus, the total T-count of the approximate circuit is: $16 + (\sum_{j=3}^{n/2} 0) + 4(\frac{n}{2} + 2) = 2n + 24$.

T-Depth T-depths for the exact QCAS, approximate QCAS, and approximate QCA are 1, 0, and 1, respectively. Thus, the total T-depth for the approximate circuit is: $1 + 0 + 1 = 2$.

Number of Qubits The number of qubits in the exact and approximate circuits is the same: $\frac{3n}{2} + 3$.

Table 5.11 compares our proposed QSQR circuits with five other designs [51], [44],[43], [47]. The T-count in all other designs is in the order of $O(n^2)$, whereas the T-count in our proposed approximate circuit changes linearly with the size of the input. We compare the T-count of our proposed circuits with those presented by Sultana *et al.* [43], Bhaskar *et al.* [44], Lakshmi *et al.* [47], and Coreas *et al.* [51] for values of n equal to 4, 8, 16, 32, 64, and 128. The enhanced exact QSQR circuit reduces T-count ranging from 71.42% to 91.11%, 99.31% to 99.84%, 42.85% to 87.9%, and 57.14% to 83.2% compared with the designs by Sultana *et al.* [43], Bhaskar *et al.* [44], Lakshmi *et al.* [47], and Coreas *et al.* [51], respectively. The approximate QSQR circuit reduces T-count ranging from 80.95% to 99.48%, 99.54% to 99.99%, 61.9% to 99.3%, and 71.42% to 99% compared to the designs by Sultana *et al.* [43], Bhaskar *et al.* [44], Lakshmi *et al.* [47], and Coreas *et al.* [51], respectively.

Table 5.11: Comparison of the Cost of QSQR Circuits

Quantum Circuit	**T-Count**	**T-Depth**	**Number of Qubits**
Sultana *et al.* [43]	$7n^2 + 14n$	$3n + 8$	$\frac{n^2}{4} + 6n - 2$
Bhaskar *et al.* [44]	$420n^2 + 168n - 364$	NA	$\approx 42n + 10$
Lakshmi *et al.* [47]	$\frac{21n^2}{4} + \frac{7n}{2} - 14$	NA	$\approx \frac{n^2}{2} + 3n + 4$
Coreas *et al.* [51]	$\frac{7n^2}{2} + 21n - 28$	$5n + 3$	$2n + 1$
Proposed exact QSQR	$\frac{n^2}{2} + 9n + 4$	$\frac{n}{2} + 1$	$\frac{3n}{2} + 3$
Proposed approximate QSQR	$2n + 24$	2	$\frac{3n}{2} + 3$

The T-depth in the proposed approximate circuit is constant. However, the other designs offer a T-depth of $O(n)$. The enhanced exact QSQR circuit reduces T-depth ranging from 83.5% to 85% and 86.95% to 89.78% compared to the designs by Sultana *et al.*[43] and Coreas et al. [51], respectively. The approximate QSQR circuit reduces T-depth ranging from 90% to 99% and 91.3% to 9.38% compared to the designs by Sultana *et al.* [43] and Coreas *et al.* [51], respectively.

In terms of the number of qubits, some designs [51], [44],[43], [47] offer circuits with qubits of order $O(n)$. However, the number of qubits increases at a faster pace than in our proposed designs when the value of n increases. The enhanced exact QSQR circuit reduces the number of qubits ranging from 65.38% to 92.95%, 94.94% to 96.33%, 62.5% to 95.58%, and 0% to 23.25% compared with the designs by Sultana *et al.* [43], Bhaskar *et al.* [44], Lakshmi *et al.* [47], and Coreas *et al.* [51], respectively. The number of qubits in the approximate and enhanced exact circuits is the same.

In summary, our proposed circuits have a significant reduction over previous works in terms of T-count, T-depth, and required qubits.

Table 5.12: Accuracy of Approximate QSQR Circuit for BCI Dataset

	Min	**Max**	**Avg.**
SSIM	0.73	0.91	0.83

5.4.4 Application of Approximate QSQR

In this section, we evaluate the accuracy of the approximate QSQR circuit for two real-world applications: image-contrast enhancement [119] and machine-learning applications.

In image contrast enhancement [119], the intensity level of an image is adjusted such that its visual quality is improved. This involves increasing the difference between the light and dark pixels in an image to make it more visually distinguishable. This type of contrast adjustment can help to bring out details in X-ray images that have low contrast, thereby making the features of the image more visible.

The image contrast program exploits the SQR function. We replace the exact SQR functions with approximate ones and generate approximate outputs. A metric is required to compare the quality of an approximate image and an exact image. One of the metrics used for image comparison is the peak signal-to-noise ratio [117]. However, this metric does not necessarily reflect the quality of images perceived by humans. A better metric to measure the quality of an approximate image is the SSIM [117]. The maximum value for SSIM is one where the exact and approximate images are identical. We evaluate the proposed approximate QSQR circuit using X-ray images from the Breast Cancer Image (BCI) Dataset [120]. Table 5.12 reports the minimum, maximum, and average SSIM values for the BCI images. The average value of SSIM obtained over all images in the dataset is 0.83, which is greater than the SSIM of some classical approximate arithmetic operations in the literature [35]. Thus, the evaluation of the proposed approximate QSQR circuit using the BCI dataset indicates that the circuit is applicable to applications that can tolerate

inexactness.

The second application is taken from the machine learning domain: K-nearest neighbors (KNN). KNN is a type of supervised learning algorithm that predicts the correct class for a test input by calculating the distance between the test input and a set of training points. It determines the similarity between data points using Euclidean distance, which utilizes QSQR operations to calculate distances.

The KNN consists of three steps to compute the similarity between an input and all training points. First, it computes the distance between the test input and each training point. Then, it sorts all the calculated distances. Finally, it selects K points with the smallest distances.

To evaluate KNN, we select a dataset from Social-Network-Ads [121], which is a categorical dataset to determine whether a user purchased a particular product. We utilize our proposed approximate SQR circuit to calculate the Euclidean distances in KNN. Our evaluation reveals that the gap between the accuracy of exact and approximate KNN is less than 1%.

5.4.5 Experiments on Real Quantum Computer

When a quantum gate manipulates its input state, it may inject error into the output state due to quantum noise, because quantum gates are not perfect. For example, a rotation gate may introduce extra erroneous rotation to its output state. For publicly available data from IBM, the error of single-qubit gates is in the order of 10^{-4} to 10^{-3}, whereas the error of two-qubit gates is in the order of 10^{-3} to 10^{-2}. The only two-qubit gate used in the QSQR circuit is CNOT. Given that the error rate of a two-qubit gate is an order of magnitude higher than a single-qubit gate, two-qubit gates usually dominate the overall

error rate. Thus, we compare the number of CNOT gates in our proposed circuits with the design proposed in [51] in Table 5.13. Our proposed enhanced exact QSQR circuit reduces the number of CNOT gates from 38% to 68.5% when the size of the radicand changes from four to 20. The reduction is even higher for the approximate QSQR circuit (76%-90%). Our proposed enhanced QSQR circuit reduces the width of the adders/subtractors and thus eliminates unnecessary CNOT gates. The approximate QSQR circuit removes the carry path in QCAS and QCA. As a result, the CNOT reduction is even higher in the approximate circuit.

Figure 5.3 shows the architecture of the IBM-Brisbane, which has 127 qubits. Each node indicates a qubit, and each edge indicates whether a 2-input quantum gate, such as CNOT, is physically executable. Similar to other NISQ devices [108], IBM-Brisbane offers limited connectivity. If two physical qubits are not connected by a link, then the entanglement of the two qubits can be achieved through a series of SWAP gates. For example, if a CNOT gate is mapped to physical qubits 1 and 5 in Figure 5.3, then 3 SAWP gates are sufficient to move the state of qubit 5 to qubit 2 so that the coupling link between physical qubits 1 and 2 can be used for the execution of the CNOT gate. These SWAP gates are the overhead of quantum circuits and increase the susceptibility of qubits to noise by prolonging the execution time of quantum operations. A circuit with a greater number of SWAP gates suffers more from quantum errors, and thus it is less likely to generate correct results on an NISQ device.

