Visual and Interactive Tools in Support of a Hands-on Introduction to Quantum Computing

by

Samantha Norrie

B.Sc., University of Victoria, Canada, 2023

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

We acknowledge and respect the Lək̓ʷəŋən (Songhees and Xʷsepsəm / Esquimalt) Peoples on whose territory the university stands, and the Lək̓ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Visual and Interactive Tools in Support of a Hands-on Introduction to Quantum Computing

by

Samantha Norrie

B.Sc., University of Victoria, Canada, 2023

Supervisory Committee:

Dr. Ulrike Stege, Supervisor
Department of Computer Science, University of Victoria

Dr. A. Estey, Supervisor
Department of Computer Science, University of Victoria

ABSTRACT

The rapid growth of quantum computing technology has created a need for educational tools that help foster intuitive understanding of quantum concepts. Although exploratory learning has demonstrated effectiveness in other science, technology, engineering, and mathematics domains, its role in quantum computing education remains underexplored. This thesis presents the design, implementation, and evaluation of two exploratory learning tools, QNotation and QGrover, which were developed to support introductory quantum computing instruction.

QNotation is a browser-based application that displays quantum circuits written using quantum software libraries (such as `Qiskit`, `PennyLane`, or `Cirq`) in circuit, Dirac, and matrix notation. To ensure reliability and conceptual accuracy, QNotation incorporates a comprehensive automated testing and verification framework, including integration tests executed through continuous integration workflows. We conducted an exploratory study to examine how students interact with QNotation. Preliminary survey data and usage observations provide initial insights into learner engagement patterns within exploratory quantum computing environments. QGrover, another browser-based application, complements QNotation by allowing users to interact with Grover's algorithm through dynamic visualizations. Users can configure the number of qubits, marked states, and iterations (or select the optimal count) used by the algorithm. This enables them to observe how the circuit and state vectors evolve step-by-step. By supporting exploratory manipulation, QGrover helps learners develop intuition about the algorithm's dependence on its problem parameters.

The contributions of this work is an overview of how the aforementioned tools have been designed according to software engineering best practices to align with modern industry needs. Additionally, we demonstrate that well-designed exploratory tools can meaningfully improve the teaching and learning of quantum computing. We also provide example material to enable these tools to be easily integrated into learning exercises and assessments.

Table of Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Figures vii

List of Tables ix

Acknowledgments x

Chapter 1. Introduction 1
- **1.1 Background and Context** . . . 1
- **1.2 Problem Statement** . . . 2
- **1.3 Research Questions** . . . 2
- **1.4 Thesis Structure** . . . 3

Chapter 2. Literature Review and Background 4
- **2.1 Quantum Computing Courses and Curricula** . . . 4
- **2.2 Learning Tools** . . . 8
  - 2.2.1 Traits of Successful Learning Tools . . . 8
  - 2.2.2 Learning Tools in Classical Computer Science . . . 9
  - 2.2.3 Technologies and Learning Tools for Quantum Computing . . . 10

Chapter 3. QNotation 11
- **3.1 Background** . . . 12
  - 3.1.1 Notations in Quantum Computing . . . 12
  - 3.1.2 Endianness in Quantum Computing . . . 12
  - 3.1.3 Programming in Quantum Computing . . . 12
- **3.2 Motivation** . . . 14
- **3.3 How to Use QNotation** . . . 14
- **3.4 Implementation** . . . 16
  - 3.4.1 Initial Prototyping . . . 16
  - 3.4.2 Backend . . . 16
  - 3.4.3 Frontend . . . 22
  - 3.4.4 Gate Support . . . 22
- **3.5 Additional Library Implementation** . . . 23

3.6 Related Learning Exercises and Assessments . . . 23

Chapter 4. Verification, Testing, and Reproducibility in QNotation **27**
4.1 Verifying Correctness . . . 28
4.1.1 Correctness of Library Integration . . . 30
4.2 Preventing Regressions . . . 30
4.3 Increasing Reliability . . . 30

Chapter 5. QNotation Pilot User Study and Feedback **32**
5.1 Pilot User Study . . . 32
5.1.1 Study Background . . . 32
5.1.2 Pilot Study Anecdotes . . . 33
5.2 Feedback . . . 35

Chapter 6. QGrover **36**
6.1 Grover's Algorithm . . . 36
6.2 How To Use QGrover . . . 39
6.3 Implementation . . . 41
6.3.1 Backend . . . 41
6.3.2 Frontend . . . 45
6.4 Related Learning Exercises and Assessments . . . 45

Chapter 7. Discussion **49**
7.1 Analysis of our Research Questions . . . 49
7.2 Implications . . . 54
7.3 Limitations . . . 55
7.3.1 QNotation . . . 55
7.3.2 QNotation CI/CD Pipeline and Test Suites . . . 55
7.3.3 QNotation Study . . . 55
7.4 Discussion Summary . . . 56

Chapter 8. Conclusion and Future Work **57**
8.1 Contributions . . . 57
8.2 Future Work . . . 59
8.2.1 Quantum Algorithms Repository . . . 59
8.2.2 QNotation . . . 60
8.2.3 QNotation User Study . . . 62

Bibliography **64**

Appendix A. Terms **77**

Appendix B. Qiskit (2.2.3) gate coverage within QNotation **79**

Appendix C. PennyLane (0.42.3) gate coverage within QNotation **82**

Appendix D. Cirq (1.6.1) gate coverage within QNotation **84**

Appendix E. QNotation Tests **86**

Appendix F. QNotation Study: Start-of-Term Questionnaire **90**

Appendix G. QNotation Study: During Term Questionnaire **93**

Appendix H. QNotation Study: End-of-Term Questionnaire **94**

List of Figures

Figure 3.1 A. Circuit, B. Dirac, and C. matrix notation representations of the first Bell state [108]. 11
Figure 3.2 In little-endian ordering, the leftmost bit of a binary number is the least significant bit and the rightmost bit is the most significant. This is flipped for big-endian ordering. 13
Figure 3.3 QNotation without an inputted circuit. 15
Figure 3.4 QNotation with input and output. 15
Figure 3.5 QNotation as a prototype Jupyter Notebook widget. The visualization part of the image is from our extended abstract *QNotation: An Interactive Visual Tool to Lower Learning Barriers in Quantum Computing* [79] 17
Figure 3.6 The steps taken on QNotation's backend once code is submitted from the frontend. 18
Figure 3.7 The `convert_code_string_to_circuit_object()` abstract method within the `Parser` ABC. 19
Figure 3.8 The `convert_code_string_to_circuit_object()` method within `PennylaneParser`. The method formats the code string received from the frontend so that details on its quantum circuit can be extracted by the `convert_code_string_to_circuit_object()` abstract method defined in the `Parser` ABC superclass. 19
Figure 3.9 QNotation's parser classes as an XML Diagram. When `get_parser()` is called, a `Qiskit`, `PennyLane`, or `Cirq` parser is created depending on the argument given (`qc_type`). Some parsers have extra behaviour integrated into their implementations of the methods defined by `Parser`. An example of this is `convert_code_string_to_circuit_object()`, shown in Figure 3.7 and Figure 3.8. 20
Figure 3.10 The one-qubit solution for **B.Q.1**. 25
Figure 3.11 The three-qubit solution for **B.Q.1** Users can scroll vertically in the matrix notation component to see the full matrix. 25

Figure 4.1 The `TestQNotation` ABC 29

Figure 6.1 Analogy demonstrating the application of two iterations of the Grover iterator. The piece of candy in the upper right corner of each iteration is the desired piece. 38
Figure 6.2 The last slide of QGrover's modal. 40
Figure 6.3 The `create_oracle` algorithm. 42

Figure 6.4 The `create_diffuser` algorithm. 42
Figure 6.5 QGrover's input bars. 43
Figure 6.6 QGrover's input bars with error messages. 44
Figure 6.7 The application of a phase oracle to a set of size $N = 4$, with number of qubits $n = 2$, and with $M = 1$ where 1 is the desired value. Each value initially has an amplitude of $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. The oracle flips the phase of the desired value. This is done by multiplying the amplitude of the desired value by -1. In this example, the new amplitude value is $-\frac{1}{2}$ (as $-1 \times \frac{1}{2} = -\frac{1}{2}$). 46
Figure 6.8 The application of a diffuser to a set of size $N = 4$. The equation used to calculate the new amplitudes is $\alpha'_x = 2a - \alpha_x$. When the diffuser is applied for the first time, $a = \frac{N-2}{N\sqrt{N}}$. When doing the calculations for this example specifically, one should infer that this is the first application of the of the diffuser due to the magnitudes of the amplitudes all being equal. Therefore, for the desired value, $a = \frac{4-2}{4\sqrt{4}} = \frac{1}{4}$ and the amplitude of the term of wanted element is $2 \cdot \frac{1}{4} - (-\frac{1}{2}) = 1$, while all other amplitudes equal $2 \cdot \frac{1}{4} - \frac{1}{2} = 0$. 46

Figure 7.1 QNotation's custom error classes. The constants used are defined outside this code snippet 52

Figure 8.1 Conversion between `Qiskit`, `OpenQasm 3`, and `PennyLane`. 61
Figure 8.2 Equivalent `PennyLane` circuits where A shows a `PauliRot` gate being applied, and B shows an equivalent operation. `PauliRot` is not currently supported in QNotation, but `Hadamard` and `RZ` are supported. . 61

Figure F.1 The image for question 5 92

List of Tables

Table 2.1 Undergraduate computer science curricula in Canada with an optional quantum computing course. The institutions listed are all from Times Higher Education's list of top Canadian universities in 2026 [41]. While all courses listed can be taken by computer science students, the courses vary in their coverage of quantum computing concepts. [99, 142, 25, 21, 133, 135, 134, 136, 26, 137, 138, 139, 27, 141, 145, 146, 144, 67, 147, 34, 33, 83, 82, 86, 84, 85, 87, 92, 93, 91, 98, 97, 95, 100, 102, 103, 149, 148, 152, 151, 35, 90, 89, 104, 140, 94, 132, 143, 150, 88]. 7

Table 4.1 Gate Coverage within QNotation 28

Table 5.1 Academic backgrounds of Participant A and Participant B 34
Table 5.2 Participant A and Participant B's answers to Survey One 34
Table 5.3 Participant A's answers to Survey Two and Survey Three. 35
Table 5.4 The topics being taught during the weeks the surveys were answered . 35

Table A.1 Quantum terms used in this thesis. We refer readers to the textbooks *Quantum Computation and Quantum Information* [78] and *Mathematical Foundations of Quantum Computing: A Scaffolding Approach* for further details [68]. 78

Table B.1 Qiskit gate support details 79

Table C.1 PennyLane gate support details 82

Table D.1 Cirq gate support details 85

Table E.1 Coverage of QNotation tests across libraries 86

Acknowledgments

I would first like to express my heartfelt gratitude to my supervisors, Dr. Ulrike Stege and Dr. Anthony Estey, for their guidance, encouragement, and unwavering support throughout this thesis. Their insight and patience have been invaluable, and I am grateful for the opportunities they have given me to grow as a researcher and as a person.

My sincere thanks go to my collaborators and colleagues in the Rigi-Pita Lab. In particular, I would like to acknowledge Addie Jordon, who invited me to join their research project while I was still an undergraduate student. That opportunity ultimately led me to join the lab as a graduate student, and I am deeply appreciative of the mentorship and support I received along the way.

I am grateful to the NSERC CREATE program for supporting this research and providing an environment where I could pursue this work. I am also grateful for everyone I was able to meet through the program.

To my friends, Danielle Torres, George Wang, and Hollie Krutz, thank you for all the support you have given me over the last several years. I am grateful for our hikes, adventures, and endless 'yaps', and I cannot wait for many more.

I would like to express appreciation to my partner, José Manuel Ossorio, for all his emotional support and for keeping our home *hygge* while I wrote this thesis.

Finally, I would like to express my deepest gratitude to my parents, Keri Norrie and David Ellis, for their endless support and belief in me. I could not have asked for better parents.

Chapter 1

Introduction

1.1. Background and Context

Since the first quantum computer was made publicly accessible on the cloud in 2016, the field of quantum computing has gained significant momentum [5]. What was once studied in only a few research labs is now accessible to nearly anyone. Today, anyone with an internet connection can write and execute quantum programs using platforms such as `IBM Quantum` [54], `Amazon Braket` [15], or `Strawberry Fields` by Xanadu [164].

Quantum computing combines quantum mechanics with computer science to create a computational paradigm capable of solving problems beyond the reach of classical computers. Instead of using bits, quantum computers use qubits as their unit of information. Qubits can exist in superposition, meaning each qubit can each represent a linear combination 0 and 1, and they can also become entangled, allowing their states to be correlated across even spatially separated qubits. These phenomena help enable quantum algorithms to solve problems like search and factorization quadratically or even exponentially faster than classical methods.

Despite its promise, quantum computing remains conceptually challenging to newcomers, likely because its behaviour departs sharply from intuition about classical computing. Learners must understand both abstract mathematical notations (Dirac notation, matrix algebra, and circuit diagrams) and mathematical reasoning (such as superposition, and entanglement). We point the reader to Table A.1 in the appendix for referencing quantum terms and their definitions.

We developed two tools to address these challenges: QNotation and QGrover. QNotation supports students in navigating between the aforementioned notations [81], and as such supports the development of the important mathematical and conceptual literacy of quantum circuits. QGrover introduces learners to Grover's algorithm [80], a quantum algorithm that solves the unstructured search problem quantumly, and asymptotically faster than classically possible [50]. Together, these tools aim to reduce the cognitive barrier to entry by providing visual, interactive, and exploratory ways to understand some basic quantum computing concepts.

1.2. Problem Statement

As quantum computing continues to expand from the theoretical field of quantum information science to an applied discipline, there remains a pressing need for educational approaches that make its foundational concepts accessible to a diverse audience. While some learning tools are available [53, 162, 69], not many exist due to how relatively young the field of quantum computing is.

While exploratory learning has been proven to enhance engagement in other STEM fields, little is known about student interactions and learning in exploratory quantum computing [113]. This lack of empirical understanding hinders educators and researchers from designing effective interventions to promote quantum literacy.

In this thesis, we address these gaps by

1. Presenting two exploratory learning tools, QNotation and QGrover, both of which allow users to explore introductory quantum computing concepts
2. Detailing how QNotation and QGrover can be integrated into learning exercises and assessments
3. Exploring the use of QNotation by students
4. Validating QNotation's software engineering robustness
5. Demonstrating how QNotation and QGrover currently meet and can be expanded to grow with industry needs

1.3. Research Questions

In this thesis we address five research questions. The first three ensure that our contributions are impactful from an educational point of view:

1. How can exploratory learning tools be designed to support the learning of foundational concepts in quantum computing education?
2. How can exploratory tools be effectively integrated into learning exercises and assessments?
3. How do learners and educators engage with and learn from such tools in practice?

The remaining two questions address the software engineering solidity of the tools we created:

4. How can software engineering principles be incorporated into QNotation and QGrover?
5. How can QNotation and QGrover be built to align with industry needs?

The objectives and questions outlined above frame the design, development, testing, and evaluation of our tools. Together, these tools embody the exploratory learning approach central to our research, which aim to emphasize active engagement and discovery-based understanding.

1.4. Thesis Structure

In the remaining chapters, we describe the design, implementation, testing and evaluation of two tools, QNotation and QGrover, before discussing their implications for teaching and learning of quantum computing.

Chapter 2 presents a literature review on the quantum computing education space, the traits of strong learning tools, as well as learning tools currently available for supporting the learning of quantum computing and classical computer science.

Chapter 3 describes QNotation, an exploratory browser-based learning tool for three different notations used to describe quantum circuits and quantum states: circuit, Dirac, and matrix notation. The chapter describes the three notations, the programming libraries used as input, how QNotation can be used, how it was designed and developed, and how it can be extended.