Table 5.14 compares the number of SWAP gates in the enhanced exact QSQR and approximate QSQR with the design proposed by [51] when the size of the radicand (n) increases from four to 20. Our proposed approximate quantum circuit reduces T-count and T-depth. Thus, when mapped to a real quantum computer, it requires a smaller number of SWAP gates for operation. As a result, it is more likely that the proposed circuit runs

Table 5.13: Number of CNOT Gates in QSQR

Size of Radicand (n)	**Coreas *et al.*** [51]	**Enhanced Exact QSQR**	**Approximate QSQR**
4	220	136	52
6	467	248	86
8	804	384	126
10	1256	494	172
12	1856	688	224
14	2576	906	282
16	3334	1048	346
18	4222	1314	416
20	5423	1604	492

Table 5.14: Number of SWAP Gates in QSQR

Size of Radicand (n)	**Coreas *et al.*** [51]	**Enhanced Exact QSQR**	**Approximate QSQR**
4	498	192	98
6	773	414	202
8	1179	688	347
10	1787	842	531
12	2331	1201	614
14	3477	1496	753
16	4651	2078	949
18	6387	2626	1152
20	7529	3139	1458

successfully on an NISQ device.

Figure 5.5 shows the outputs of the exact and approximate circuits on two IBM quantum computers: IBM-Montreal (27-qubit) and IBM-Brisbane (127-qubit) [108] for 10,000 shots. We do not show the remainder because we focus only on the integer part ($|Q\rangle$) of the

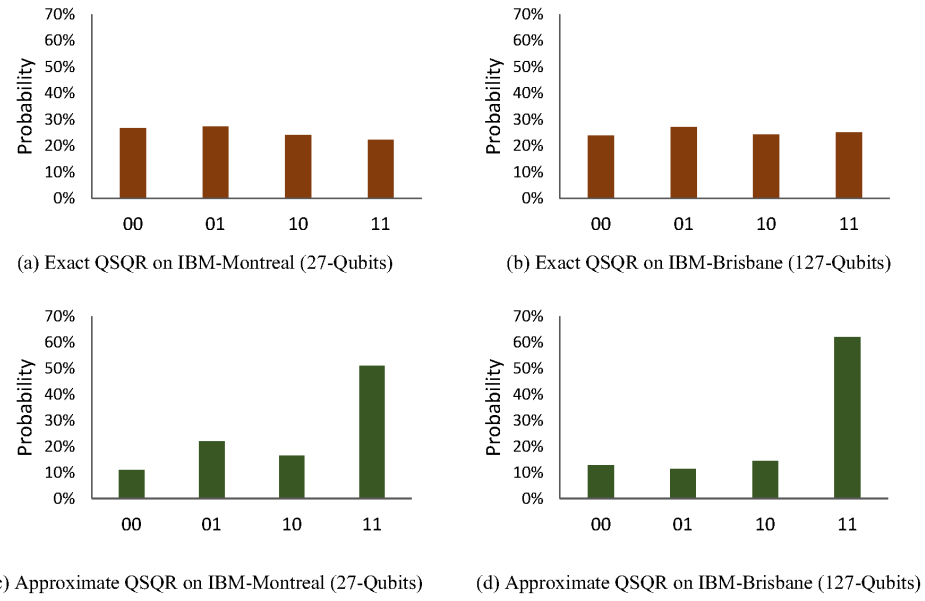


Figure 5.5: Outputs of 4-qubit QSQR circuits: (a) enhanced exact QSQR on IBM-Montreal, (b) enhanced exact QSQR on IBM-Brisbane, (c) approximate QSQR on IBM-Montreal, (d) approximate QSQR on IBM-Brisbane.

output. The input to the circuit is equal to $|5\rangle$ and the expected outputs of exact and approximate circuits are $|2\rangle$ and $|3\rangle$, respectively. Figures fig8(a) and fig8(b) show that the exact circuits do not generate the correct result due to the excessive noise. The probability of the correct result (10) is not higher than the other values, and the output is too noisy on real quantum hardware. In contrast, the approximate circuits generate the expected result (11) and the difference between the probability of the expected output and the next highest observed value is approximately 29.1% (Figure 5.5(c)) on IBM-Montreal and 47.5% on IBM-Brisbane (Figure 5.5(d)). Although the two quantum computers have different noise characterizations, our proposed approximate QSQR circuit is resilient to quantum noise. From Figure 5.5, we conclude that the approximate QSQR circuit is practical and generates the expected results on NISQ devices.

5.5 Evaluation of NR-QNN

In this section, we present results for NR-QNN. We compare the accuracy of QNNs in noiseless simulation, hardware emulation, and NISQ hardware. The goal of this work is to optimize QNN circuits so that they overcome noise when they are run on NISQ devices and generate meaningful results. We do not focus on QNN designs with the highest accuracies, as the accuracy of QNNs is not the focus of this work.

5.5.1 Evaluation Methodology

We conduct experiments using MNIST, Fashion-MNIST, and CIFAR-10 datasets. MNIST and Fashion-MNIST datasets contain 28×28 single-channel images and both share the same number of classes for classification. The MNIST dataset contains images of handwritten digits from zero to nine. The Fashion-MNIST dataset includes images of different pieces of clothing. Each of the two datasets is divided into 50,000 and 10,000 images for training and testing, respectively. The CIFAR-10 dataset contains RGB images with size $3 \times 32 \times 32$ pixels. CIFAR-10 has 50,000 images for training and 10,000 images for testing. CIFAR-10 consists of ten different classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. We use average accuracy as a metric to measure the performance of QNN models. The average accuracy is calculated by performing a forward pass through the network for each sample in the test dataset, measuring the fraction of the samples that are classified correctly.

For MNIST and Fashion-MNIST, we use QNNs with 4 qubits [77]. The encoding section of the QNNs uses 4 rotation gates per qubit (Figure 4.1). As a result, for a 28×28 image, $7 \times 7 = 49$ quantum circuit executions are required to generate $4 \times 49 = 196$ features for the next fully connected layer. For CIFAR-10, we use 6 qubits, where each

qubit uses two rotation gates for the encoding section [77]. Thus, each $3 \times 32 \times 32$ image requires $16 \times 16 = 256$ circuit executions to generate $6 \times 256 = 1536$ output features. After measurements, the output features are concatenated and flattened for the next fully connected layer.

We use PyTorch [107], PennyLane [75], and QISKIT [32] packages to model and train QNNs used in this work. All networks are trained with Adagrad optimizer [122] with a learning rate of 0.5. We use the default PennyLane configuration to run QNNs in noiseless simulations. For hardware emulation, we use fake backends provided in QISKIT V0.45.0 for IBM quantum computers. The fakebackend is used to model noise in quantum hardware. The noise is based on IBM calibration data and involves T_1 and T_2 coherence errors, gate errors, and read-out (measurement) errors. Several techniques are used to model coherence errors, such as Inversion Recovery, Ramsey Experiments, and Hahn Echoes [123]. Errors in single-qubit gates are simulated using a single-qubit depolarization error followed by a single-qubit relaxation error. Similarly, errors in a two-qubit gate are simulated by applying depolarization error and thermal relaxation error on both qubits of the gate. Each measured qubit is flipped with a probability to model the impact of measurement error. IBM provides access to calibrated data through QISKIT APIs. During the simulation, the hardware emulator loads these noises, compiles the circuit with native gates, and generates outputs under the aforementioned noise model. We use the hardware emulator to compare the performance of various QNNs under noise. We also run QNNs on a 127-qubit real IBM quantum computer: IBM-Brisbane. The IBM quantum computers use the Pauli-Z expectation values for the measurement of the outputs. Each QNN is run for 20,000 shots on the IBM-Brisbane.