Chapter 4 outlines QNotation's testing processes and CI/CD pipeline.

Chapter 5 describes a pilot user study on the use of QNotation in an introductory quantum computing learning environment .

Chapter 6 describes QGrover, an exploratory browser-based learning tool for visualizing Grover's algorithm. The chapter details how Grover's algorithm works, how QGrover can be used, and how it was developed.

Chapter 7 discusses how the work presented in the previous chapters answers the research questions highlighted in Chapter 1.

Chapter 8 summarizes the work presented in the thesis and highlights future work.

Chapter 2

Literature Review and Background

In this chapter we review relevant work in quantum computing education, the design of learning tools for STEM education, and the broader landscape of quantum computing courses and technologies. The goal of this chapter is to situate the tools QNotation and QGrover, described in Chapter 3 and Chapter 6 respectively, within current educational and technological trends in both classical and quantum computing. Section 2.1 surveys the availability and structure of quantum computing courses across Canadian post-secondary institutions. Subsequent sections discuss characteristics of effective learning tools, review existing learning tools from classical computer science education, and highlight current quantum computing learning technologies and visualization platforms. Together, these findings reveal a need for browser-based, multi-notation tools that provide immediate, elaborated feedback while aligning with mainstream libraries, which is the niche QNotation and QGrover target.

2.1. Quantum Computing Courses and Curricula

Despite being a relatively new field, many post-secondary institutions are starting to offer quantum computing courses. Table 2.1 shows which Canadian post-secondary institutions offer at least one course at the undergraduate level in quantum computing and which year it is typically offered. Normally, an undergraduate quantum computing course contains the following five components [19, 66]:

1. Foundational knowledge:
    - Math: Some linear algebra concepts including matrix and vector math
    - Computer Science: Basic programming skills, typically in Python
    - Physics: Basic wave mechanics[1], Dirac notation
2. Quantum protocols: these may include
    - Superdense coding

[1] Physics pre-requisites may not be present for courses offered by departments like computer science or mathematics. For example, the *Quantum Algorithms and Software Engineering* course offered at by the University of Victoria within their computer science department [101] does not require physics knowledge as a pre-requisite.

- BB84

3. Quantum algorithms: these may include
   - Quantum teleportation
   - Grover's algorithm
   - Shor's algorithm
   - Quantum Fourier Transform
   - Quantum Phase Estimation
   - Variational Quantum algorithms
     - Quantum Approximate Optimization Algorithm
     - Variational Quantum Eigensolver
4. Quantum programming: these may include the teaching of popular quantum libraries/languages such as `Qiskit`, `PennyLane`, `Cirq`, or `Q#`.
5. Topics and applications in quantum computing: these may include
   - Quantum algorithms for chemistry
   - Quantum software engineering
   - Quantum annealing
   - Quantum machine learning
   - Quantum and hybrid algorithms for optimization problem

Foundational knowledge required for quantum computing is also often covered in industry-offered courses, even when those courses are highly specialized. Some examples of this include D-Wave's courses, which focus primarily on solving optimization problems on their quantum annealers [32], and IBM Quantum Learning's courses, which tend to focus on writing quantum code using `Qiskit`, their quantum library, but also review quantum foundations [55]. Several institutions have started offering publicly available quantum content. University of Victoria (UVic) and the Quantum Algorithms Institute (QAI) offer jointly a synchronous and asynchronous quantum software engineering bootcamp for those with some STEM or programming experience [53]. UVic, in collaboration with Honeywell, Quantinuum or Institute Quantique (l'Université de Sherbrooke), offered quantum computing workshops for highschoolers at IEEE Quantum Week in four consecutive year [3, 13, 12, 11], the now largest quantum computing conference worldwide [59]. The experiences with designing and offering such workshops is also described in [14]. Institute Quantique's Quantum Enigmas consist of several videos on quantum phenomena and algorithms designed to help those getting into the field understand quantum concepts at a high level. These videos include mentions of content from IBM Quantum to help viewers understand quantum foundations [69]. The University of British Columbia offers workshops and resources for teaching and learning about quantum computing within their Diversifying

Talent in Quantum Computing hub, which is primarily designed for K-12 students [153]. The Institute for Quantum Computing at the University of Waterloo offers Quantum 101, an asynchronous course for those with some foundational STEM knowledge that covers the foundations of quantum computing and quantum mechanics [158]. Similar to Quantum Enigmas, Quantum 101 explains quantum concepts at a high level while introducing learners to terms used in the field.

Table 2.1.: Undergraduate computer science curricula in Canada with an optional quantum computing course. The institutions listed are all from Times Higher Education's list of top Canadian universities in 2026 [41]. While all courses listed can be taken by computer science students, the courses vary in their coverage of quantum computing concepts. [99, 142, 25, 21, 133, 135, 134, 136, 26, 137, 138, 139, 27, 141, 145, 146, 144, 67, 147, 34, 33, 83, 82, 86, 84, 85, 87, 92, 93, 91, 98, 97, 95, 100, 102, 103, 149, 148, 152, 151, 35, 90, 89, 104, 140, 94, 132, 143, 150, 88].

Curriculum	Department	Year of QC Course
University of Toronto	Physics	Year 3
McGill University	–	–
University of British Columbia (Vancouver)	Computer Engineering	Year 4
McMaster University	Math	Year 3
University of Alberta	Computer Science	Year 4
Université de Montréal	Computer Science	Year 3
University of Waterloo	Computer Science and Software Engineering	Year 4
University of Ottawa	Math	Year 4
University of Calgary	–	–
Western University	–	–
Queen's University	–	–
Simon Fraser University	Mathematics and Computer Science	Year 4
University of Victoria	Computer Science	Year 4
Dalhousie University	Mathematics and Statistics	Year 4
University of Saskatchewan	–	–
Université Laval	–	–
University of Guelph	Nanoscience	Year 4
York University	Electrical Engineering and Computer Science	Year 4
Carleton University	Computer Science	Year 4
Memorial University of Newfoundland	Physics	Year 4
Université du Québec	–	–
University of Manitoba	–	–
University of Windsor	–	–
Toronto Metropolitan University	Computer Science	Year 4
University of New Brunswick	Physics	Year 2
Ontario Tech University	Computer Science	Year 4
University of Regina	Mathematics	Year 4
Lakehead University	Physics	Year 3
Brock University	–	–
St. Francis Xavier University	–	–
University of Northern British Columbia	–	–
Wilfrid Laurier University	Computer Science	Year 3
University of Lethbridge	–	–

2.2. Learning Tools

In this section we describe traits of successful learning tools, learning tools for classical computer science, and learning tools for quantum computing.

2.2.1. Traits of Successful Learning Tools

This sub-section highlights the following traits of successful learning tools [23, 70, 38]:

- Documentation
- Feedback
- Ease of accessibility
- Interactivity

2.2.1.1. Documentation

Documentation refers to the written and example-based resources that explain how a software tool works, how it should be used, and what functionality it provides. Effective documentation lowers the barrier to entry for new users, supports self-directed learning, and enables users to explore a system independently without external instruction.

The 2024 Quantum Open Source Survey by the Unitary Foundation revealed that 52.2% of students cited poor documentation as the main reason they were not using technologies they were interested in [131]. In their study on software documentation, *A Field Study of Developer Documentation Format*, Nassif and Robillard found that the majority of their participants liked documentation that includes code examples [77]. Rieman, in *A Field Study of Exploratory Learning Strategies*, reports that participants liked to refer to an application's documentation when first using it [121].

2.2.1.2. Feedback

Feedback describes the information a learning tool provides to users in response to their actions, with the goal of supporting understanding, correcting misconceptions, and guiding progress. In interactive learning environments, feedback plays a central role in helping learners interpret outcomes and refine their mental models.

In their paper, *Defining the Learner Feedback Experience*, Crisp and Bonk highlight how receiving feedback from a tool is crucial to the user's learning experience [31]. They also define the six dimensions of feedback, which are timeliness, frequency, distribution, source, individualization, and content of the feedback.

2.2.1.3. Ease of Accessibility

Ease of accessibility refers to how easily learners can access, deploy, and begin using a learning tool, including considerations such as cost, setup complexity, platform requirements, and usability. Tools that are difficult to access or require extensive configuration can discourage adoption, regardless of their educational value.

In their literature review, *A Systematic Literature Review on Software Applications Used to Support Curriculum Development and Delivery in Primary and Secondary Education*, Agbo et al. highlight that deployment cost and ease of use greatly impact whether learners will adopt a learning tool [7]. While creating Learning Moment, a learning tool designed to help medical students document their learning during clinical rotations, Chu et al. found ease of accessibility to be an important tool-adoption trait [22].

2.2.1.4. Interactivity

Interactivity refers to the degree to which a learning tool allows users to actively engage with content through manipulation, exploration, and immediate system response. Interactive tools support learning by encouraging experimentation and enabling learners to observe the effects of their actions in real time.

Shappee, in Characteristics to Consider When Evaluating Educational Technology Tools, identifies interactivity as a key characteristic of effective learning tools [126]. Similarly, Martín-Sómer et al. report that post-secondary students place a high value on interactivity when engaging with educational technologies, as shown in their study Utilising Interactive Applications as Educational Tools in Higher Education [72].

2.2.2. Learning Tools in Classical Computer Science

Recently, browser-based code editors have gained popularity in introductory computer science courses [119]. This is because they remove the need to set up a local development environment on the learner's computer. `Codepen`, `CodeSandbox`, and `JSFiddle` are popular browser-based editors [1, 2, 4].

Many topic-specific browser-based learning tools exist. For example, the Data Structure Visualizations repository from the University of San Francisco contains several interactive algorithm visualizations, including visualizations for red/black trees and breadth-first search [156, 157, 155]. The National University of Singapore also has a similar repository titled VisuAlgo [96]. One of the highlighted perks of VisuAlgo is the ability to use sample inputs and custom inputs for its visualizations. AlgoMonster helps student practice algorithmic interview questions through the use of interactive flowcharts [9].

2.2.3. Technologies and Learning Tools for Quantum Computing

There are several quantum computing software libraries, including `Qiskit` [56], `PennyLane` [110], `Cirq` [48], `PyQuil` [123], and `QuTip` [118]. Several are complemented by interactive learning tools and courses. For `Qiskit`, these include online courses through the `IBM Quantum Platform` and `Qiskit`'s exploratory learning tool Composer [55, 53]; for `PennyLane`, this includes the Codebook [162]; for `Cirq`, this includes Google Quantum AI's Coursera courses that use `Cirq` and `Cirq`'s documentation [49]; for `PyQuil`, this includes educational exercises using `PyQuil` [122]; and for `QuTip`, this includes a repository of Jupyter notebook tutorials [117]. These resources demonstrate how the topics highlighted in Section 2.1 can be used within the libraries in addition to their library-specific strengths.

Over the last decade, more browser-based visualization tools have been made to demonstrate how different quantum phenomena operate. QWalkvis is a web application designed to help users visualize the steps taken in a quantum walk based on different parameters [65]. Similar to QWalkvis, the QML playground application lets users visualize different quantum machine learning models through the modification of different parameters [36].

The Quantum Programming Studio by Rigetti Computing is a browser-based IDE that allows for quantum circuits to be visualized and simulated [124]. The Quantum User Interface by the University of Melbourne visualizes the state of a quantum circuit (made up of drag-and-dropped gates) at each of its steps [154]. Google's Quantum Computing Playground contains its own scripting language and allows for 3D visualizations of quantum states [161].

Multiple visualization libraries are available for quantum computing as well. `Qiskit` contains visualizations, a sub-module of the library, that supports `matplotlib` visualizations, Bloch sphere, and Qsphere visualizations [58]. `Cirq` also supports `matplotlib` visualizations [48]. Classiq's Quantum Program Visualization Tool creates drag-and-droppable circuits [24]. `QMSolve` is a `Python` library that visualizes Schrödinger's equation [6]. Microsoft's `Quantum Development Kit` provides circuit visualization support [75].

Throughout this chapter, we highlighted the need for robust learning tools for supporting the teaching and learning of introductory quantum computing concepts. Throughout Chapters 3- 6, we detail the design, implementation, and evaluation of QNotation and QGrover. While developing QNotation and QGrover, we made sure to prioritize documentation, feedback, ease of accessibility, and interactivity to help ensure that they would become strong learning tools. Taken together, these features suggest that exploratory learning tools for quantum computing must combine strong documentation, immediate and rich feedback, accessible deployment, and interactive visualizations to effectively support students with diverse prior backgrounds.

Chapter 3

QNotation

QNotation, a browser-based application we designed and developed to support users in building intuition across the three primary notations used to describe quantum states and quantum algorithms: circuit, Dirac, and matrix notation [81]. By allowing users to construct quantum circuits in the library of their choice and instantly visualize them in all three notations, QNotation bridges the gap between symbolic and computational representations of quantum information. Figure 3.1 illustrates a Bell state written in these notations. The goal of QNotation is to support the development of literacy in representing quantum information while engaging directly with widely used quantum libraries. We refer the reader to our paper, *QNotation: A Visual Browser-Based Notation Translator for Learning Quantum Computing*, for more information on an earlier version of QNotation [81].

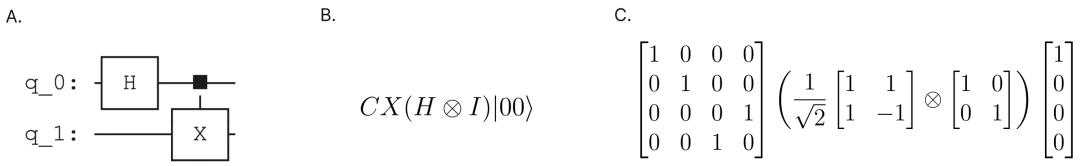


Figure 3.1.|: A. Circuit, B. Dirac, and C. matrix notation representations of the first Bell state [108].

3.1. Background

The following sub-sections detail the core concepts covered by QNotation:

3.1.1. Notations in Quantum Computing

The three main notations used to describe quantum states and quantum algorithms are circuit, Dirac, and matrix notation. Converting between the three is often covered in introductory quantum computing courses [19]; being able to do so is thought to improve quantum literacy and intuition [45, 43]. Circuit notation tends to be the most familiar to those coming from a software background as it resembles classical gate notation. Figure 3.1 A. shows an example of circuit notation. The qubit in the system is listed on the left, and each gate that is applied to them is depicted on a horizontal wire. Circuit notation is interpreted from left to right. It is assumed, unless explicitly shown otherwise, that each qubit is initialized with a $|0\rangle$.

Dirac notation, shown in Figure 3.1 B., is interpreted from right to left and represents quantum operations as their acronyms and quantum states as *kets*. Quantum operations are used to express a singular quantum gate as a tensor product or matrix product. Dirac notation is considered the most compact of the three notations. Given the two-qubit circuit in our example, the controlled not two-qubit gate, CX, is multiplied with the tensor product of two single qubit gates, $H \otimes I$.

Matrix notation, Figure 3.1 C., represents quantum operations as matrices and quantum states as vectors. It is considered to be the most verbose of the three notations. Like with Dirac notation, matrices used within the same quantum operation are combined using matrix multiplication and tensor products. Matrix notation is interpreted from right to left.

3.1.2. Endianness in Quantum Computing

One of QNotation's most prominent features is its ability to display its visualizations in big-endian and little-endian ordering [44, 114]. Endianness describes the ordering of bits in a system. When little-endian ordering is used, the leftmost bit is the least significant bit, and the rightmost bit is the most significant bit. The opposite is done for big-endian ordering. An example of the differences between the two orderings can be seen in Figure 3.2.

It is important to be able to read and differentiate between these two orderings because they are both used in quantum computing. For example, while `Qiskit` uses little-endian ordering, most textbooks in quantum computing use big-endian ordering [114]. `PennyLane` and `Cirq` also use big-endian ordering.