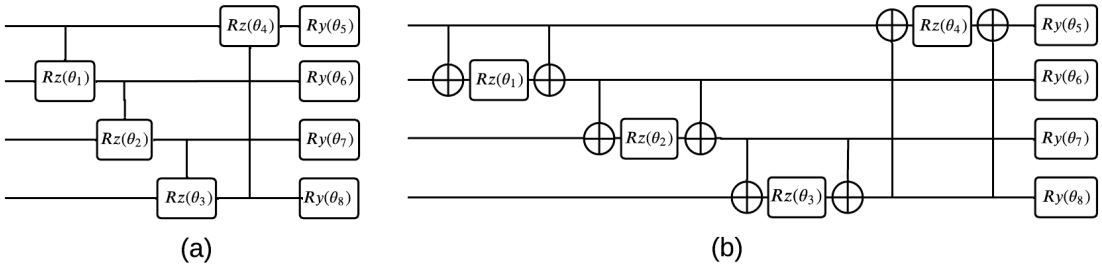


Figure 5.6: PQCs for MNIST and FashionMNIST.

5.5.2 Experimental Results

Figure 4.1 is the skeleton of all QNNs used to evaluate NR-QNN. We change the configuration of the PQC section of the QNNs to evaluate the impact of different quantum circuits on accuracy. By no means is this an exhaustive exploration of design space for PQCs. We only evaluate a few configurations and select the best one for the rest of this paper.

The PQC in Figure 4.1 consists of CNOT and R_y gates. Figure 5.6 shows two other configurations for PQCs for MNIST and Fashion-MNIST. The first one is built out of controlled R_z and R_y gates, and the second one uses blocks of CNOT and R_z gates. Regardless of the configuration of PQCs, the circuits must include two-qubit gates to benefit from entanglement and thus cover larger regions of Hilbert space.

The images in MNIST and Fashion-MNIST are single channels, as all images in the two datasets are black-and-white. However, the CIFAR-10 images are colored and each image has three channels: R, G, and B. To exploit the correlation between the output of CIFAR-10 QNN and the three input channels, we use PQCs that differ from the configurations presented in Figure 5.6. Figure 5.7 shows PQCs for CIFAR-10. While the intra-channel PQC (Figure 5.7(a)) uses entanglement between qubits within a channel,

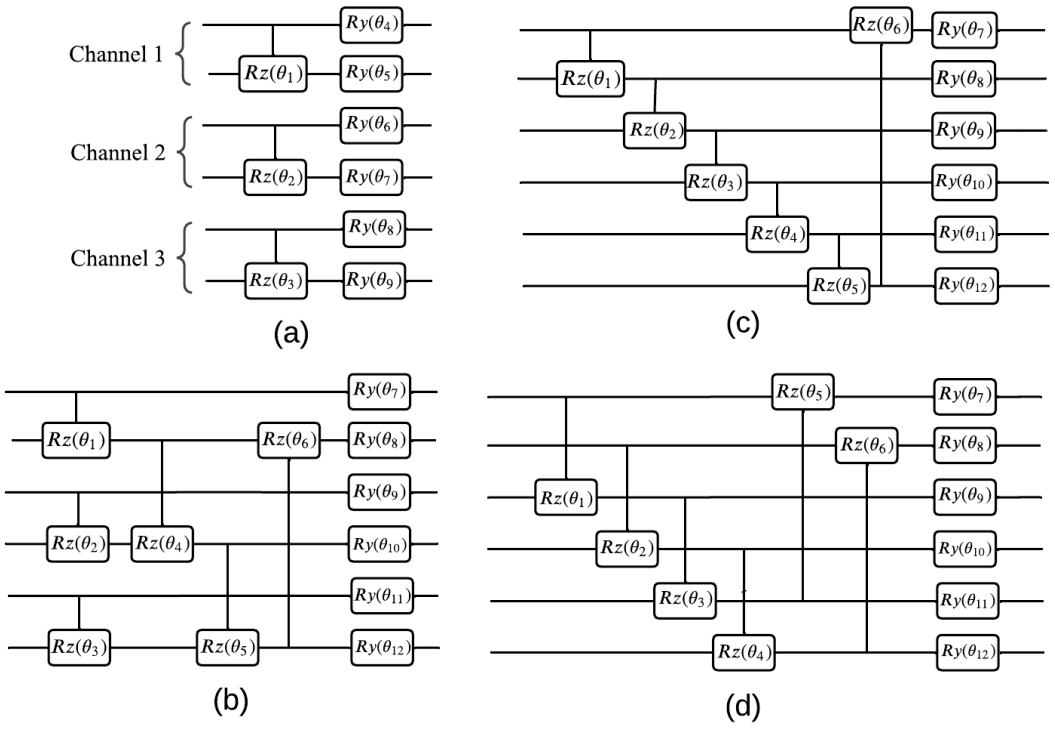


Figure 5.7: PQCs for CIFAR-10. (a) Intra-Channel. (b) Inter-Channel. (c) and (d) Intra and Inter-Channel

the inter-channel PQC (Figure 5.7(b)) exploits entanglement across channels. The inter-/intra-channel PQCs (figures 5.7(c) and 5.7(d)) exploit entanglements both within and across channels, and thus it is expected that they offer higher accuracies. As we already mentioned, the goal of this work is to optimize QNNs to overcome noise in NISQ devices. Exploring the design space of PQCs and offering the optimum circuit is beyond the scope of this work.

Table 5.15 shows the average accuracy of QNNs for MNIST, Fashion-MNIST, and CIFAR-10. The accuracy of MNIST and Fashion-MNIST is higher than CIFAR-10. This is mainly due to the simplicity of the two datasets. Both datasets exploit black-and-white

Table 5.15: Accuracy of Noiseless QNNs

Method	**MNIST**	**FashionMNIST**
CRZ (Figure 5.6(a))	97.50%	79.25%
CNOT (Figure 4.1)	88.33%	71.66%
CNOT+RZ (Figure 5.6(b))	94.17%	78.33%
	CIFAR10	
Intra	25.83%	
Inter	30.83%	
Intra/Inter-1	38.33%	
Intra/Inter-2	40.83%	

images. Based on Table 5.15, we use CR_Z gates for both MNIST and Fashion-MNIST. In CIFAR-10, intra/inter-1 and intra/inter-2 offer higher accuracies. This is expected as they exploit correlations both within and across channels and can differentiate input images better than intra and inter-schemes. We use intra/inter-2 for CIFAR-10 as its accuracy is slightly higher than the intra/inter-1 scheme.

Table 5.16 compares predictions made by fake-backend and noiseless simulations. The goal of NR-QNN is to mitigate the impact of quantum noise on the accuracy of QNNs. Thus, Table 5.16 reports the similarity between pruned QNNs and baseline noiseless QNNs. For example, the similarity of 90% in Table 5.16 shows that 90% of outputs in a pruned QNN on a fake backend is the same as the baseline noiseless QNN. We also report the fraction of gates that are removed due to pruning.

To evaluate how our proposed techniques work in deep circuits, we replicate the PQC section of QNNs. For example, L_3 in Table 5.16 indicates that the PQC has three layers. It is important to note that the fake-backend does not model the entire spectrum of quantum noise, and as a result, the output of a quantum circuit on a real quantum computer is not necessarily the same as the fake-backend simulation. It is used to provide a first-order estimation of how a circuit behaves when it is deployed on a real quantum computer.

Table 5.16: Similarity between Pruned QNNs and Noiseless Simulations and Pruning Rate. The number of layers changes from one to four ($L_1 \sim L_4$).