3.1.3. Programming in Quantum Computing

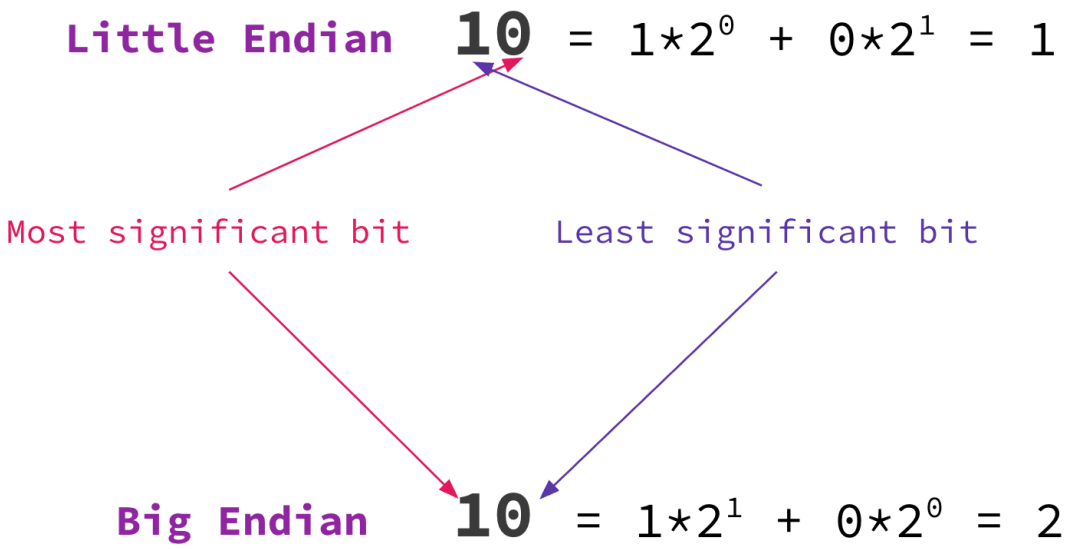


Figure 3.2.|: In little-endian ordering, the leftmost bit of a binary number is the least significant bit and the rightmost bit is the most significant. This is flipped for big-endian ordering.

3.2. Motivation

Although quantum foundations are crucial for other topics, we are not aware of any existing tool translates between the three notations as effectively as QNotation. QNotation takes inspiration from IBM Quantum Learning's Composer, which converts input from drag-and-drop circuits, `Qiskit` code, or `OpenQASM 3` code into QSphere and bar chart visualizations [53, 29]. In Gire and Price's paper, *Structural features of algebraic quantum notations*, the authors survey students on their use of the notations and found that they struggle to convert between them [45]. Ferryansyah et al.'s report that students most commonly struggle with notation and converting it [43]. In Tunde and Thacker's paper, *Investigating Students' Strengths and Difficulties in Quantum Computing*, the authors report the qualitative results of their study of physics students' strengths and difficulties in an introductory quantum computing course [66]. While the students, who had previously taken a linear algebra course, did not experience difficulties with matrix operations, they struggled with Dirac notation as well as converting between the different notations. Hu et al. report that students tend to find Dirac notation to be difficult to understand [51]. QNotation also helps learners practice some of the foundations of quantum programming, mainly the creation of quantum circuits in each of its supported libraries. This is crucial as creating quantum circuits is one of the most common quantum programming tasks. In Simões and Queirós paper *On the Nature of Programming Exercises*, the authors highlight that the *code skeleton* approach to programming exercises (where students are given some code, like a function stub, and input their answers) is effective for teaching programming. This is the formatting style we adopt in QNotation.

3.3. How to Use QNotation

This section gives a step-by-step guide of how to use QNotation referencing Figure 3.3 and Figure 3.4.

1. Select a quantum library that will be used to input a quantum circuit into QNotation (`Qiskit`, `PennyLane`, or `Cirq`).
2. Enter code under 'Enter code below' comment(s).
    a) Instead of entering code in manually, example code can be selected. The following examples can be generated for each library:
        - two-qubit Bell state
        - two-qubit Grover's algorithm
        - three-qubit hermitian gates
        - three-qubit quantum teleportation
3. Click 'Run' to generate notation visualizations

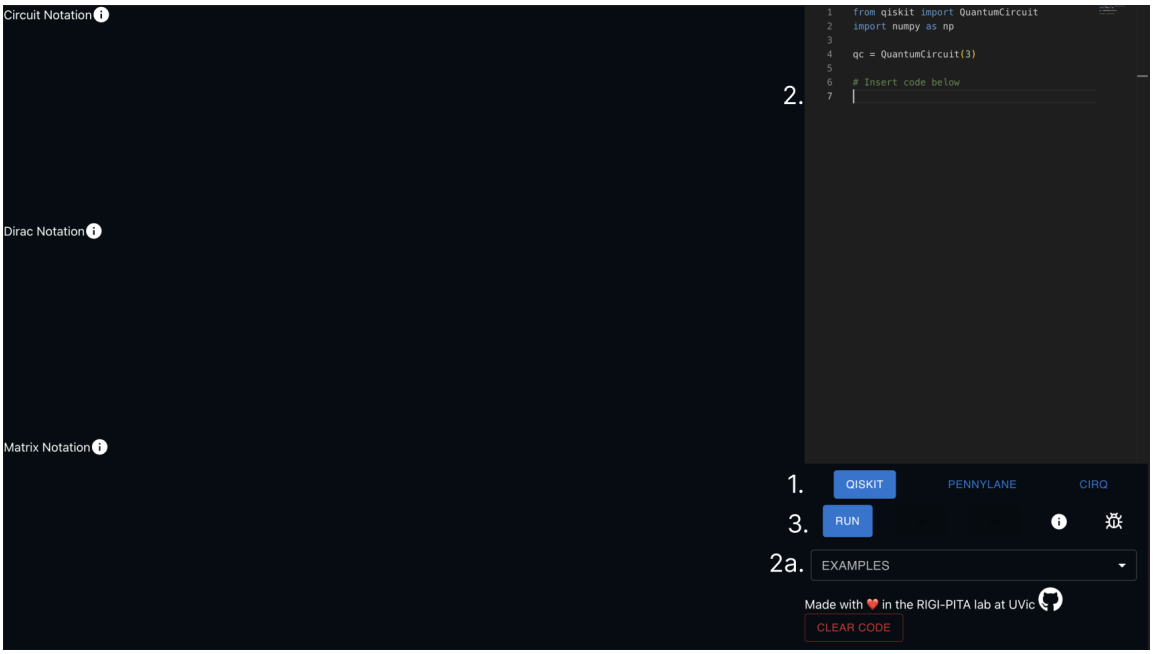


Figure 3.3.|: QNotation without an inputted circuit.

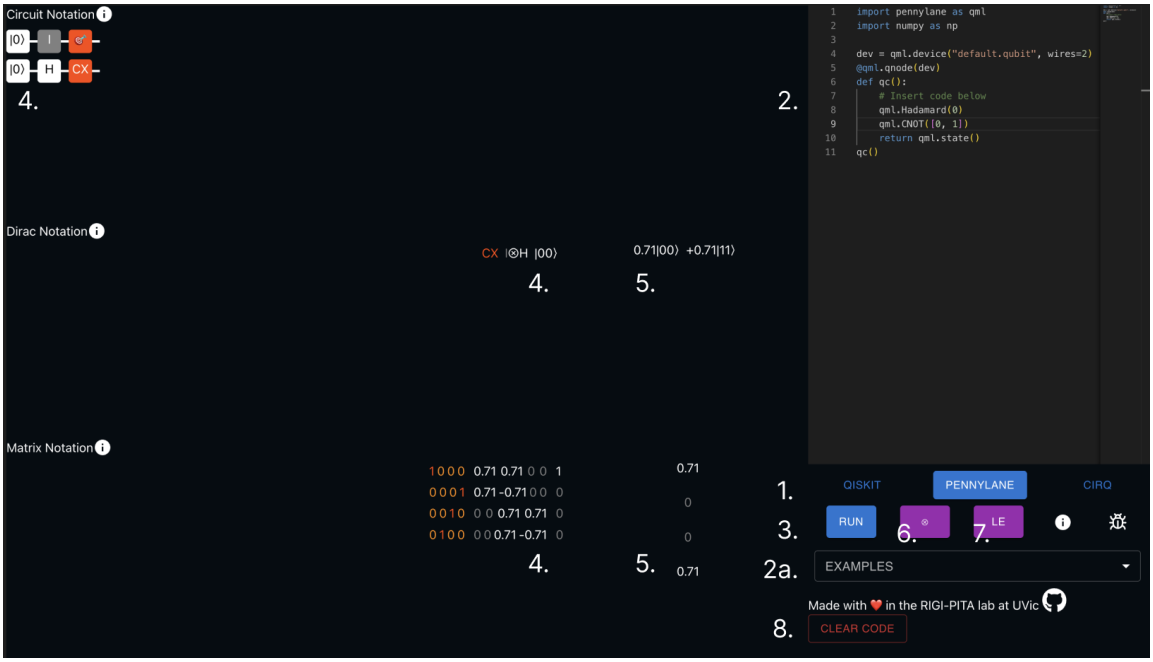


Figure 3.4.|: QNotation with input and output.

4. Click on operation components of notation visualizations to see their respective equivalent highlighted in the other notation visualizations
5. View the quantum states at the clicked upon component on the right of the Dirac and matrix visualizations. The quantum state shown is the state of the quantum system after the clicked upon operation has been applied.
6. Toggle between little-endian and big-endian representations.
7. Toggle between having the matrix notation visualization display each operation as a single matrix or as a tensor product. This option is available for quantum circuits with three qubits or less.
8. Clear code before generating more visualizations.

3.4. Implementation

The following subsections detail QNotation's implementation. The backend and frontend repositories for the application can be found on GitHub.

3.4.1. Initial Prototyping

As a first prototype, we implemented QNotation as a Jupyter Notebook widget [79]. This decision was made to ensure the tool is easily accessible within Jupyter Notebooks, which are commonly used for teaching foundational concepts. Figure 3.5 shows the prototype. While the prototype works for quantum circuits with up to three qubits, it could not support larger quantum circuits due to UI limitations.

We decided to implement the final version as a browser-based application, reducing UI limitations while also making the tool operating system agnostic.

3.4.2. Backend

The QNotation backend was created using `Python`, `Flask`, `Qiskit`, `PennyLane`, and `Cirq`. The implementation of `Qiskit`, `PennyLane`, and `Cirq` was decided due to them being popular and flexible quantum computing libraries [131]. Noiseless simulators are used to ensure that the visualizations shown are mathematically correct. Noiseless simulators run small pieces of quantum code on a classical computers, ensuring that no errors occur during its run. Noise typically refers to errors caused by disturbances that impact quantum systems [116]. Th simulators used in QNotation are `Aer` [52], `default.qubit` [106], and `Simulator` [47] for `Qiskit`, `PennyLane`, and `Cirq` respectively. The backend uses the factory method to process input received in any one of the supported libraries in a consistent manner [127]. Figure 3.9 shows an XML diagram of QNotation's factory classes. Figure 3.7

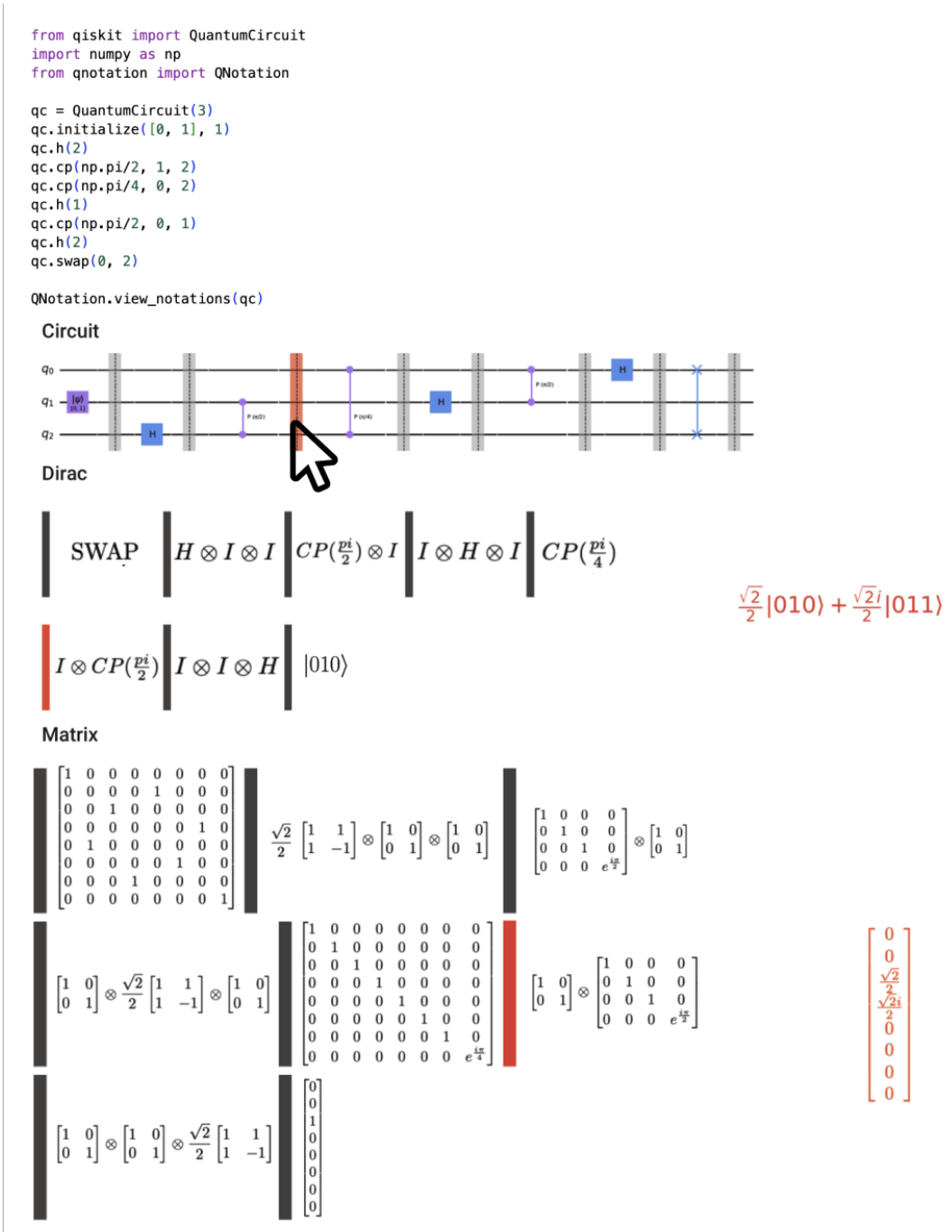


Figure 3.5.|: QNotation as a prototype Jupyter Notebook widget. The visualization part of the image is from our extended abstract *QNotation: An Interactive Visual Tool to Lower Learning Barriers in Quantum Computing* [79]

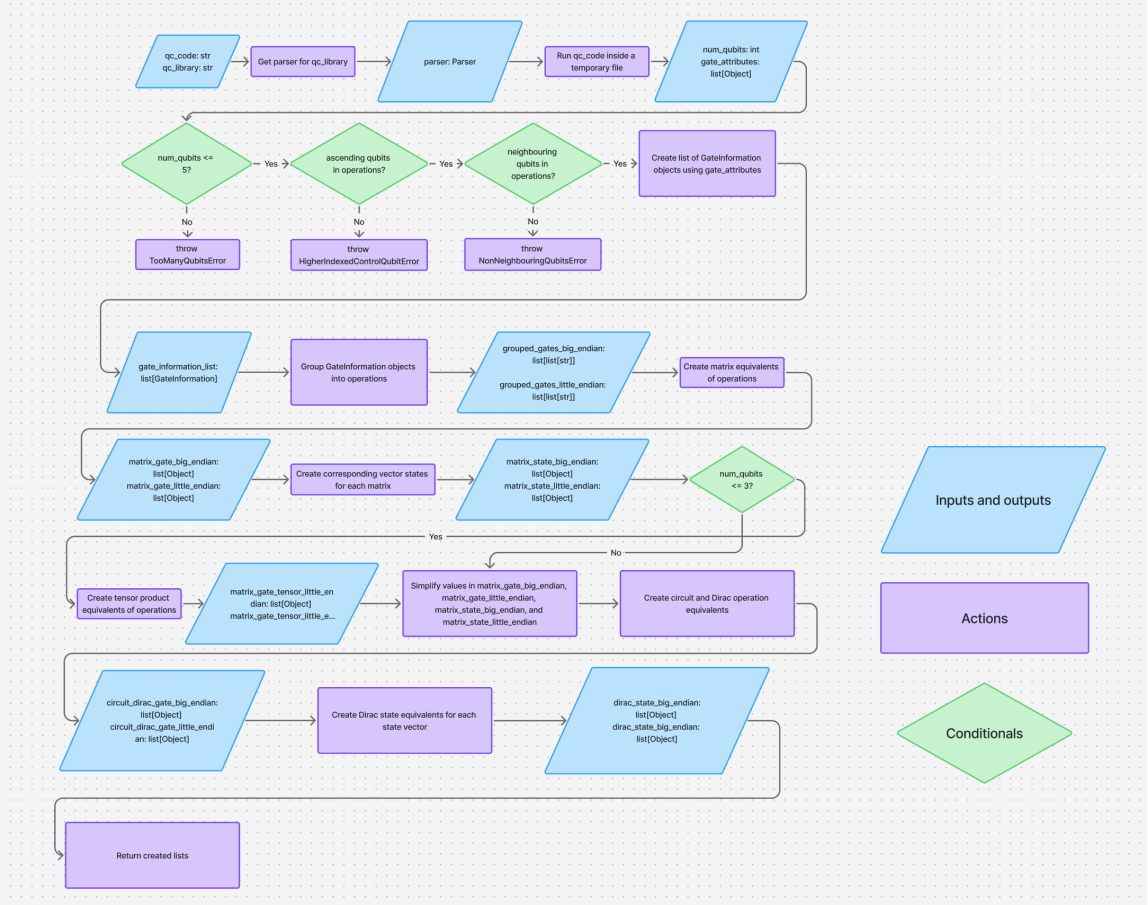


Figure 3.6.|: The steps taken on QNotation's backend once code is submitted from the frontend.

```python
    @abstractmethod
    def convert_code_string_to_circuit_object(self, qc_string):
        """
        Takes code as a string, runs it inside a temp file, and returns what is received from stdout

        Args:
        qc_string (str): string of code to be run

        Returns:
            list[int, Unknown]: list containing the number of qubits in the circuit and information about its gates
        """
        with tempfile.NamedTemporaryFile(delete=False, suffix=".py") as temp_file:
            temp_file.write(qc_string.encode("utf-8"))
            temp_file_name = temp_file.name
        try:
            result = subprocess.run(
                [sys.executable, temp_file_name],
                capture_output=True,
                text=True,
                timeout=5,
            )
            output = eval(result.stdout, {"qml": qml, "array": array, "cirq": cirq})

        except Exception:
            raise InputError
        finally:
            os.remove(temp_file_name)

        return output
```

Figure 3.7.|: The `convert_code_string_to_circuit_object()` abstract method within the `Parser` ABC.

```python
    def convert_code_string_to_circuit_object(self, qc_string):
        """
        Adds code for retrieving data from original qc_string and calls convert_code_string_to_circuit_object

        Args:
            qc_string (str): Pennylane code as a string

        Returns:
            list[int, Unknown]: list containing the number of qubits in the circuit and information about its gates
        """
        qc_string = qc_string + PENNYLANE_CIRCUIT_GATE_LOOP
        return super().convert_code_string_to_circuit_object(qc_string)
```

Figure 3.8.|: The `convert_code_string_to_circuit_object()` method within `PennylaneParser`. The method formats the code string received from the frontend so that details on its quantum circuit can be extracted by the `convert_code_string_to_circuit_object()` abstract method defined in the `Parser` ABC superclass.

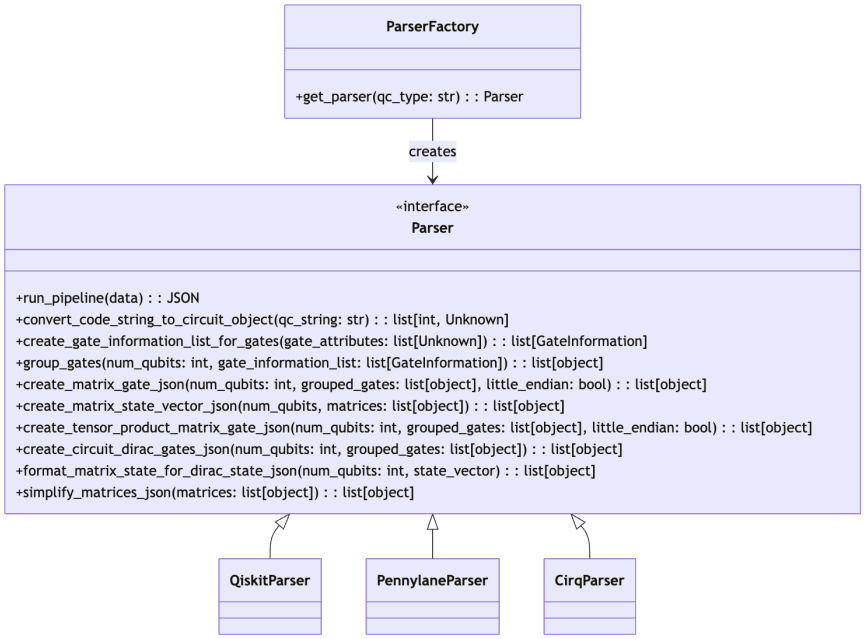


Figure 3.9.|: QNotation's parser classes as an XML Diagram. When `get_parser()` is called, a `Qiskit`, `PennyLane`, or `Cirq` parser is created depending on the argument given (`qc_type`). Some parsers have extra behaviour integrated into their implementations of the methods defined by `Parser`. An example of this is `convert_code_string_to_circuit_object()`, shown in Figure 3.7 and Figure 3.8.

and Figure 3.8 show the implementation of `convert_code_string_to_circuit_object()` in the Parser ABC and `PennylaneParser`, one of the subclasses of `Parser`.

Once the parser for the input received has been created, the following steps occur:

1. The inputted quantum circuit (received as a string) is run inside a temporary file, where, upon being successfully run, returns the number of qubits in the circuit and a list of the circuit's gate objects. These objects will vary between each library. If an error occurs, an `InputError` will be raised.
    a) If the number of qubits in the quantum circuit is greater than 5, a `TooManyQubitsError` will be raised.
    b) If a multi-qubit gate targets a qubit of a lower index, a `HigherIndexedControlQubitError` will be raised.
    c) If a multi-qubit gate targets non-neighbouring qubits, a `NonNeighbouringQubitsError` will be raised. This constraint ensures unambiguous mapping to column-wise matrices and Dirac factors under both endianness conventions.
2. The list of gate objects from the quantum circuit is converted into a list of `GateInformation` objects. At this point, it is not possible to determine which library the input originally came from.
3. The list of `GateInformation` objects is used to group the `GateInformation` objects into 'columns'. A column is an operation on the quantum circuit, meaning that it is applied to all qubits. If gates cannot cover all qubits identity matrices will be added in to ensure that all operations are valid. This process is done for little-endian and big-endian representation. After this step, it is impossible to tell which library the circuit originated from.
4. The list of gate columns are used to create a list of matrices with each matrix representing a column. This process is done for little-endian and big-endian representation.
5. The lists of matrices are used to create lists of state vectors representing the state of the quantum circuit after each column.
6. If the number of qubits in the quantum circuit is less than or equal to 3, the lists of grouped gates will be used to create tensor product representations of each column in little endian and big endian, respectively.
7. The values of the matrix lists are simplified and reformatted to ensure clean display.
8. The grouped gates lists are used to create a list of Dirac gates, which can also be used for the circuit visualizations.
9. The vector state lists are used to create Dirac states for each operation.

Figure 3.6 depicts the full visualization creation procedure.

3.4.3. Frontend

We developed the QNotation frontend using `React` and `Next.js` [128, 160]. `React` was selected for its efficient component-based architecture and real-time state updates, both essential for dynamically highlighting and redrawing visualizations as users explore the different notations. `Next.js` was chosen to facilitate server-side rendering, simplifying deployment and ensuring consistent performance across operating systems and browsers.

The frontend is organized into three distinct components: the code editor, the toolbar, and the notation visualization panels (circuit, Dirac, and matrix). User code is entered through the code editor and sent to the backend via a single API call, `notation_data`. `Monaco Code Editor for React` was used to create the code editor component, and `axios` is used to handle the API call [120, 16]. The components used in the toolbar are from the `Material UI` library [76]. The backend response is parsed into a structured `JSON` object containing circuit metadata, gate groupings, and formatted Dirac and matrix values. These are then rendered using custom components for each notation.

All visualizations are fully reactive. Selecting an operation in one notation triggers highlighting of the corresponding components in the other two notations. Endianness and tensor toggles immediately update the visualizations without additional backend requests. Errors returned from the backend, such as invalid syntax or exceeding qubit limits, are caught and displayed in contextual pop-ups to preserve workflow continuity.

3.4.4. Gate Support

To ensure that QNotation is robust–meaning it can reliably process diverse quantum circuits and gate combinations without unexpected failures–we ensured that a wide variety of quantum gates were supported by the tool. QNotation supports 43 `Qiskit` gates, 40 `PennyLane` gates, and 33 `Cirq` gates. Table B.1, Table C.1, and Table D.1 in the appendix display which gates are supported in their respective libraries. Currently, gates with varying qubit sizes are not supported by QNotation. These gates are denoted by † within the aforementioned tables. Gates that have been flagged as deprecated within the libraries have also not been implemented. These gates are marked with ‡. For `Qiskit`, only gates that have a method call were considered for implementation. Gates that were excluded due to this are denoted by * within Table B.1. For `PennyLane`, only gates labeled as parametrized and non-parametrized were considered for implementation [111]. Multi-qubit gates in QNotation must act on ascending neighbouring qubits (i.e., contiguous qubit indices in the circuit layout). Circuits with operations where the qubits used are in descending order (i.e. target qubit(s) are indexed lower than the control qubit(s) used) raise a `HigherIndexedControlQubitError`, and circuits with non-neighbouring qubit gates will cause a `NonNeighbouringQubit` error to be raised.

3.5. Additional Library Implementation

With quantum computing being a rapidly developing field, it is crucial that QNotation supports the addition of new quantum libraries as input. While QNotation currently supports input written using `Qiskit`, `PennyLane`, or `Cirq`, other quantum libraries written in Python can easily be added due to QNotation's parser factory and library-agnostic algorithms.

The following are traits that a quantum library must have in order to be added into QNotation:

- must be gate-based
- must be able to use a noiseless classical simulator
- must be written in `Python` or have a `Python` wrapper

When adding a library to QNotation, the following steps must be taken on the backend:

- The library's version must be added into `requirements.txt`
- A parser for the library must be created within `quantum_library_parsers` using the `Parser` abstract base class (ABC). Depending on the specifications of the library, certain functions within the ABC may need to be implemented or partially implemented before their super class version is called.
- Multi-qubit gates not yet supported by existing libraries must be added into `operation_info` using the `MultiQubitMatrixInformation` ABC. This includes adding the gates' imports `into matrix_info_registry`.
- A `Pytest` test suite must be added into test suites. The test suite's coverage must align with the coverage of the other test suites as much as possible.
- Gate coverage and library limitations must be documented within QNotation's `README`.

The following steps must be taken on the frontend:

- An option for selecting the library must be added.
- Starter code and example code for the library must be added into `Utils`.

3.6. Related Learning Exercises and Assessments

The following questions can be used to support the teaching and learning of foundational quantum computing concepts. We mark the questions that require knowledge beyond the notations with *.

B.Q.1 (a) In QNotation, create quantum circuits with $1, 2, ..., 5$ qubits that are all in equal superposition. (b) How are these circuits similar, and how are they different?

B.Q.2 How can entangled states versus non-entangled states be differentiated in Dirac versus matrix notation?

B.Q.3 Generate the pre-made Quantum Teleportation algorithm, and examine what happens to the qubit that originally contained the teleported values. What does this represent?

B.Q.4* Generate the pre-made Grover's search algorithm example, and identify the following components

1. Qubit initialization
2. Grover iteration application
    a) Phase oracle
    b) Diffuser

What are some of the characteristics of each component? You are allowed to iterate through the algorithm to get more information.

B.Q.5* Generate the pre-made Grover's search algorithm example. Which parts of the algorithm need to be changed in order for the desired value to be 0?

Algorithm 1: `Qiskit` code for the new oracle for **B.R.5**

```
qc.x(0)
qc.x(1)
qc.cz(0,1)
qc.x(0)
qc.x(1)
```

Potential answers for the aforementioned questions are

B.R.1 (a) Figure 3.10 and Figure 3.11 show the one-qubit and three-qubit solutions in QNotation (b) Equal superposition can be achieved by applying a $Hadamard$-gate to each qubit in the circuit. In Dirac notation, all states will be present in the quantum state with probability values equal to $\frac{1}{2^n} \times 100$, where *n* is the number of qubits in the circuit. The same probability values will be seen in the state vector in matrix notation.

B.R.2 In matrix notation, entangled states are represented by vectors that cannot be described as a tensor product. All states in superposition must contain two ket components in Dirac notation. As states that are entangled must be in superposition, this means that this trait applies to all entangled states as well.

B.R.3 The original qubit had to be collapsed. This demonstrates the no-cloning theorem [109].

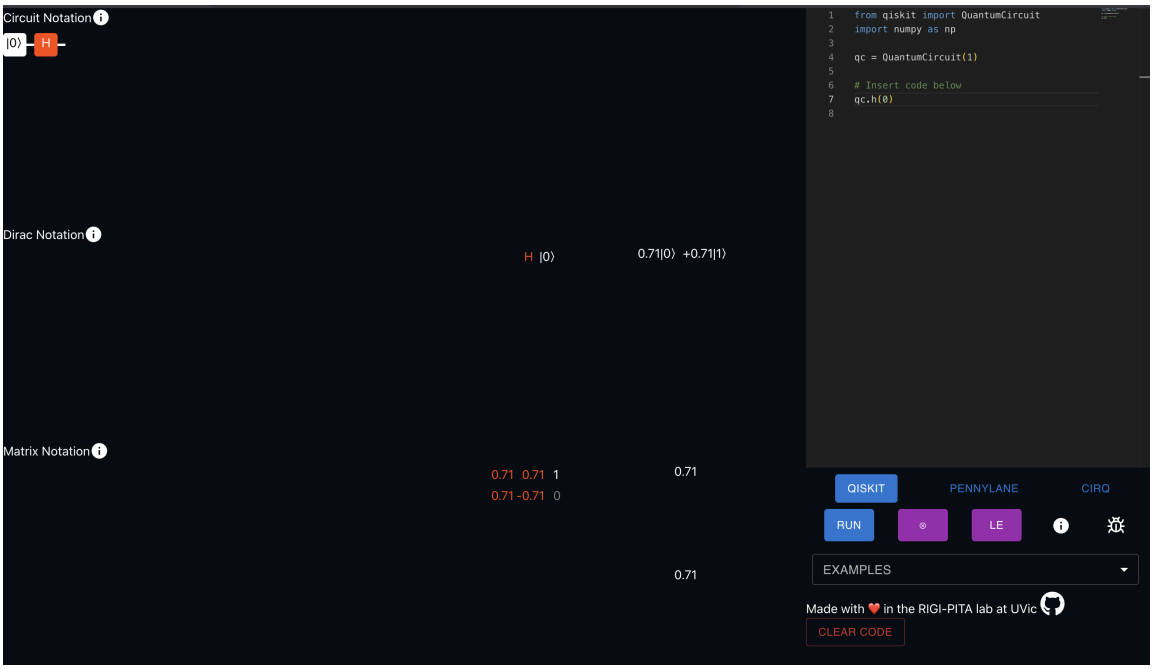


Figure 3.10.|: The one-qubit solution for **B.Q.1**.