Layers	Pruning Angle	Similarity (%) / Pruning rate (%)		
		MNIST	FashionMNIST	CIFAR10
L_1	$\pi/4$	87%/12%	87%/13%	**87%**/16%
	$2\pi/4$	87%/13%	**90%**/13%	83%/17%
	$3\pi/4$	**87%**/37%	83%/50%	80%/42%
	$4\pi/4$	83%/38%	83%/50%	80%/58%
	$5\pi/4$	80%/74%	80%/63%	80%/67%
	$6\pi/4$	80%/75%	83%/88%	80%/92%
L_2	$\pi/4$	83%/13%	**83%**/13%	74%/13%
	$2\pi/4$	**87%**/31%	80%/31%	73%/21%
	$3\pi/4$	83%/44%	80%/56%	73%/46%
	$4\pi/4$	83%/69%	80%/63%	70%/54%
	$5\pi/4$	77%/93%	73%/75%	**77%**/63%
	$6\pi/4$	77%/94%	73%/88%	70%/79%
L_3	$\pi/4$	**83%**/8%	**73%**/17%	**77%**/11%
	$2\pi/4$	77%/25%	67%/21%	70%/33%
	$3\pi/4$	77%/42%	73%/29%	70%/53%
	$4\pi/4$	77%/46%	63%/46%	67%/69%
	$5\pi/4$	73%/54%	63%/75%	63%/72%
	$6\pi/4$	73%/79%	63%/88%	60%/83%
L_4	$\pi/4$	**83%**/9%	**67%**/9%	**70%**/17%
	$2\pi/4$	77%/22%	63%/34%	60%/31%
	$3\pi/4$	73%/44%	63%/53%	60%/44%
	$4\pi/4$	70%/56%	63%/56%	60%/63%
	$5\pi/4$	67%/72%	53%/75%	50%/67%
	$6\pi/4$	67%/78%	53%/84%	50%/79%

We prune QNNs using thresholds ranging from $\pi/4$ to $6\pi/4$ in increments of $\pi/4$, as shown in Table 5.16. We begin pruning at $\pi/4$ because smaller thresholds would have a negligible impact on the circuit—very few parameters fall below such small magnitudes—resulting in minimal pruning and insignificant reductions in cost. While increasing the threshold reduces the cost of the pruned circuit as a greater number of gates are elim-

inated, a higher threshold does not necessarily offer a higher level of accuracy. As a PQC shrinks due to pruning, a smaller subset of the Hilbert space is covered by the circuit, which makes it more difficult for the steepest descent algorithm to optimize the network. For each dataset, we select the highest accuracy per PQC layer and use it for the rest of the paper (entries with bold fonts in Table 5.16).

Table 5.17: Running MNIST, FashionMNIST, and CIFAR-10 on IBM-Brisbane.

Dataset	Layers	Pruning Angle	Baseline	Pruning QNN	NR-QNN
MNIST	L_1	$3\pi/4$	93%	100%	100%
	L_2	$2\pi/4$	77%	87%	93%
	L_3	$1\pi/4$	63%	80%	90%
	L_4	$1\pi/4$	57%	73%	80%
FashionMNIST	L_1	$2\pi/4$	67%	80%	90%
	L_2	$1\pi/4$	63%	77%	83%
	L_3	$1\pi/4$	57%	63%	80%
	L_4	$1\pi/4$	43%	57%	77%
CIFAR-10	L_1	$1\pi/4$	67%	80%	87%
	L_2	$5\pi/4$	57%	67%	77%
	L_3	$1\pi/4$	53%	60%	73%
	L_4	$1\pi/4$	30%	53%	67%

Table 5.17 shows the result of running QNNs on the IBM-Brisbane quantum computer. The third column of Table 5.17 shows the angles determined by pruning (Table 5.16). As the number of layers increases, the similarity rate drops for the baseline QNN rapidly (fourth column). This is expected as a deeper PQC results in more quantum noise on IBM-Brisbane, causing a rapid increase in dissimilarity between the output of baseline QNN and noiseless simulations. As an example, in MNIST, the similarity rate drops from 93% in L_1 to 57% in L_4. The fifth column in Table 5.17 shows the similarity between noiseless circuits and pruned QNNs, whereas the last column shows the results for NR-QNN. Pruning reduces quantum noise and enhances accuracy over the baseline scheme.

NR-QNN improves accuracy even further and reduces the gap between NR-QNN and noiseless simulations. On average, NR-QNN increases the similarity between QNNs run on IBM-Brisbane and noiseless simulations by 17%, 19%, 23%, and 31% for L_1, L_2, L_3, and L_4, respectively.

5.6 Evaluation of QRAM

The quantum circuits for QRAM are modeled and trained using the PyTorch [107], PennyLane [75], and QISKIT [32] packages. We use the digits dataset from the UCI Machine Learning Repository to train and test the QRAM. The dataset consists of digits 0 to 9, with each image having 8×8 pixels. All networks are trained using the Adam optimizer with a batch size of 16 and a learning rate of 0.001. We employ 9 address lines for the QRAM, as we selected 360 images for classification. We use 80% of the dataset for training and the rest for testing. The output of the QRAM is fed into a QNN for image classification. The goal of this work is not to design the optimal QNN classifier; therefore, we use the same architecture for both the QRAM and the QNN classifier, as shown in Figure 4.5. The QRAM is first trained for 100 epochs. Then, it is pruned and fine-tuned for another 100 epochs (step 1 in Figure 4.6). The QNN classifier is fed with the trained QRAM from step 1 and is trained for 100 epochs (step 2 in Figure 4.6). Finally, trained QRAM and QNN are evaluated on noisy quantum hardware (step 3 in Figure 4.6).

Figure 5.8 illustrates the MSE loss of a QRAM with a single PQC on the test set as training progresses. The MSE loss of the QRAM decreases rapidly, reaching 0.004. Figure 5.9 presents the accuracy of the QNN classifier for both noiseless simulation and the IBM-Brisbane quantum device. The number of PQCs in the QRAM and QNN varies from one to three. In the noiseless simulation, the accuracy reaches 100%. However, when the

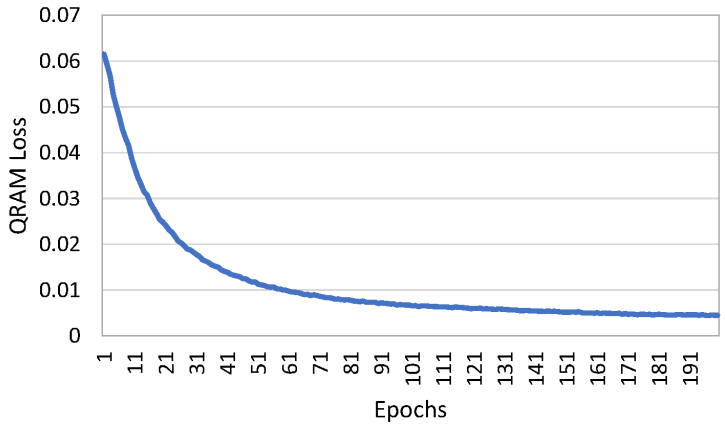


Figure 5.8: MSE loss of QRAM vs. epochs.

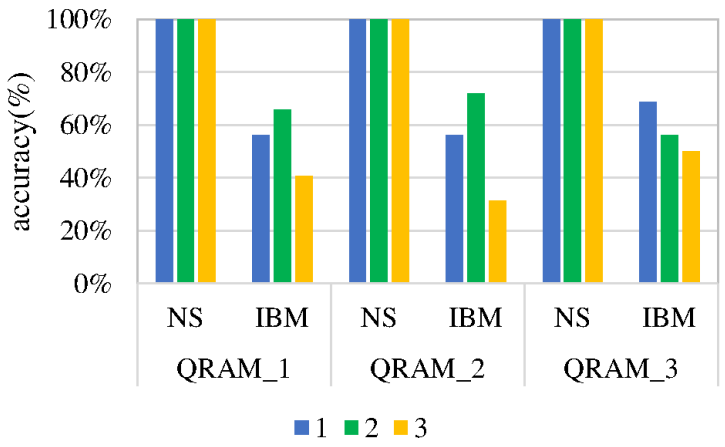


Figure 5.9: Accuracy of the QNN classifier for noiseless simulation and IBM-Brisbane (IBM). QRAM_i denotes a QRAM with i PQCs. The legends 1, 2, and 3 correspond to QNNs with 1, 2, and 3 PQCs, respectively.

quantum circuits are deployed on IBM-Brisbane, the accuracy significantly decreases. For instance, in the case of QRAM_2 with three PQCs in the QNN, the accuracy is 31.25%. From Figure 5.9, it can be concluded that QRAM is impractical on current NISQ devices.