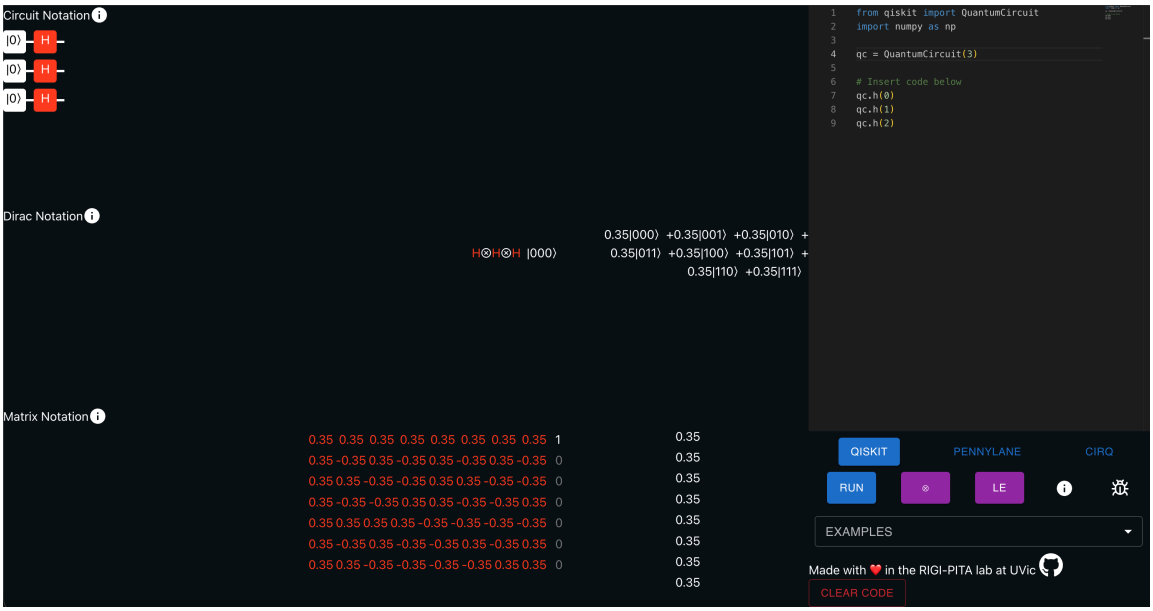


Figure 3.11.|: The three-qubit solution for **B.Q.1** Users can scroll vertically in the matrix notation component to see the full matrix.

B.R.4 After the qubits have been initialized, all values are in equal superposition. This can be seen in matrix notation, as all values inside the state vector are the same. The phase oracle applies a negative phase to the desired solution(s), which in this case is only 3 (i.e. $|0011\rangle$). This leads to a negative sign being seen in the state vector. The diffuser changes the amplitudes of the values. The amplitude of 3 increases while the amplitudes of the other values decrease.

B.R.5 Only the oracle needs to be changed as it is the component of the algorithm that applies an alteration (i.e. the phase flip) to the desired value(s) specifically. Therefore the oracle should be updated to a quantum circuit that is equal to Algorithm 1.

Chapter 4

Verification, Testing, and Reproducibility in QNotation

Because educational tools must maintain conceptual accuracy across evolving software dependencies, QNotation's testing and verification framework was developed to ensure predictability, reproducibility, and stability over time. This framework aligns with best practices in scientific software engineering, emphasizing automated validation and end-to-end testing to preserve correctness as the system evolves. Three main goals guide this framework: verifying correctness, preventing regressions, and increasing reliability. The following sections describe how each goal is achieved through automated testing and continuous integration practices.

Table 4.1.: Gate Coverage within QNotation

Library	Supported Gates	Library Coverage	Library Coverage Within Scope	Test Suite Coverage
Qiskit	45	54%	100%	100%
PennyLane	40	87%	100%	100%
Cirq	33	85%	100%	100%

4.1. Verifying Correctness

To ensure QNotation's correctness, we split its verification into two components: (1) the correctness of the supported gate implementations and (2) the fidelity of its integration with external quantum libraries. Correctness refers to whether QNotation's internal gate implementations produce mathematically accurate quantum states and transformation. Fidelity, in contrast, refers to how faithfully QNotation integrates with and reflects the behaviour and outputs of external quantum computing libraries. Both are verified through QNotation's test suites and further analysis done within this chapter.

4.1.0.1. Correctness of Gates

QNotation's test suites are built using `Pytest`. Pytest was chosen due to its flexible and class-based architecture. Each test suite contains several test classes, which are all subclasses of the `TestQNotation` ABC, shown in Figure 4.1. Each test class, listed in Table E.1, tests a different quantum circuit. The goal of the test suites is to ensure that every implemented quantum gates behaves correctly. Whenever a test class is run, an `httpx` call using the given code input for that test class (ex. a `Qiskit` quantum circuit with a `CCX` gate for `TESTCCX` within the `Qiskit` test suite) is executed [42]. The call happens within the `setup_class` class method, and its output is used by all tests within its test class. The `httpx` library is used to send HTTP requests to the QNotation backend, allowing tests to replicate real user calls to the API rather than isolated function calls. Calling the API once before an entire test class is run instead of calling it for each individual test minimizes the number of calls needed whenever the test suites are run.

As shown in Table 4.1, all gates supported by QNotation are tested within the test suites. This helps ensure the correctness and robustness of the tool's gate support. Table B.1, C.1, and D.1 provide further details on the gates implemented and tested by the QNotation test suites. Some gates are marked with the following symbols:

- §: one-parameter gate tested with $\pi/4$, $\pi/2$, and π (3 separate test instances).
- ¶: two-parameter gate tested with $\pi/4$, π; and $\pi/2$, π (two separate test instances).
- ‖: three-parameter tested with $\pi/4$, $\pi/2$, and π (one test instance)

```python
class TestQNotation(ABC):
    """Abstract base class for test cases"""
    @classmethod
    @abstractmethod
    def setup_class(cls):
        pass

    @abstractmethod
    def test_status_code(self):
        pass

    @abstractmethod
    def test_num_qubits(self):
        pass

    @abstractmethod
    def test_circuit_dirac_little_endian(self):
        pass

    @abstractmethod
    def test_circuit_dirac_big_endian(self):
        pass

    @abstractmethod
    def test_dirac_state_little_endian(self):
        pass

    @abstractmethod
    def test_dirac_state_big_endian(self):
        pass

    @abstractmethod
    def test_matrix_gate_little_endian(self):
        pass

    @abstractmethod
    def test_matrix_gate_big_endian(self):
        pass

    @abstractmethod
    def test_matrix_gate_tensor_little_endian(self):
        pass

    @abstractmethod
    def test_matrix_gate_tensor_big_endian(self):
        pass

    @abstractmethod
    def test_matrix_state_little_endian(self):
        pass

    @abstractmethod
    def test_matrix_state_big_endian(self):
        pass
```

Figure 4.1.|: The `TestQNotation` ABC

This structured parameterization ensures that QNotation's mathematical transformations are validated across representative input values.

4.1.1. Correctness of Library Integration

QNotation integrates multiple quantum libraries, each of which differs in syntax and default qubit ordering. The aforementioned tests verify that QNotation preserves consistent behaviour across these libraries by comparing the results of equivalent tests (ex. `TESTCCX` using a `Qiskit` circuit and `TESTCCX` using a `Cirq` circuit) against the same expected output. These tests ensure that QNotation does not introduce discrepancies between libraries, thereby validating the correctness of its parser layer.

Table 4.1 highlights the total number of gates implemented in QNotation for each library. While library coverage varies between the libraries, 100% scope coverage is achieved for all of them. For `Qiskit`, this means that all non-deprecated gates with fixed sizes and designated method calls have been implemented; for `PennyLane`, all fixed-size gates classified as parametrized and non-parametrized have been implemented; and for `Cirq`, all fixed-size gates have been implemented.

4.2. Preventing Regressions

Whenever a change is made to a branch in the QNotation repository, the QNotation test suites are run using `GitHub Actions` [46]. This helps prevent accidental system regressions. Line coverage is also tested to ensure that the QNotation test suites are comprehensively testing the system. The `pytest-cov` library is used to test line coverage, further ensuring the robustness of the tool [61]. In order for code to be merged, all tests within the test suites must pass, and a line coverage value of 95% must be achieved.

4.3. Increasing Reliability

QNotation ensures consistent gate behaviour between libraries by testing each of a gate's test classes against the same expected output. Reliability is also ensured through the testing of expected user errors. These include inputting code with a typo (`TestTypo`), inputting code with unsupported gate(s) (`TestUnsupported`), and inputting code with non-neighbouring qubits within multi-qubit gates (`TestNonNeighbouringQubits`).

Reproducibility further strengthens QNotation's reliability by guaranteeing that identical inputs produce identical outputs across environments and versions. This is achieved through automated testing and continuous integration via `GitHub Actions`, which execute all tests in controlled environments. Version control and dependency specifications ensure that library versions (e.g., `Qiskit`, `Cirq`, and `PennyLane`) remain consistent between test runs. Together, these practices allow QNotation's behaviour to be reliably replicated by any

contributor, reinforcing confidence in its stability. Rigorous automated testing is particularly important in educational software, where conceptual accuracy directly affects learning outcomes.

Chapter 5

QNotation Pilot User Study and Feedback

This chapter details the construction of a user study for QNotation as well as anecdotes from a pilot version of the study. Due to the low participation count in the pilot offering of the study, we also include informal user feedback we have received on QNotation.

5.1. Pilot User Study

To ensure that QNotation fits the needs of the introductory quantum computing classroom, an exploratory pilot study was done. The study, titled *Achieving Quantum Computing Literacy through Investigating Student Learning Patterns Utilizing Exploratory Learning through QNotation*, was conducted in *SENG457/CSC557: Quantum Algorithms and Software Engineering* at the University of Victoria during the Summer 2025 semester [101].[1] The class consisted primarily of computer science, software engineering, and physics students. We detail the format of our study and the anecdotes received from the pilot offering of the study in the following sub-sections:

5.1.1. Study Background

The study consisted of three different questionnaires, shown in Appendix F, Appendix G, and Appendix H. The version of QNotation used for the study supported `Qiskit` input using the following gates: `CCX`, `CCZ`, `CH`, `CP`, `CRX`, `CRZ`, `CS`, `CSDG`, `CU`, `CX`, `CZ`, `H`, `ID`, `R`, `RCCX`, `RX`, `RY`, `RZ`, `X`, `Y`, `Z`. The study aimed to answer the following research questions:

1. Is QNotation effective for teaching the three notations commonly used in quantum computing?
2. Is QNotation a useful tool in later parts of introductory courses, once the introduction to notation is completed?
3. Is QNotation useful for teaching quantum computing foundations to audiences of diverse technical backgrounds?

[1] The Human Research Ethics Standard Application number for the study is $24 - 0554$.

Our intention was to hold the study over eight weeks, with the second week of the semester being when we would start the study. We planned to give Appendix F as the initial survey, Appendix G during weeks three through eight, then Appendix H during week nine. The surveys were delivered to participants using UVic's SurveyMonkey [129]. We cover how we believe the surveys help answer the aforementioned research questions in the following sub-sections:

5.1.1.1. Is QNotation effective for teaching the three notations commonly used in quantum computing?

We aim to answer this question using the notation comprehension questions from Appendix F and the Appendix G survey given during week three. We believe that comparing participants' initial comprehension with their use of QNotation during the week that notations are being covered will show how students learn or revisit the notations.

5.1.1.2. Is QNotation a useful tool in later parts of introductory courses, once the introduction to notation is completed?

The surveys given after week 3 help answer this question through prompts such as *Did you use QNotation while studying?* (question 2, Appendix G), and *When [during the course] was QNotation most useful for you? (long answer)* (question 3, Appendix H).

5.1.1.3. Is QNotation useful for teaching quantum computing foundations to audiences of diverse technical backgrounds?

Appendix F aims to establish the academic background of each participant so that we can see how they use QNotation and grow over the course of the term. We aim to answer this question over the course of the study with question 3 (*If you did use QNotation, please explain how you used it. (long answer)*) from Appendix G.

5.1.2. Pilot Study Anecdotes

While interest in QNotation has been high at the conferences it has been presented at and during classroom presentations, two students signed up for the pilot study, and only one participant filled out most of the surveys. In this section, we refer to them as Participant A and Participant B. Due to delayed responses and low sign-up count, not all surveys were able to be collected. Participant A completed the most surveys and Participant B completed the initial survey. The data is further analyzed in Chapter 7.

Table 5.1.: Academic backgrounds of Participant A and Participant B

Question	Participant A	Participant B
Have you ever taken an undergraduate physics course (not counting courses being taken this semester)?	Yes	Yes
Have you ever taken an undergraduate computer science course (not counting courses being taken this semester)?	Yes	Yes
Have you ever taken an undergraduate linear algebra course (not counting courses being taken this semester)?	Yes	Yes

Table 5.2.: Participant A and Participant B's answers to Survey One

Question	Participant A	Participant B
Please select the correct resulting vector	B	D
Please select all statements that describe the following circuit	A, C	A, C
Please select the correct answer for $\|\psi\rangle$	A	B

5.1.2.1. Academic Backgrounds

Table 5.1 shows the answers given by Participant A and Participant B for the academic background questions asked in Survey One (Appendix F). It was found that Participant A and Participant B have similar academic backgrounds.

5.1.2.2. Quantum Basics Proficiencies

Table 5.2 shows Participant A and Participant B's responses to the quantum basics proficiencies asked in Survey 1. Despite having similar academic background, there was some variety in their responses.

5.1.2.3. Participant A's Experiences with QNotation

Table 5.3 shows Participant A's answers to Survey Two and Survey Three. To give more context to these answers, we detail the the topics being covered at the time of the surveys being taken in Table 5.4 [101].

Table 5.3.: Participant A's answers to Survey Two and Survey Three.

Question	Survey Two Answers	Survey Three Answers
How many hours did you study for the course this week?	4-6 hours	1-3 hours
Did you use QNotation while studying?	No	No
If you did use QNotation, please explain how you used it	–	–
QNotation can be used with what I learned this week in class	Yes	No

Table 5.4.: The topics being taught during the weeks the surveys were answered

Survey	Topics
Survey One	Measurement, bases, Bloch sphere, one and two-qubit circuits
Survey Two	Teleportation, phase kickback, and Grover's algorithm
Survey Three	Expectation values and Quantum Fourier Transform

5.2. Feedback

We have been privileged to receive informal feedback throughout the creation of the QNotation prototype and the current version of QNotation. This feedback has come from conference goers at IEEE Quantum Week [59] as well as students and professors at the University of Victoria, Simon Fraser University, and the University of British Columbia. The following list details the different themes we have seen in the feedback that we have received:

- **QNotation is easy to use.** Many of those who we have talked to have been surprised by how easy it is to start using QNotation, especially since it supports the learning of notations, which are known to be difficult to learn.
- **QNotation would benefit from more algorithmic code examples.** Some students, particularly those just beginning to learn about quantum algorithms stated that more algorithmic code examples would help them with their learning.
- **From an educational perspective, QNotation's number of supported qubits is not a limiting factor.** The educators who have reviewed QNotations have stated that they believe that five qubits is a sufficient number for showing diverse examples within an introductory course.
- **Descending qubit order and non-neighbouring qubit support would make QNotation's integration into classrooms even easier.** Currently, QNotation only supports ascending neighbouring qubits for multi-qubit gates. While QNotation's endian toggle does address the former, not being able to input descending qubits may confuse learners initially. Allowing for non-neighbouring qubits to be used in multi-qubit gates will expand the quantum algorithms that can be visualized.