Table 5.18 shows the similarity between noiseless simulations and implementation on

Table 5.18: Similarity vs. Pruning angle on ibm-brisbane

	#QNN Layer	Pruning Angle	Similarity (%)	Pruning Rate (%)	Circuit Depth	#of CNOT Gates
QRAM-1	1	$1\pi/10$	81.25	31.25	164	54
		$2\pi/10$	96.88	42.37	133	46
		$3\pi/10$	100.00	63.20	97	28
	2	$1\pi/10$	75.00	28.24	234	83
		$2\pi/10$	84.38	43.06	176	74
		$3\pi/10$	96.88	64.35	114	38
		$4\pi/10$	100.00	84.73	25	9
	3	$1\pi/10$	96.88	25.35	505	138
		$2\pi/10$	87.50	46.18	265	93
		$3\pi/10$	93.75	67.71	104	56
		$4\pi/10$	100.00	78.82	56	21
QRAM-2	1	$1\pi/10$	59.38	36.58	263	84
		$2\pi/10$	78.13	49.08	115	48
		$3\pi/10$	90.63	68.98	90	38
		$4\pi/10$	100.00	78.24	44	20
	2	$1\pi/10$	81.25	30.91	483	133
		$2\pi/10$	84.38	43.06	214	90
		$3\pi/10$	90.63	73.27	119	48
		$4\pi/10$	100.00	81.60	60	33
	3	$1\pi/10$	71.88	36.11	432	153
		$2\pi/10$	75.00	55.00	170	98
		$3\pi/10$	75.00	63.61	158	67
		$4\pi/10$	100.00	75.84	99	57
QRAM-3	1	$1\pi/10$	71.88	28.13	411	124
		$2\pi/10$	78.13	52.09	162	76
		$3\pi/10$	81.25	72.23	81	40
		$4\pi/10$	81.25	75.70	114	53
		$5\pi/10$	100.00	86.12	36	16
	2	$1\pi/10$	59.38	25.28	596	165
		$2\pi/10$	78.13	51.11	356	123
		$3\pi/10$	81.25	70.56	87	44
		$4\pi/10$	90.63	80.56	90	36
		$5\pi/10$	100.00	86.67	27	10
	3	$1\pi/10$	56.25	28.94	610	194
		$2\pi/10$	84.38	51.39	332	126
		$3\pi/10$	93.75	70.37	156	84

IBM-Brisbane for various pruning thresholds. It is important to note that the accuracy of noiseless simulations is reported in Figure 5.9. The fourth column in the table indicates the fraction of gates removed through pruning. We prune QRAM+QNN starting with a threshold of $\pi/10$ and increment the threshold in steps of $\pi/10$ until the similarity ceases to improve. The fifth column in the table shows circuit depth. As the pruning

threshold increases, the depth of the quantum circuits reduces. While higher thresholds reduce the cost of the pruned circuit by eliminating more gates, they do not necessarily lead to better accuracy. As the QRAM+QNN is pruned, it covers a smaller portion of the Hilbert space, making it more challenging for the steepest descent algorithm to optimize the network. Thus, pruning represents a trade-off between circuit simplicity and similarity. Table 5.18 shows that pruning offers a viable solution for the implementation of QRAMs on contemporary NISQ devices.

When the quantum circuit for QRAM is implemented on a real quantum backend, two entangled qubits must be mapped to physically adjacent qubits. If the required physical qubits are not adjacent, a series of SWAP gates is used to bring them next to each other. However, the SWAP gate is not a primitive gate on IBM-Brisbane; each SWAP gate is transpiled into three CNOT gates. The error rate of a CNOT gate is significantly higher than that of a single-qubit gate [108]. Consequently, a quantum circuit with a higher number of CNOT gates is more vulnerable to quantum noise. The sixth column in Table 5.18 lists the number of CNOT gates in both the baseline and pruned QRAM. Pruning reduces the number of CNOT gates by up to 96.1% in comparison with the baseline scheme. By decreasing the use of gates prone to quantum error, pruning facilitates the successful execution of QRAM on IBM-Brisbane.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

This work has addressed the challenges of designing and implementing practical quantum arithmetic and neural network circuits for NISQ devices. Motivated by the limitations of current quantum hardware—such as short coherence times, limited qubit counts, and high susceptibility to noise—this work focused on developing quantum circuits that are both functionally correct and optimized for resource efficiency and noise resilience.

The research began by proposing practical arithmetic units, including four novel approximate quantum adders with constant depth and reduced T-count, making them ideal for NISQ deployment. These adders are scalable, as their depth is independent of input size, and they outperform existing designs in both depth and T-count. Building on these, we introduced approximate quantum multipliers, dividers, and a new square root circuit based on a non-restoring algorithm, further reducing quantum resource requirements. The proposed circuits were mathematically proven to maintain accuracy while achieving signif-

icant reductions in gate count and qubit usage, and their application in image processing demonstrated their practical utility.

In addition to arithmetic circuits, the work explored quantum machine learning by developing a noise-resilience QNN architecture. The QNN exploits parameterized rotation gates, with trainable angles optimized via backpropagation. Entanglement is achieved using CNOT and CRz gates to capture data correlations. The optimized QNN was then implemented on real quantum processors, utilizing sensitivity analysis techniques to minimize the effects of noise and generate valid results on real quantum backends. Experimental results on IBM quantum computers and various datasets revealed that even shallow QNNs can achieve high accuracy, with rapid convergence compared to classical neural networks. This highlights the potential of QNNs for future quantum machine learning applications.

A unifying theme across the arithmetic circuits, QNN architecture, and QRAM design developed in this work is the deliberate use of approximation and noise-aware optimization to enable practical execution on today's NISQ hardware. Although these components serve different computational purposes—arithmetic for data transformation, QNNs for learning tasks, and QRAM for structured data access—they share a consistent design philosophy: minimize circuit depth, reduce gate count, required qubits, and explicitly account for hardware noise during both circuit construction and experimental evaluation. By adopting approximation as a first-class design strategy, each module can operate reliably under the constraints of limited coherence times, restricted qubit counts, and gate infidelity. This creates a cohesive framework in which arithmetic blocks, memory access circuits, and neural layers are not isolated contributions but complementary components of a broader effort to develop practical, hardware-efficient quantum computation.

The contributions of this work are multifaceted. First, it provides a clear methodology for designing quantum circuits that are both efficient and resilient to noise, a critical re-

quirement for practical quantum computing in the NISQ era. Second, it bridges the gap between theoretical circuit design and experimental realization by validating the proposed circuits on real quantum hardware. Third, it extends the applicability of quantum computing to machine learning tasks, paving the way for future research in quantum-enhanced data analysis and artificial intelligence.

Furthermore, the work presented here has broader implications for quantum algorithm development. The principles of approximate computing, noise-aware design, and hardware-oriented optimization can be generalized to other quantum algorithms, guiding the design of scalable and robust quantum systems as hardware continues to evolve. The demonstrated feasibility of running meaningful quantum computations on NISQ devices is a significant step toward realizing the full potential of quantum computing.

6.2 Experimental Limitations

Despite the promising results and advancements presented in this work, several experimental limitations were encountered during the implementation and evaluation of quantum circuits on real quantum hardware. First, the limited number of qubits available on current NISQ devices imposes a strict constraint on the scale of executable circuits. Consequently, experiments are typically restricted to small problem sizes, often involving only a few to several dozen qubits, depending on the hardware. Algorithms intended for real quantum devices must be designed to fit within these qubit limitations, which inherently bound the complexity and size of feasible quantum computations.

Second, the short coherence times of qubits and the high error rates in quantum gates and measurements — inherent characteristics of current NISQ devices — introduce significant noise. This reduces the fidelity of experimental results and necessitates the use of

error mitigation and sensitivity analysis techniques to obtain reliable outcomes

Third, limited access to quantum hardware due to queuing times and resource sharing with other users sometimes delays experimental progress and reduces the number of runs possible for statistical validation. Additionally, the lack of advanced error correction and the variability in hardware calibration further impacted the reproducibility and reliability of results. These limitations highlight the need for continued improvements in quantum hardware and software, as well as the importance of developing noise-resilient algorithms and robust benchmarking methodologies for future research.