Chapter 6

QGrover

Grover's algorithm is one of the quantum algorithms introduced in undergraduate and early graduate quantum computing courses [19]. It offers a clear, concrete speedup over classical unstructured search [50], but students often struggle to build intuition for how its amplitudes evolve and why the algorithm works [105]. In particular, the role of the phase oracle and diffuser, and their combined effect over multiple iterations, can be difficult to grasp from algebraic derivations or static circuit diagrams alone.

Taking inspiration from QWalkVis, which visualizes quantum walks as an exploratory learning tool [65], QGrover was created as a complementary tool for visualizing Grover's algorithm. QGrover is a browser-based application that allows users to specify the number of qubits, the number of solution values, and the number of iterations (or to use the ideal number of iterations), and then observe how the quantum circuit and corresponding state amplitudes change throughout the algorithm. By clicking on different parts of the circuit, learners can inspect the amplitudes at each stage, helping them connect the mathematical description of Grover's algorithm with its operational behaviour.

QGrover is designed with exploratory learning in mind. Rather than simply running a fixed example, students can vary the aforementioned parameters, and immediately see how these choices affect the algorithm.

6.1. Grover's Algorithm

Grover's algorithm [50] is a popular unstructured-search algorithm with a variety of applications. It is used as a subroutine in quantum walk, quantum cryptography, and quantum machine-learning algorithms [63, 159, 73, 40]. Let us imagine searching through a bag of wrapped candy where the pieces of candy are varied, but all of the wrappers are the same colour, making it impossible to distinguish between the pieces. If there exists exactly one piece of the candy that is wanted, in the worst-case scenario, one will have to take out and unwrap each piece of candy in order to find their desired piece. In other words, for N pieces of candy, one must potentially pick up and unwrap all N pieces. This describes how classic unstructured-search algorithms function.

When using Grover's algorithm, one does not need to pick up and unwrap each piece of candy one at a time. Instead, as the search progresses, one might imagine that the wrapper

of the desired piece of candy beginning to glow more and more, making it more and more apparent where the desired piece is. This leads to only $\sqrt{N}$ candy unwrapping actions needing to happen. An example of this analogy is shown in Figure 6.1.

To search using Grover's algorithm, we assume an unstructured search space S of size N, consisting of elements $0, 1, 2, \ldots, N-1$, where N is a power of two. The size of the search space N defines the number of qubits, n, to be initialized for running Grover's algorithm on this search space: $n = \lceil \log N \rceil$.

In addition to the search space, Grover's algorithm takes as input one or more desired values to be found. These are encoded in the form of something called a *phase oracle*. The oracle is an operator made up of quantum gates that, when given an element from the search space, is able to detect whether or not the element is a desired value.[1]

Grover's algorithm can be broken down into the following components:

1. Quantum state/search space preparation
2. Grover iterator
    a) Phase oracle
    b) Diffuser
3. Measurement

The *Grover iterator*, consisting of applying the phase oracle followed by the diffuser, is executed a particular number of times that maximizes the probability to measure the elements to be detected. This number depends on both the search space size (i.e. the number of qubits required to encode the search space) and the number of elements to be detected. The optimal number of iterations can be calculated by

$$\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} \right\rfloor \tag{6.1}$$

where M refers to the number of desired values. Equation 6.1, when applied to the candy example in Figure 6.1, tells us that the ideal number of iterations of the Grover iterator is 2, since for $N = 16$ and $M = 1$, $\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} \right\rfloor = 2$. Note that for this example, Grover's algorithm requires a circuit with $n = \lceil \log 16 \rceil = 4$ qubits. The following equations can be used to calculate the amplitudes at each stage of the algorithm.

$$\alpha = \left(\frac{1}{\sqrt{2}}\right)^n \tag{6.2}$$

$$a = \frac{1}{N}\sum_{x \in S} \alpha_x \tag{6.3}$$

[1] For an illustrative explanation on oracles and their role, we refer the reader to [64]. Figure 6.7 walks through a small example of applying a phase oracle.

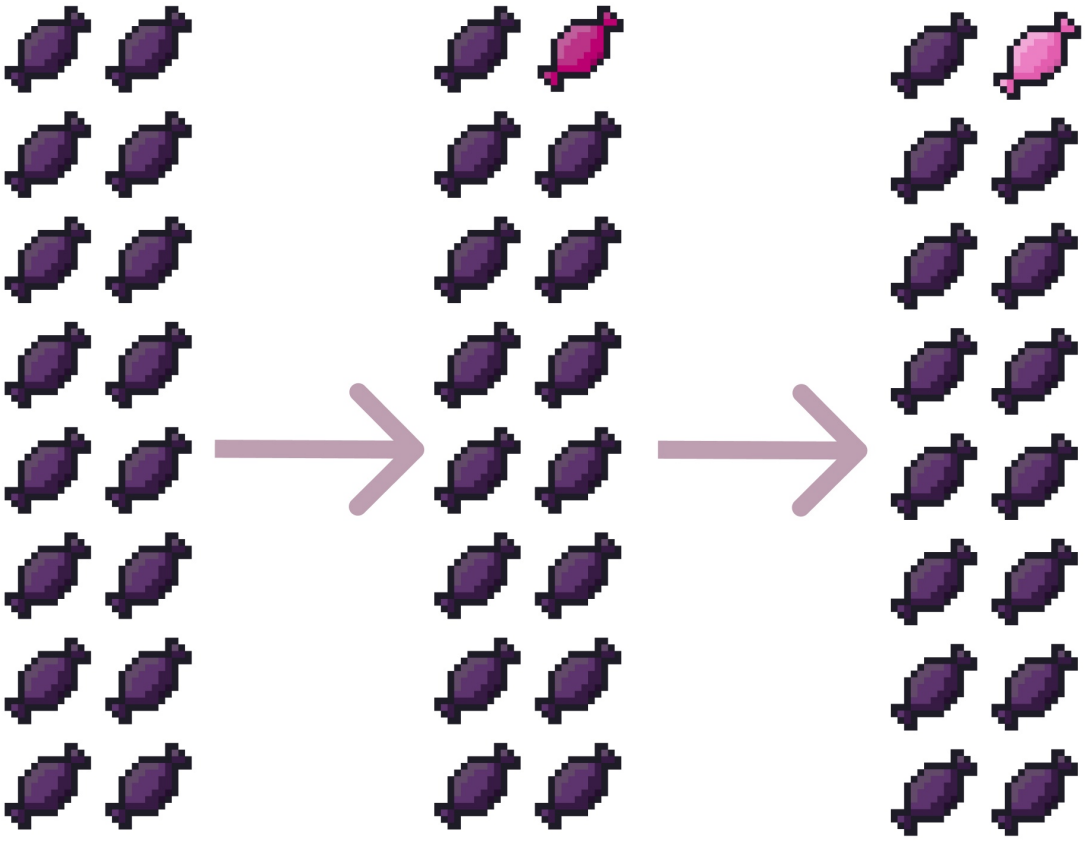


Figure 6.1.|: Analogy demonstrating the application of two iterations of the Grover iterator. The piece of candy in the upper right corner of each iteration is the desired piece.

$$\alpha'_x = 2a - \alpha_x \tag{6.4}$$

Equation 6.2 is used to calculate the initial amplitude α of all values in the set. Equation 6.3 and Equation 6.4 are used to calculate the subsequent amplitude means a and amplitudes α'_x.

For the candy example, initially $\alpha = (\frac{1}{\sqrt{2}})^4 = 0.25$. After the application of the oracle, the amplitude of the desired piece of candy is -0.25 while the undesired pieces still have an amplitude of 0.25.

Applying Equation 6.3 gives $a = 0.21875$, which gives the following amplitudes for the desired candy piece and the undesired pieces of candy respectively:

$\alpha'_{desired} = 2 \times 0.21875 - (-0.25) = 0.6875$

$\alpha'_{undesired} = 2 \times 0.21875 - (0.25) = 0.125$

After applying the oracle a second time, the amplitudes become -0.6875 and 0.125. This leads to the mean becoming 0.07421875. Applying the diffuser leads to the following updated amplitudes:

$\alpha'_{desired} = 2 \times 0.07421875 - (-0.6875) = 0.8359375$

$\alpha'_{undesired} = 2 \times 0.07421875 - (0.125) = 0.0234375$

6.2. How To Use QGrover

The following steps explain QGrover's workflow:

1. When the application is first opened, the user is greeted with a modal[2] explaining Grover's algorithm and QGrover. The modal, shown in Figure 6.2 includes pages on the following questions:

   - What is QGrover?
   - What does Grover's algorithm do?
   - How does Grover's algorithm work?
   - How do I use QGrover?

   We included this modal to help ensure that users are provided enough documentation on Grover's algorithm and QGrover.

2. Once the user exits the modal, they are greeted by a blank page with a few input boxes, which can be seen in Figure 6.5. The following are specifications for the inputs:

[2] A *modal* is a user interface element that appears as a temporary overlay on top of the main application view, requiring the user to interact with it or dismiss it before returning to the underlying content. Modals are commonly used to present focused information, request user input, or confirm actions.

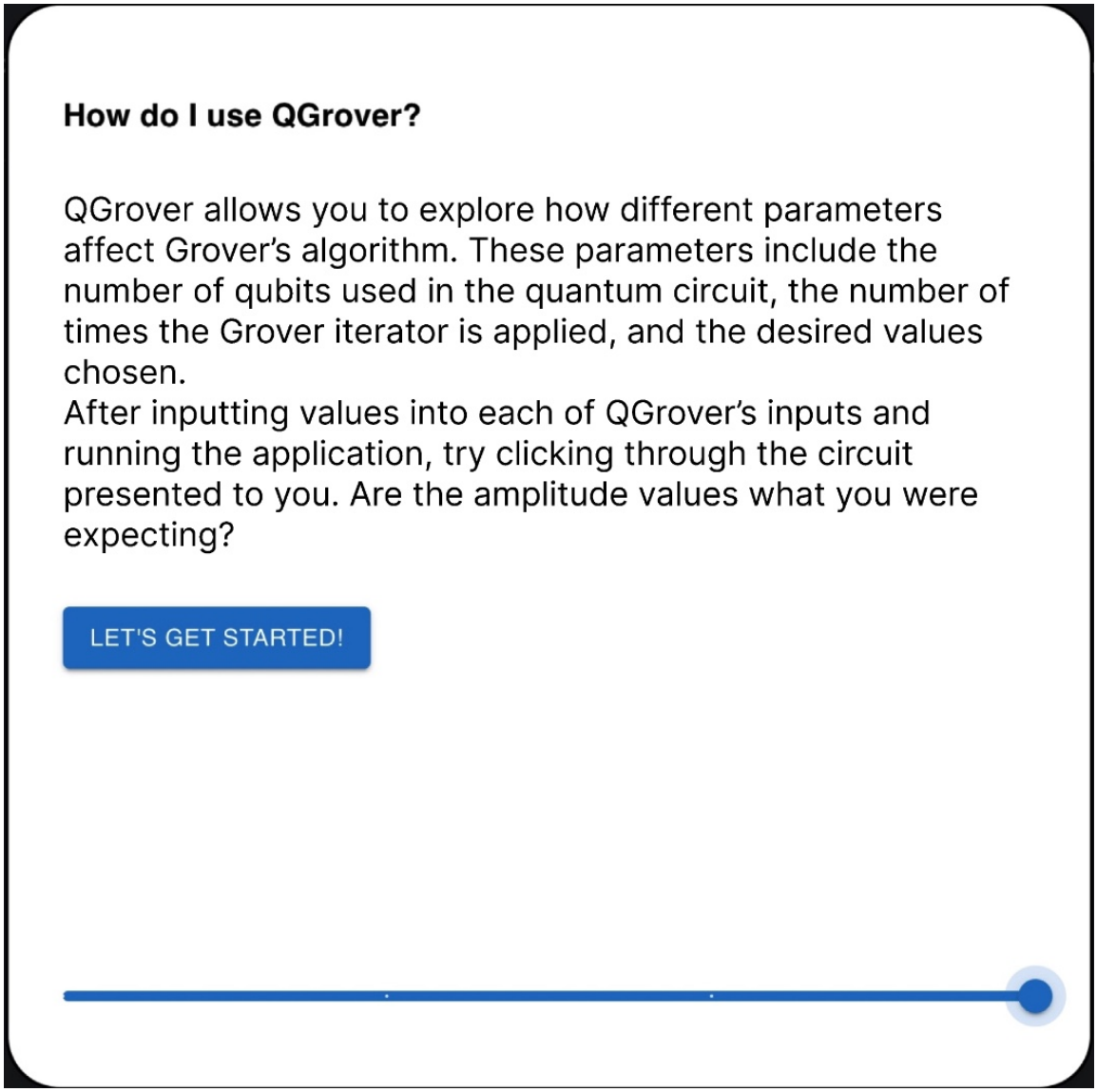


Figure 6.2.|: The last slide of QGrover's modal.

- The number of qubits given must be between 2 and 6.
- The number of iterations given must be between 1 and 6. If desired, the user can choose for the ideal number of iterations to be used by checking 'Use ideal iterations'.
- The number of solution values must be between 0 and $N - 1$, where N is the square of the number of qubits given

If values outside of these are given, QGrover will display errors intended to help the user input correct values. This was done to ensure that users receive timely feedback on how to use the application. These error messages can be seen in Figure 6.6.

3. Once all inputs have been given, the user clicks on 'RUN GROVER' to generate the visualizations

4. QGrover contains two visualizations: Grover's algorithm as a quantum circuit and a quantum state bar chart. Clicking on different components of the circuit will show the respective quantum state amplitudes at that point in the algorithm. We made the quantum circuit clickable to allow users to further interact with Grover's algorithm.

6.3. Implementation

The following subsections detail QGrover's implementation. The backend and frontend repositories for the application can be found on GitHub.

6.3.1. Backend

QGrover's backend was made using `Python`, `Qiskit`, and `Flask`. Upon receiving the number of qubits, number of iterations, and number of solution values from the frontend, the backend begins constructing Grover's algorithm. If the user chose to have the number of ideal iterations used, the number of qubits and number of solution values are used to calculate the number of iterations. This is done using Equation 6.1.

These values are then used to construct custom unitary matrices representing the oracle and diffuser for the algorithm. Figure 6.3 and Figure 6.4 show the functions used to create the oracle and diffuser. The lists returned are then used to create a unitary matrix that can be appended onto a `Qiskit QuantumCircuit` object using `UnitaryGate` [57]. Then, after all qubits have been put into superposition, the oracle and diffuser are applied repeatedly based on how many iterations were chosen.

```python
def create_oracle(qubits, solutions):
    operator_matrix_rows = []
    n = pow(2, qubits)
    for i in range(0, n):
        row = []
        for j in range(0, n):
            if i == j and solutions.count(i):
                row.append(-1)
            elif i == j:
                row.append(1)
            else:
                row.append(0)
        operator_matrix_rows.append(row)
    return operator_matrix_rows
```

Figure 6.3.|: The `create_oracle` algorithm.

```python
def create_diffuser(qubits):
    operator_matrix_rows = []
    n = pow(2, qubits)
    for i in range(0, n):
        row = []
        for j in range(0, n):
            if i == j:
                row.append((2.0/n)-1)
            else:
                row.append(2.0/n)
        operator_matrix_rows.append(row)
    return operator_matrix_rows
```

Figure 6.4.|: The `create_diffuser` algorithm.

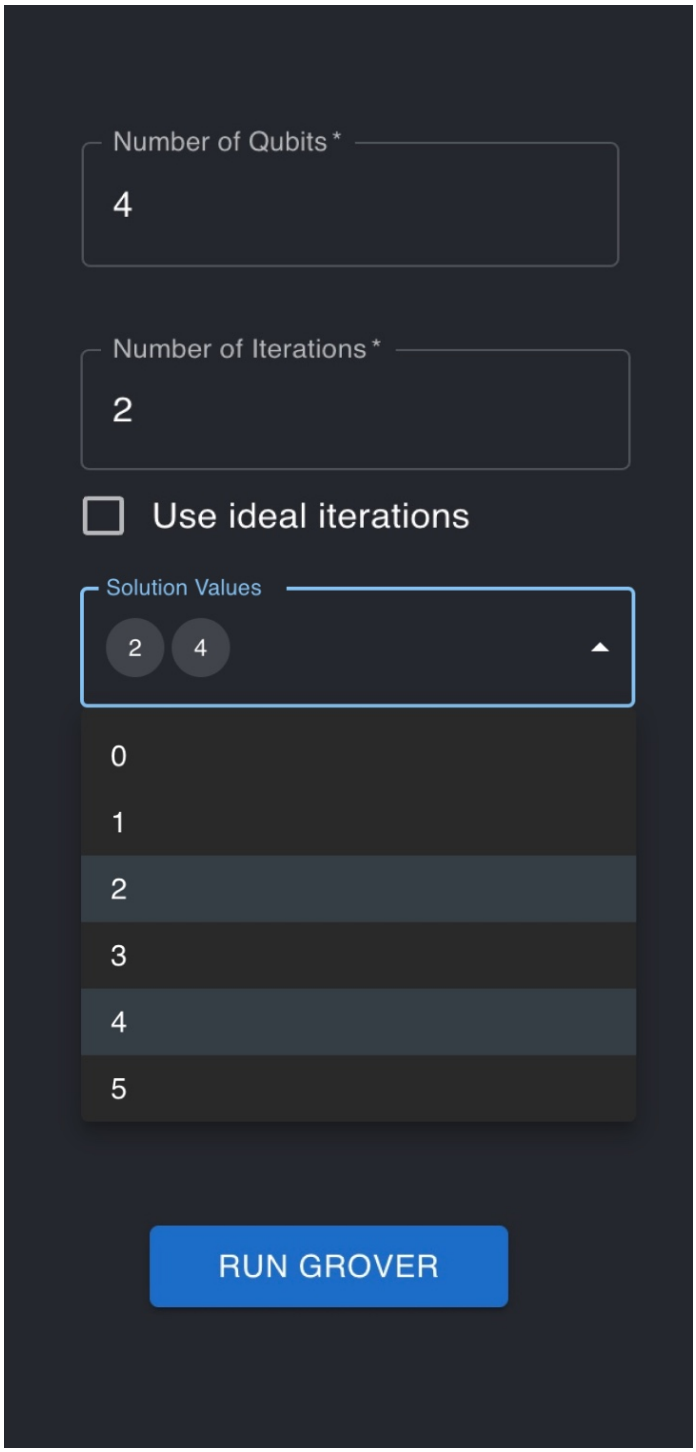


Figure 6.5.|: QGrover's input bars.

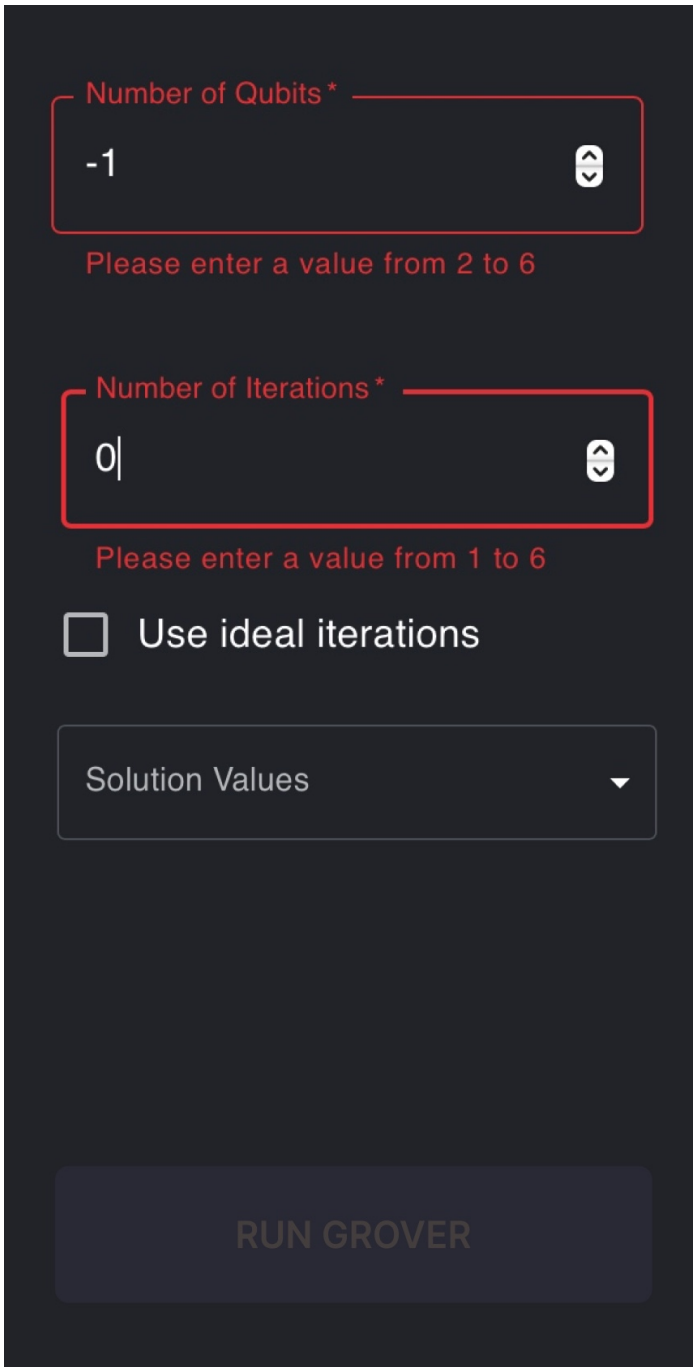


Figure 6.6.|: QGrover's input bars with error messages.

6.3.2. Frontend

We created QGrover's frontend using `React` and `Next.js` [128, 160]. Like with QNotation, `React` and `Next.js` were chosen due to their component-based architecture and real-time state updates. These traits are important due to both visualizations needing to be reactive to user actions.

QGrover's frontend can be split into three components: input, quantum circuit visualization, and bar chart visualization. The following sub-sections detail their implementation.

6.3.2.1. Input

We created the input component, shown in Figure 6.5, using `Material UI` [76]. The aforementioned parameter ranges are enforced through error handling within the inputs. When an invalid parameter has been entered, the 'RUN GROVER' button will be dimmed and cannot be clicked upon. This can be seen in Figure 6.6.

6.3.2.2. Quantum Circuit Visualization

The quantum circuit visualization component renders the Grover circuit based on the parameters provided by the user. Each oracle and diffuser block is represented as a labeled unitary operation. When a user clicks on a component (e.g., the first diffuser), the state amplitudes after the application of the component are displayed in the bar chart visualization.

6.3.2.3. Bar Chart Visualization

The bar chart visualization was made using `React-Vis` [74]. We chose to use `React-Vis` due to its `React` compatibility and customizable components.

6.4. Related Learning Exercises and Assessments

The following sample questions can be asked in assignments or as classroom activities to support the learning of Grover's algorithm alongside QGrover. We suggest that these questions be used with students ranging from late high school to early undergraduate levels, who have a basic understanding of computer science (i.e. they have been introduced to foundational classical algorithms) as well as a basic understanding of qubits, superposition, quantum circuits and quantum gates.

B.Q.01 How many qubits are needed to support a set containing N items?

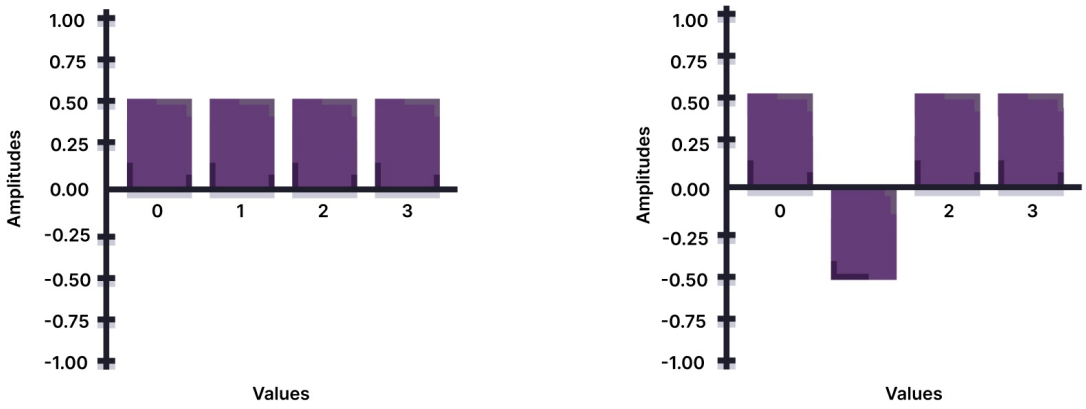


Figure 6.7.|: The application of a phase oracle to a set of size $N = 4$, with number of qubits $n = 2$, and with $M = 1$ where 1 is the desired value. Each value initially has an amplitude of $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. The oracle flips the phase of the desired value. This is done by multiplying the amplitude of the desired value by -1. In this example, the new amplitude value is $-\frac{1}{2}$ (as $-1 \times \frac{1}{2} = -\frac{1}{2}$).

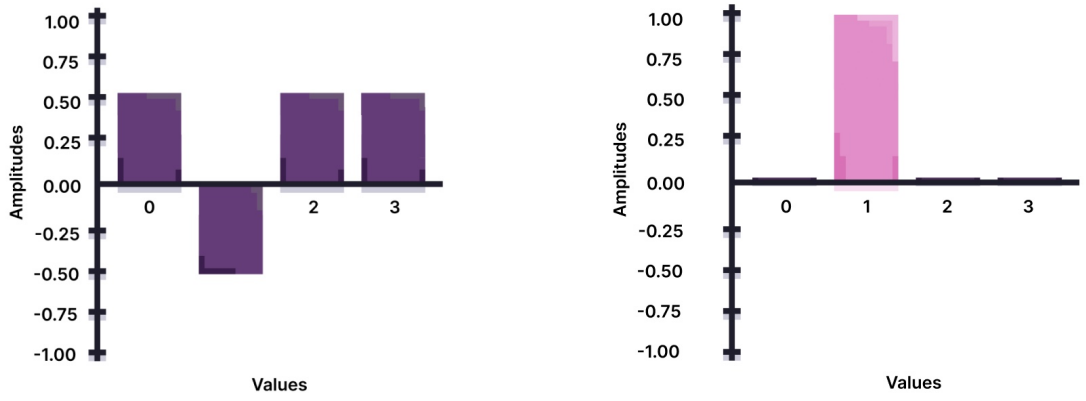


Figure 6.8.|: The application of a diffuser to a set of size $N = 4$. The equation used to calculate the new amplitudes is $\alpha'_x = 2a - \alpha_x$. When the diffuser is applied for the first time, $a = \frac{N-2}{N\sqrt{N}}$. When doing the calculations for this example specifically, one should infer that this is the first application of the of the diffuser due to the magnitudes of the amplitudes all being equal. Therefore, for the desired value, $a = \frac{4-2}{4\sqrt{4}} = \frac{1}{4}$ and the amplitude of the term of wanted element is $2 \cdot \frac{1}{4} - (-\frac{1}{2}) = 1$, while all other amplitudes equal $2 \cdot \frac{1}{4} - \frac{1}{2} = 0$.

B.Q.02 How does Grover's algorithm compare to search algorithms in classical computing?

B.Q.03 Examine Figure 6.7. Use QGrover to figure out which component of Grover's algorithm it represents. Can you replicate the example in QGrover?

B.Q.04 Examine Figure 6.8. Use QGrover to figure out which component of Grover's algorithm it represents. Can you replicate the example in QGrover?

B.Q.05 How does the phase oracle seen in Grover's algorithm work? Explain in mathematical terms.

B.Q.06 How does the diffuser seen in Grover's algorithm work? Explain in mathematical terms.

B.Q.07 How can one calculate the number of iterations that should be used for Grover's algorithm? Which variables are important?

B.Q.08 Does increasing the number of solutions in a set affect Grover's algorithm?

B.Q.09 Does adding more iterations always improve Grover's algorithm?

B.Q.10 Is it possible for solution values to be found with 100% certainty using Grover's algorithm? If yes, explain what such instances must satisfy. If no, explain why.

Possible answers for the aforementioned questions have been included in the list below.

B.R.01 $\log N$ qubits are needed to support a set of N items.

B.R.02 Grover's algorithm vastly outperforms a classical unstructured search. This is due to, in the worst case, every set item having to be checked during a search. This is referred to as having a running time of $O(N)$. The running time of Grover's algorithm is $O(\sqrt{N})$. Fortunately, not all classical search algorithms have a running time of $O(N)$. For example, binary search has a running time of $O(\log N)$ [125] for search operations. However, this running time can only be achieved due to the explicit structure provided to the search space. The implementation of these structures cause the algorithms to have greater space complexity (i.e. the memory that has to be used to maintain the structure) than Grover's algorithm.

B.R.03 The image represents the first application of a phase oracle onto a set with four values ($N = 2^2$) and one desired value ($M = 1$). The image can be recreated in QGrover by setting the number of qubits equal to 2 and 1 as the desired value and then clicking on the first/only oracle (depending on what is put for the number of iterations).

B.R.04 The image represents the first application of a diffuser onto a set of values. The image can be recreated in QGrover by setting the number of qubits equal to 2 and 1 as the desired value and then clicking on the first/only diffuser (depending on what is put for the number of iterations).

B.R.05 The phase oracle applies a negative phase-flip to the amplitudes of the desired values (i.e. the amplitudes of the desired values are multiplied by -1). This means that the oracle used in an instance of Grover's algorithm varies depending on the size of the value set as well as which value(s) are desired values.

B.R.06 The diffuser inverts the amplitudes about their mean. This causes the magnitude of amplitudes of the desired values to increase and the magnitudes of the other amplitudes to decrease. Calculations have been included in Figure 6.8.

B.R.07 The equation used to calculate the optimum number of iterations of the Grover iterator is $\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} \right\rfloor$, which depends on both the number of values N in the set and the number M of desired values.

B.R.08 Yes, increasing the number of solutions can reduce the number of ideal iterations. It will reduce the maximum possible amplitudes that the desired values can have.

B.R.09 Adding more iterations does not always improve Grover's algorithm. Adding any iterations beyond the number of ideal iterations may lead to the amplitudes of the solution values being reduced.

B.R.10 In special cases, is possible to obtain a probability of 100% for a desired value. This is the case, for example, when applying Grover's algorithm to a set of size $N = 4$ and the number of desired values is $M = 1$.

Because the aforementioned questions are exploratory in nature, they can be used to foster collaborative classroom discussions and group work. In addition to improving student learning outcomes by encouraging active learning, collaborative learning allows students to get feedback on their learning by seeing the perspectives of others. Currently, many computer science and software engineering curricula do not consistently prioritize collaborative learning, making it an important deficit to fill.

In addition to being a learning tool that can be used in post-secondary education spaces, QGrover can be used in secondary curricula. This is because QGrover does not require much linear algebra knowledge. While quantum computing is typically taught in upper-level post-secondary courses, introducing it in secondary schools can motivate students to pursue computer science, quantum computing, or STEM in general by showing them an exciting application of the STEM skills that they are beginning to build. We believe that QGrover could easily be integrated into curricula such as Perry et al.'s or Ivory et al.'s high school bootcamp, *QCaMP* [112, 62]. With computer science's popularity amongst high-school students and curricula steadily increasing, it is crucial to accommodate the secondary education space [28, 17].