6.3 Future Works

While this work has made significant progress in designing efficient quantum arithmetic and neural network circuits, the field of quantum computing remains in its infancy, and numerous exciting research directions are open for exploration. The following avenues are particularly promising for advancing both the theoretical and practical aspects of quantum circuit design and application.

6.3.1 Circuit-Centric Future Works

1) Advanced Quantum Arithmetic and Functional Units

Future research can focus on extending the proposed approximate arithmetic circuits to support a broader range of mathematical operations. Developing efficient, noise-resilient quantum circuits for these operations will be crucial for enabling more complex quantum algorithms in scientific computing, cryptography, and signal processing. Additionally,

optimizing these circuits for larger-scale quantum devices could further expand their applicability.

2) Error Mitigation, Correction, and Noise-Aware Design

As NISQ devices are inherently noisy, robust error mitigation and correction strategies tailored to the specific structure of the proposed circuits are essential. Research can investigate adaptive error mitigation techniques, circuit-level error suppression, and the use of logical qubits or error-correcting codes that are compatible with approximate computing paradigms. Furthermore, developing analytical models to predict and minimize the impact of noise on circuit fidelity will help guide the design of future quantum hardware and software.

3) Cross-Platform Evaluation and Hardware Co-Design

Evaluating the performance of the proposed circuits on a variety of emerging quantum hardware platforms, including those from IBM, Google, Rigetti, IonQ, and others, will provide valuable insights into hardware-specific optimizations and limitations. Collaborative research between circuit designers and hardware engineers can lead to co-design strategies that tailor circuit architectures to the strengths and constraints of specific quantum technologies, ultimately improving overall system performance and reliability.

4) Theoretical Foundations and Complexity Analysis

Further investigation into the theoretical limits of approximate quantum computing, including complexity analysis, error bounds, and trade-offs between accuracy and resource

usage, will deepen our understanding of what is achievable on NISQ devices. Establishing rigorous frameworks for quantifying the benefits and limitations of approximate circuits will guide future algorithm and hardware development.

6.3.2 ML-Centric Future Works

The QNN architecture introduced in this work opens the door to a new class of quantum machine learning models. Future work can explore alternative quantum data encoding schemes, quantum activation functions, and advanced training algorithms that leverage hybrid quantum-classical optimization. Investigating the scalability of QNNs to larger datasets and more complex learning tasks, as well as benchmarking their performance against classical and other quantum models, will be vital for assessing their practical utility.

In summary, the results presented in this work lay the groundwork for practical quantum computing applications and open up a rich landscape of research opportunities. By pursuing these directions, the quantum computing community can accelerate the transition from theoretical exploration to impactful, real-world solutions, ultimately unlocking the transformative potential of quantum technologies.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[2] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 439, pp. 553–558, 1992.

[3] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 212–219, Association for Computing Machinery, 1996.

[5] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[6] M. Brooks, "Beyond quantum supremacy: the hunt for useful quantum computers," *Nature*, vol. 574, pp. 19–21, 10 2019.

[7] G. Q. AI and Collaborators, "Observation of constructive interference at the edge of quantum ergodicity," *Nature*, vol. 646, no. 8086, pp. 825–830, 2025.

[8] IBM, "Ibm q experience systems: Quantum computer systems." Online. Available: https://www.ibm.com/quantum-computing/systems/, Accessed: Nov. 3, 2024.

[9] F. Arute, K. Arya, R. Babbush, D. Bacon, J. Bardin, R. Barends, R. Biswas, S. Boixo, F. Brandao, D. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, and J. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, 10 2019.

[10] IBM Quantum, "On "quantum supremacy"," October 2019. Accessed: 2025-09-13.

[11] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "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*, ASPLOS '19, (New York, NY, USA), p. 1015–1029, Association for Computing Machinery, 2019.

[12] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-52, (New York, NY, USA), p. 253–265, Association for Computing Machinery, 2019.

[13] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, p. 1031–1044, ACM, Apr. 2019.

[14] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), p. 1001–1014, Association for Computing Machinery, 2019.

[15] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, 2014.

[16] H. V. Jayashree, H. Thapliyal, H. R. Arabnia, and V. K. Agrawal, "Ancilla-input and garbage-output optimized design of a reversible quantum integer multiplier," *J. Supercomput.*, vol. 72, p. 1477–1493, Apr. 2016.

[17] E. Muñoz-Coreas and H. Thapliyal, "Quantum circuit design of a t-count optimized integer multiplier," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 729–739, 2019.

[18] C.-C. Lin, A. Chakrabarti, and N. K. Jha, "Qlib: Quantum module library," *J. Emerg. Technol. Comput. Syst.*, vol. 11, Oct. 2014.

[19] S. Kotiyal, H. Thapliyal, and N. Ranganathan, "Circuit for reversible quantum multiplier based on binary tree optimizing ancilla and garbage bits," in *Proceedings - 27th International Conference on VLSI Design, VLSID 2014; Held Concurrently with 13th International Conference on Embedded Systems Design*, Proceedings of the IEEE International Conference on VLSI Design, pp. 545–550, 2014. 27th International Conference on VLSI Design, VLSID 2014 - Held Concurrently with 13th In-

ternational Conference on Embedded Systems Design ; Conference date: 05-01-2014 Through 09-01-2014.

[20] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, "Layered architecture for quantum computing," *Phys. Rev. X*, vol. 2, p. 031007, Jul 2012.

[21] A. Paler, I. Polian, K. Nemoto, and S. J. Devitt, "Fault-tolerant, high-level quantum circuits: form, compilation and description," *Quantum Science and Technology*, vol. 2, p. 025003, Apr. 2017.

[22] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, p. 021318, 06 2019.

[23] M. B. Plenio and P. L. Knight, "Decoherence limits to quantum computation using trapped ions," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 453, pp. 2017–2041, Oct. 1997.

[24] M. McEwen, D. Kafri, Z. Chen, J. Atalaya, K. J. Satzinger, C. Quintana, P. V. Klimov, D. Sank, C. Gidney, A. G. Fowler, F. Arute, K. Arya, B. Buckley, B. Burkett, N. Bushnell, B. Chiaro, R. Collins, S. Demura, A. Dunsworth, C. Erickson, B. Foxen, M. Giustina, T. Huang, S. Hong, E. Jeffrey, S. Kim, K. Kechedzhi, F. Kostritsa, P. Laptev, A. Megrant, X. Mi, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Niu, A. Paler, N. Redd, P. Roushan, T. C. White, J. Yao, P. Yeh, A. Zalcman, Y. Chen, V. N. Smelyanskiy, J. M. Martinis, H. Neven, J. Kelly, A. N. Korotkov, A. G. Petukhov, and R. Barends, "Removing leakage-induced correlated errors in supercon-

ducting quantum error correction," *Nature Communications*, vol. 12, no. 1, p. 1761, 2021.

[25] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, p. 818–830, June 2013.

[26] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, June 2018.

[27] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Phys. Rev. A*, vol. 76, p. 042319, Oct 2007.

[28] D. Nigg, M. Müller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt, "Quantum computations on a topologically encoded qubit," *Science*, vol. 345, no. 6194, pp. 302–305, 2014.

[29] D. M. Zajac, T. M. Hazard, X. Mi, E. Nielsen, and J. R. Petta, "Scalable gate architecture for a one-dimensional array of semiconductor spin qubits," *Phys. Rev. Appl.*, vol. 6, p. 054013, Nov 2016.

[30] M. Saffman, T. G. Walker, and K. Mølmer, "Quantum information with rydberg atoms," *Rev. Mod. Phys.*, vol. 82, pp. 2313–2363, Aug 2010.

[31] C. Rigetti and M. Devoret, "Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies," *Phys. Rev. B*, vol. 81, p. 134507, Apr 2010.

[32] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," 2024.

[33] F. Hua, Y. Jin, Y. Chen, S. Vittal, K. Krsulich, L. S. Bishop, J. Lapeyre, A. Javadi-Abhari, and E. Z. Zhang, "Caqr: A compiler-assisted approach for qubit reuse through dynamic circuit," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, (New York, NY, USA), p. 59–71, Association for Computing Machinery, 2023.

[34] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[35] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4: 2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, p. 1352–1361, Apr. 2017.

[36] W. Liu, T. Zhang, E. McLarnon, M. Oneill, P. Montuschi, and F. Lombardi, "Design and analysis of majority logic based approximate adders and multipliers," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, pp. 1–1, 07 2019.

[37] G. Florio and D. Picca, "Quantum implementation of elementary arithmetic operations," *International Journal of Quantum Information*, April 2004. Add missing details if available.

[38] L. Ruiz-Perez and J. C. Garcia-Escartin, "Quantum arithmetic with the quantum fourier transform," *Quantum Information Processing*, vol. 16, Apr. 2017.

[39] H. M. Babu, "Cost-efficient design of a quantum multiplier—accumulator unit," *Quantum Information Processing*, vol. 16, p. 1–38, Jan. 2017.

[40] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, vol. 17, p. 525–532, Nov. 1973.

[41] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, Nov 1995.

[42] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pp. 220–225, 2008.

[43] S. Sultana and K. Radecka, "Reversible implementation of square-root circuit," in *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*, pp. 141–144, 2011.

[44] M. K. Bhaskar, S. Hadfield, A. Papageorgiou, and I. Petras, "Quantum algorithms and circuits for scientific computing," *Quantum Info. Comput.*, vol. 16, p. 197–236, Mar. 2016.

[45] L. Parrilla, A. Lloris, E. Castillo, and A. García, "Table-free seed generation for hardware newton–raphson square root and inverse square root implementations in iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6985–6995, 2022.

[46] S. Samavi, A. Sadrabadi, and A. Fanian, "Modular array structure for non-restoring square root circuit," *J. Syst. Archit.*, vol. 54, p. 957–966, Oct. 2008.

[47] A. AnanthaLakshmi and G. F. Sudha, "Design of a reversible floating-point square root using modified non-restoring algorithm," *Microprocessors and Microsystems*, vol. 50, pp. 39–53, 2017.

[48] S. Swathi, S. Sushma, V. Bindusree, L. Babitha, G. K. Sukesh, S. C. Venkateswarlu, V. Vijay, and R. R. Vallabhuni, "Implementation of an energy-efficient binary square rooter using reversible logic by applying the non-restoring algorithm," in *2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4)*, pp. 1–6, 2021.

[49] S. Dutta, Y. Tavva, D. Bhattacharjee, and A. Chattopadhyay, "Efficient quantum circuits for square-root and inverse square-root," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pp. 55–60, 2020.

[50] M. Wilkes, *The Preparation of Programs for an Electronic Digital Computer: With Special Reference to the EDSAC and the Use of a Library of Subroutines*. Addison-Wesley mathematics series, Addison-Wesley Press, 1951.

[51] E. Muñoz Coreas and H. Thapliyal, "T-count and qubit optimized quantum circuit design of the non-restoring square root algorithm," *J. Emerg. Technol. Comput. Syst.*, vol. 14, Oct. 2018.

[52] Y. Li and W. Chu, "Parallel-array implementations of a non-restoring square root algorithm," in *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, pp. 690–695, 1997.

[53] H. Jiang, L. liu, F. Lombardi, and J. Han, "Low-power unsigned divider and square root circuit designs using adaptive approximation," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1635–1646, 2019.

[54] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, (Leuven, BEL), p. 258–261, European Design and Automation Association, 2017.

[55] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, Mar. 2016.

[56] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6, 2013.

[57] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[58] U. A. Kumar, V. Ramchandani, S. S. Hashmi, S. Dubey, and S. E. Ahmed, "Low power approximate divider and square root circuits for error resilient applications," in *2023 11th International Conference on Intelligent Systems and Embedded Design (ISED)*, pp. 1–6, 2023.

[59] N. Arya, T. Soni, M. Pattanaik, and G. Sharma, "Bit significance based reconfigurable approximate restoring dividers and square rooters," *Microelectronics Journal*, vol. 104, p. 104861, 2020.

[60] N. Arya, T. Soni, M. Pattanaik, and G. Sharma, "Area and energy efficient approximate square rooters for error resilient applications," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pp. 90–95, 2020.

[61] L. Bandil and B. C. Nagar, "Modified restoring array-based power efficient approximate square root circuit and its application," *Integr. VLSI J.*, vol. 94, Jan. 2024.

[62] B. Gaur, T. Humble, and H. Thapliyal, "Noise-resilient and reduced depth approximate adders for nisq quantum computing," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, GLSVLSI '23, p. 427–431, ACM, June 2023.

[63] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[64] N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, "Continuous-variable quantum neural networks," *Phys. Rev. Res.*, vol. 1, p. 033063, Oct 2019.

[65] M. Alam, S. Kundu, R. O. Topaloglu, and S. Ghosh, "Quantum-classical hybrid machine learning for image classification (iccad special session paper)," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, p. 1–7, IEEE Press, 2021.

[66] K. Batra, K. M. Zorn, D. H. Foil, E. Minerali, V. O. Gawriljuk, T. R. Lane, and S. Ekins, "Quantum machine learning algorithms for drug discovery applications," *Journal of Chemical Information and Modeling*, vol. 61, p. 2641–2647, May 2021.

[67] M. Pistoia, S. F. Ahmad, A. Ajagekar, A. Buts, S. Chakrabarti, D. Herman, S. Hu, A. Jena, P. Minssen, P. Niroula, A. Rattew, Y. Sun, and R. Yalovetzky, "Quantum machine learning for finance," 2021.

[68] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Phys. Rev. Lett.*, vol. 113, p. 130503, Sep 2014.

[69] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, "Circuit-centric quantum classifiers," *Phys. Rev. A*, vol. 101, p. 032308, Mar 2020.

[70] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Transformation of quantum states using uniformly controlled rotations," *Quantum Info. Comput.*, vol. 5, p. 467–473, Sept. 2005.

[71] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory," *Phys. Rev. Lett.*, vol. 100, p. 160501, Apr 2008.

[72] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, p. 226, Feb. 2020.

[73] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.

[74] L. Banchi and G. E. Crooks, "Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule," *Quantum*, vol. 5, p. 386, Jan. 2021.

[75] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Ban-

ning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," 2022.

[76] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, p. 1120–1128, JMLR.org, 2016.

[77] M. Alam and S. Ghosh, "Qnet: A scalable and noise-resilient quantum neural network architecture for noisy intermediate-scale quantum computers," *Frontiers in Physics*, vol. 9, 2022.

[78] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, p. 023023, Feb. 2016.

[79] R. Orús, S. Mugel, and E. Lizaso, "Quantum computing for finance: Overview and prospects," *Reviews in Physics*, vol. 4, p. 100028, 2019.

[80] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, "Quantum circuit simplification and level compaction," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, p. 436–444, Mar. 2008.

[81] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum circuit placement: optimizing qubit-to-qubit interactions through mapping quantum circuits into a physical experiment," in *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, (New York, NY, USA), p. 962–965, Association for Computing Machinery, 2007.

[82] P. Murali, D. C. Mckay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, (New York, NY, USA), p. 1001–1016, Association for Computing Machinery, 2020.

[83] Y. Ding, P. Gokhale, S. F. Lin, R. Rines, T. Propson, and F. T. Chong, "Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation ," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, (Los Alamitos, CA, USA), pp. 201–214, IEEE Computer Society, Oct. 2020.

[84] R. Barends, C. M. Quintana, A. G. Petukhov, Y. Chen, D. Kafri, K. Kechedzhi, R. Collins, O. Naaman, S. Boixo, F. Arute, K. Arya, D. Buell, B. Burkett, Z. Chen, B. Chiaro, A. Dunsworth, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, J. Kelly, P. V. Klimov, F. Kostritsa, D. Landhuis, E. Lucero, M. McEwen, A. Megrant, X. Mi, J. Mutus, M. Neeley, C. Neill, E. Ostby, P. Roushan, D. Sank, K. J. Satzinger, A. Vainsencher, T. White, J. Yao, P. Yeh, A. Zalcman, H. Neven, V. N. Smelyanskiy, and J. M. Martinis, "Diabatic gates for frequency-tunable superconducting qubits," *Phys. Rev. Lett.*, vol. 123, p. 210501, Nov 2019.

[85] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the*

Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '19, (New York, NY, USA), p. 987–999, Association for Computing Machinery, 2019.

[86] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits ," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Los Alamitos, CA, USA), pp. 692–708, IEEE Computer Society, Apr. 2022.

[87] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "Qoc: quantum on-chip training with parameter shift and gradient pruning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, (New York, NY, USA), p. 655–660, Association for Computing Machinery, 2022.

[88] S. Arunachalam, V. Gheorghiu, T. Jochym-O'Connor, M. Mosca, and P. V. Srinivasan, "On the robustness of bucket brigade quantum ram," *New Journal of Physics*, vol. 17, p. 123010, Dec. 2015.

[89] A. Paler, O. Oumarou, and R. Basmadjian, "Parallelizing the queries in a bucket-brigade quantum random access memory," *Phys. Rev. A*, vol. 102, p. 032608, Sep 2020.

[90] V. Giovannetti, S. Lloyd, and L. Maccone, "Architectures for a quantum random access memory," *Phys. Rev. A*, vol. 78, p. 052310, Nov 2008.

[91] C. T. Hann, G. Lee, S. Girvin, and L. Jiang, "Resilience of quantum random access memory to generic noise," *PRX Quantum*, vol. 2, p. 020311, Apr 2021.

[92] D. K. Park, F. Petruccione, and J.-K. K. Rhee, "Circuit-based quantum random access memory for classical data," *Scientific Reports*, vol. 9, no. 1, p. 3949, 2019.

[93] L. Bugalho, E. Z. Cruzeiro, K. C. Chen, W. Dai, D. Englund, and Y. Omar, "Resource-efficient simulation of noisy quantum circuits and application to network-enabled qram optimization," *npj Quantum Information*, vol. 9, no. 1, p. 105, 2023.

[94] K. C. Chen, W. Dai, C. Errando-Herranz, S. Lloyd, and D. Englund, "Scalable and high-fidelity quantum random access memory in spin-photon networks," *PRX Quantum*, vol. 2, p. 030319, Aug 2021.

[95] M. Y. Niu, A. Zlokapa, M. Broughton, S. Boixo, M. Mohseni, V. Smelyanskyi, and H. Neven, "Entangling quantum generative adversarial networks," *Phys. Rev. Lett.*, vol. 128, p. 220505, Jun 2022.

[96] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, p. 043001, Nov. 2019.

[97] K. Phalak, J. Li, and S. Ghosh, "Approximate quantum random access memory architectures," 2022.

[98] K. Phalak, J. Li, and S. Ghosh, "Trainable pqc-based qram for quantum storage," *IEEE Access*, vol. 11, pp. 51892–51899, 2023.

[99] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *Computers, IEEE Transactions on*, vol. 62, pp. 1760–1771, 09 2013.

[100] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs, 2nd Edition*. New York: Oxford University Press, 2011.

[101] A. Khosropour, H. Aghababa, and B. Forouzandeh, "Quantum division circuit based on restoring division algorithm," in *2011 Eighth International Conference on Information Technology: New Generations*, pp. 1037–1040, 2011.

[102] L. Jamal and H. M. H. Babu, "Efficient approaches to design a reversible floating point divider," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3004–3007, 2013.

[103] Y. Li and W. Chu, "A new non-restoring square root algorithm and its vlsi implementations," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 538–544, 1996.

[104] K. E. Wires and M. J. Schulte, "Reciprocal and reciprocal square root units with operand modification and multiplication," *J. VLSI Signal Process. Syst.*, vol. 42, p. 257–272, Mar. 2006.

[105] Y. Levine, D. Yakira, N. Cohen, and A. Shashua, "Deep learning and quantum entanglement: Fundamental connections with implications to network design," in *International Conference on Learning Representations*, 2018.

[106] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang, "Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design," *Quantum*, vol. 4, p. 341, Oct. 2020.

[107] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: an imperative style, high-performance deep learning library*, pp. 8026–8037. Red Hook, NY, USA: Curran Associates Inc., 2019.

[108] IBM Quantum Computing, "Ibm quantum computing." Online. Available: https://quantum-computing.ibm.com/services/resources?tab=systems. [Accessed: Jul. 10, 2024].

[109] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), p. 3320–3328, MIT Press, 2014.

[110] J. P. Rauschecker, "Neuronal mechanisms of developmental plasticity in the cat's visual system," *Human Neurobiology*, vol. 3, no. 2, pp. 109–114, 1984.

[111] M. H. Devoret and R. J. Schoelkopf, "Superconducting Circuits for Quantum Information: An Outlook," *Science*, vol. 339, pp. 1169–1174, Mar. 2013.

[112] P. V. Klimov, J. Kelly, Z. Chen, M. Neeley, A. Megrant, B. Burkett, R. Barends, K. Arya, B. Chiaro, Y. Chen, A. Dunsworth, A. Fowler, B. Foxen, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, E. Lucero, J. Y. Mutus, O. Naaman, C. Neill, C. Quintana, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, T. C. White, S. Boixo, R. Babbush, V. N. Smelyanskiy, H. Neven, and J. M. Martinis, "Fluctuations of energy-relaxation times in superconducting qubits," *Phys. Rev. Lett.*, vol. 121, p. 090502, Aug 2018.

[113] SIPI Image Database, "Sipi image database." Online. Available: https://sipi.usc.edu/database/, Accessed: Nov. 3, 2021.

[114] M. S. Lau, K.-V. Ling, and Y.-C. Chu, "Energy-aware probabilistic multiplier: design and analysis," in *Proceedings of the 2009 International Conference on Compilers,*

Architecture, and Synthesis for Embedded Systems, CASES '09, (New York, NY, USA), p. 281–290, Association for Computing Machinery, 2009.

[115] H. Thapliyal, T. S. S. Varun, E. Munoz-Coreas, K. A. Britt, and T. S. Humble, "Quantum circuit designs of integer division optimizing t-count and t-depth," in *2017 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, pp. 123–128, 2017.

[116] S. V. Dibbo, H. M. H. Babu, and L. Jamal, "An efficient design technique of a quantum divider circuit," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2102–2105, 2016.

[117] D. R. I. M. Setiadi, "Psnr vs ssim: imperceptibility quality assessment for image steganography," *Multimedia Tools Appl.*, vol. 80, p. 8423–8444, Mar. 2021.

[118] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Pixel division." Online. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/pixdiv.htm, Accessed: Mar. 10, 2025.

[119] A. P. Dhawan, G. Buelloni, and R. Gordon, "Enhancement of mammographic features by optimal adaptive neighborhood image processing," *IEEE Transactions on Medical Imaging*, vol. 5, no. 1, pp. 8–15, 1986.

[120] Kaggle, "Cbis-ddsm: Breast cancer image dataset." Online. https://www.kaggle.com/datasets/awsaf49/cbis-ddsm-breast-cancer-image-dataset, [Accessed: Jan. 15, 2024].

[121] Kaggle, "Social network ads." Online. Available: https://www.kaggle.com/datasets/rakeshrau/social-network-ads, [Accessed: Jan. 15, 2024].

[122] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

[123] J. A. Montañez Barrera, M. R. von Spakovsky, C. E. Damian Ascencio, and S. Cano-Andrade, "Decoherence predictions in a superconducting quantum processor using the steepest-entropy-ascent quantum thermodynamics framework," *Phys. Rev. A*, vol. 106, p. 032426, Sep 2022.