Chapter 7

Discussion

This chapter explores how Chapters 3- 6 collectively address our five research questions posed in Chapter 1. We synthesize the findings from these chapters using two complementary perspectives: educational design and software engineering. From an educational standpoint, we review how our tools incorporate successful learning tool traits as laid out in Chapter 2, analyze the learning questions in Chapters 3 and 6, and extract preliminary feedback from Chapter 4. From a software engineering perspective, we examine how QNotation adheres to the Software Engineering Body of Knowledge (SWEBOK) [18], aligns with SOLID engineering principles [71], and meets emerging industry needs. By interpreting these results in light of our research questions and existing literature, this chapter underscores the broader implications for educators, software developers, and researchers. Finally, we discuss the limitations of our research.

7.1. Analysis of our Research Questions

The following sub-sections detail how our work presented in the previous chapters answers the research questions detailed in Chapter 1.

RQ 1. How can exploratory learning tools be designed to support the learning of foundational concepts in quantum computing education?

When designing exploratory learning tools for introductory quantum computing courses, it is important to take the variety of backgrounds students have into account. In the studies done by Hu et al. [51], Kushimo and Thacker [66], Tappert et al. [20], and Gire and Price [45], it was shown that students come from many academic backgrounds, including computer science, software engineering, physics, and chemistry. We demonstrate that QNotation and QGrover meet the traits of successful learning tools that we highlighted in Chapter 2 in the following sub-sections.

Documentation

QNotation and QGrover are documented in multiple ways. The first way is through modals within both web applications where we explain how the tools can be used and how the quantum concepts they teach work. The second way is through the `README` files, which are located in their respective GitHub repositories. QNotation also provides additional documentation through code examples, which were found to be a strong documentation trait by Nassif and Robillard [77].

Feedback

QNotation and QGrover provide two different types of feedback: visual feedback and custom error feedback when incorrect input is given. Both tools and their feedback accommodate Crisp and Bonk's six dimensions of feedback, which are timeliness, frequency, distribution, source, individualization, and content of the feedback [31]. They accommodate timeliness, frequency, and distribution by giving immediate automated feedback. They accommodate source by using noiseless classical simulators on their backends, which ensure that the math used in their visualizations is correct (and not skewed by noise) [52, 106, 47]. The feedback from both applications is individualized as the user customizes the visualizations through their input. For content, both applications provide elaborate feedback through their custom errors to help users become 'unstuck'.

Ease of Accessibility

QNotation and QGrover circumvent two common challenges highlighted by Agbo et al. [7]: deployment costs and ease of use. Because both tools are open-source, browser-based resources, users do not have to worry about licensing or installation costs. This also lowers the technical barrier for beginners, as the tools run on any modern device without requiring specialized hardware, a `Python` environment, or prior familiarity with quantum software libraries.

From a pedagogical standpoint, this helps reduce the cognitive load associated with environment setup, especially for learners with limited programming backgrounds. Instructors similarly benefit from this accessibility: the absence of installation steps allows QNotation and QGrover to be adopted in courses without consuming lecture or lab time on configuration issues, making the tools more feasible for integration into tightly scheduled curricula.

Finally, both tools incorporate design choices that support accessibility for diverse learners. The presence of modals and example inputs allows learners to begin interacting with the tools even if they are encountering quantum notations or Grover's algorithm for the first time, further supporting ease of use. The informal feedback we received on QNotation further supports this point.

Interactivity

QNotation and QGrover both offer interactive visualizations that allow users to iteratively explore the behaviour of the quantum circuits they construct [126]. In QNotation, learners can step through a circuit operation by operation in circuit, Dirac, and matrix notation, modify input code examples, and immediately observe how these changes affect the resulting quantum state. This supports the core aim of exploratory learning: enabling learners to test hypotheses, manipulate representations, and construct meaning through active engagement [121, 37].

QGrover provides a similarly interactive experience. Users can adjust the number of qubits, vary the number of marked items, and step through each iteration of the algorithm to observe how amplitudes evolve. The ability to inspect intermediate states helps learners develop intuition for amplitude amplification and the role of the oracle and diffuser in Grover's iterator.

This level of interactivity addresses a key finding in STEM education research: learners benefit when tools enable immediate, consequence-linked experimentation rather than passive observation [121]. By providing real-time, manipulable visualizations, both QNotation and QGrover support deeper conceptual understanding and promote the kind of exploratory behaviours associated with improved learning outcomes.

RQ 2. How can exploratory tools be effectively integrated into learning exercises and assessments?

As shown in Section 3.6 and Section 6.4, exploratory learning tools can be a catalyst for a variety of learning exercises. QNotation supports activities in which learners compare circuits across notations, modify existing code examples, and predict how changes to a circuit will affect its state. QGrover, in turn, enables exercises focused on algorithmic behaviour, such as exploring the effect of varying the number of marked items or iterations. These examples illustrate that exploratory tools can be integrated into both in-class activities and homework assignments as low-floor, high-ceiling tasks: learners can begin by running provided examples and then extend them to test their own hypotheses about quantum behaviour.

RQ 3. How do learners engage with and learn from such tools in practice?

We attempted to answer this question through our pilot study presented in Chapter 5 by asking the following questions:

1. Is QNotation effective for teaching the three notations commonly used in quantum computing?
2. Is QNotation a useful tool in later parts of introductory courses, once the introduction to notation is completed?

```python
class QNotationException(Exception, ABC):
    default_message: str = "An error occurred."

    def __init__(self, message: str = None):
        self.message = message if message is not None else self.default_message
        super().__init__(self.message)


class InputError(QNotationException):
    default_message = MESSAGE_INPUT_ERROR


class InvalidGateError(QNotationException):
    default_message = MESSAGE_INVALID_GATE_ERROR


class TooManyQubitsError(QNotationException):
    default_message = MESSAGE_TOO_MANY_QUBITS_ERROR


class TooManyQubitsForTensorError(QNotationException):
    default_message = MESSAGE_TOO_MANY_QUBITS_FOR_TENSOR_ERROR


class GateNotImplementedError(QNotationException):
    default_message = MESSAGE_GATE_NOT_SUPPORTED_ERROR


class HigherIndexedControlQubitError(QNotationException):
    default_message = MESSAGE_HIGHER_INDEXED_CONTROL_QUBIT_ERROR


class NonNeighbouringQubitsError(QNotationException):
    default_message = MESSAGE_NON_NEIGHBOURING_QUBITS_ERROR


class UnknownError(Exception):
    default_message = MESSAGE_UNKNOWN_ERROR


```

Figure 7.1.|: QNotation's custom error classes. The constants used are defined outside this code snippet

3. Is QNotation useful for teaching quantum computing foundations to audiences of diverse technical backgrounds?

While we were not able to get any conclusive results from our pilot offering of the study due to its low participant count, we were able to extract some anecdotes:

- **QNotation supports the exploration of introductory quantum computing concepts beyond circuit, Dirac, and matrix notation.** As shown by Table 5.3 in Chapter 5, QNotation was seen by Participant A as a tool that can be used to explore concepts beyond circuit, Dirac, and matrix notation. This is encouraged by the application through its variety of pre-made input examples.

- **Notations are challenging even for students with relevant background experience.** Table 5.1 shows that both of our study participants had taken previous courses in linear algebra, physics, and computer science. Despite this, both of them made mistakes on notation exercises in the first survey, as shown in Table 5.2. Although the survey was only taken by two participants, our findings align with the findings of Gire and Price [45], Ferryansyah et al. [43], Tunde and Thacker [66], and Hu et al. [51].

RQ 4. How can software engineering principles be incorporated into our tools?

To help us answer this question, we refer to the SWEBOK [18] and to SOLID software engineering principles [71]. The following points outline how QNotation satisfies the SWEBOK chapters relevant to it:

- **Software Requirements:** The requirements for QNotation were elicited through real-world observation, academic literature review, and exploratory prototyping.
- **Software Architecture:** After noticing UI constraints in QNotation's prototype, we ensured that QNotation was architected in a way that made the application more flexible. This resulted in an increase in the number of qubits that can be used. We also prioritized making the application maintainable and scalable by using well-documented libraries and by making modular components.
- **Software Design and Construction:** QNotation's backend was created using object-oriented methodologies, while its frontend focuses on component-based strategies. This was done to reduce repetition and increase modularity in the code bases with the overarching goal to keep the application's complexity at a minimum. Both repositories also focus on error handling as a way to provide users with informative feedback, which is one of the key traits of successful learning tools that we previously identified.
- **Software Testing:** QNotation's test suites and CI/CD pipeline follow the SWEBOK's software testing guidelines by running its test suites frequently (*dynamic testing*), by asserting that correct results are produced by the application (*expected outputs*), and by containing a finite number of test cases (*finiteness and explicit test cases*) that test relevant and risky behaviour within the application (*selected*).
- **Software Maintenance:** QNotation's CI/CD pipeline ensures that the application is properly maintained by regularly running its test suites and linting any code changes made using GitHub Actions.
- **Software Configuration Management:** QNotation's `requirements.txt` asserts that the most recent versions of the quantum libraries implemented in QNotation are used.
- **Software Models and Methods:** ABCs are used within QNotation to assert how its parser factory must be implemented as well as the test cases that must be implemented within each test class.

The following points give examples of how QNotation incorporates SOLID principles:

- **Single Responsibility Principle:** QNotation's `GateInformation` class' single responsibility is the production of objects that can be used to read operation data in a way that is libary agnostic.

- **Open/Closed Principle:** ABCs are used within QNotation to ensure that classes are extensible, but not modifiable. Examples of these include the `Parser` ABC and the `TestClass` ABC.
- **Liskov Substitution Principle:** The error classes, shown in Figure 7.1, are examples of the Liskov Substitution principle.
- **Interface Segregation Principle:** QNotation contains several concise interfaces. These include `Parser`, `QNotationException`, and `TestClass`.
- **Dependency Inversion Principle:** The `GateInformation` class makes it so that low-level, library-specific details of a given gate are hidden.

RQ 5. How can our tools be built to align with industry needs?

Our tools align with industry needs in multiple ways. The first is through the concepts they teach: notation comprehension and Grover's algorithm. Both address the requirement of foundational quantum knowledge needed for quantum jobs, and QNotation contributes to helping with cross-functional team collaboration due to the notations being used differently across disciplines [130].

QNotation further contributes to users developing a solid skill set of quantum foundations by using popular quantum libraries as input. Many quantum companies list experience with these libraries as desirable for new hires [30, 130, 39]. QNotation's parser library helps future proof this as new libraries can be added to the application with only a few steps. The details on how this can be done are listed in Section 3.5. Thanks to the majority of the steps taken on the QNotation backend being library-agnostic, ensuring that the latest versions of the supported libraries are used will not require any major refactoring of the application.

7.2. Implications

The work in this thesis impacts educators, software developers, and researchers. For educators, this work provides two exploratory learning tools: QNotation and QGrover. In addition to being relevant to introductory quantum computing curricula, these tools are effective due to their strong documentation, frequent feedback, accessibility, and interactivity. It also provides sample exercises for both tools (Section 6.4 and Section 3.6). These exercises can be used as is or can be built upon. Lastly, our work provides anecdotes on how QNotation can be used throughout the offering of an introductory quantum computing course, while also highlighting the different skills learners may be coming into introductory courses with.

For software developers, this work shows the use of quantum libraries within full-stack software applications. It also provides details on how these applications can be further extended and contributed to. QNotation's CI/CD pipeline and test suites highlight how

consistency, reliability, and reproducibility can be supported in an open-ended exploratory application, illustrating how SWEBOK-inspired practices can be applied in an educational context [18].

For researchers, our work can provide inspiration for future learning tools as well as feedback anecdotes and survey templates. Through the creation of our two tools, we were able to show that it is possible to create effective exploratory learning tools for supporting the learning of introductory quantum computing concepts. While the survey templates, Appendix F, Appendix G, and Appendix H, were originally created for studying QNotation, we believe that they provide a great starting point for creating studies for other tools due to how they prioritize analyzing participants' academic background, how they use the tool over the course of a semester, and retrieving qualitative data on participants' experiences.

7.3. Limitations

The following sub-sections highlight the limitations of our contributions:

7.3.1. QNotation

One of QNotation's main limitations is that no more than 5 qubits can be used. As shown by our study, 5 qubits is enough to allow for experimentation with a variety of foundational concepts, however; greater qubit support would allow for the running of more complex real-world quantum code.

Another limitation is the need for multi-qubit gates to be applied to ascending and neighbouring qubits only. This means that users cannot fully explore algorithms such as Quantum Phase Estimation due to the three-qubits or more versions requiring non-neighbouring qubit gates to be used [107].

7.3.2. QNotation CI/CD Pipeline and Test Suites

While QNotation's test suites test all supported gates, it does not test all possible gate combinations. Gates are primarily tested in isolation, or with one or two other gates. While this is good for pinpointing erroneous behaviour within gates, it may lead to erroneous edge cases not being discovered. Parametrized gates are tested with a set number of angles: π, $\frac{\pi}{2}$, and $\frac{\pi}{4}$. This means that erroneous behaviour with other angles will not be caught.

7.3.3. QNotation Study

The QNotation study was primarily limited by its small number of participants and unfinished surveys. While the data received was able to point us to some preliminary results, formal analysis could not be done on them.

7.4. Discussion Summary

In this chapter, we reflected on how QNotation, its testing framework, and its pilot study; and QGrover collectively address the five research questions posed in Chapter 1. From an educational perspective, we showed how these tools embody key traits of successful learning technologies: rich documentation, timely and individualized feedback, accessibility, and interactivity; as well as how they can be integrated into learning activities as low-floor, high-ceiling tasks. From a software engineering perspective, we discussed how QNotation follows relevant SWEBOK chapters and SOLID principles while remaining aligned with emerging industry needs through its use of widely adopted quantum libraries and extensible parser architecture. We also outlined the implications of this work for educators, software developers, and researchers, and highlighted key limitations related to qubit and gate constraints, test coverage, study methodology, and the current scope of QGrover. Together, these reflections position QNotation and QGrover as concrete, extensible examples of how exploratory tools for quantum computing can be designed, engineered, and studied in practice.

Chapter 8

Conclusion and Future Work

This final chapter synthesizes the contributions of this thesis and outlines promising directions for future development. Throughout this work, we presented QNotation, its testing and verification framework, a pilot study on QNotation and its methodology, and QGrover. While these contributions collectively advance both the pedagogical and technical landscape of quantum education, they also highlight opportunities for further research, expansion, and refinement. In the sections that follow, we first discuss future work for extending our contributions, especially in terms of visualization breadth, software architecture, and algorithmic coverage. We then propose next steps for empirical evaluation through additional user studies, before concluding with a summarization of how this thesis answers the research questions posed in Chapter 1.

8.1. Contributions

This thesis set out to investigate how exploratory learning tools can be designed, engineered, and evaluated to support introductory quantum computing education. The contributions of this thesis demonstrate that well-designed exploratory tools can meaningfully enhance how students learn quantum computing. As the field continues to expand, such tools will play a crucial role in making quantum computing education more accessible, intuitive, and scalable. As quantum computing continues to evolve into an applied discipline, there remains a clear need for accessible learning technologies and empirically grounded insights into how students engage with them. Through the development of QNotation, the accompanying QNotation testing framework, the QNotation user study, and the creation of QGrover, this work provides both theoretical and practical contributions to that need.

First, QNotation demonstrates how an exploratory visualization environment can help learners build intuition across the three major notations used in quantum computing: circuit, Dirac, and matrix notation. By integrating multiple quantum libraries and offering an interactive, beginner-friendly interface, QNotation addresses long-standing challenges in helping students connect abstract mathematical representations to operational circuits. Its design is informed by principles from prior literature on effective learning tools.

Second, the QNotation testing framework shows how software engineering rigor can be embedded into educational tools from the outset. The framework ensures correctness, prevents regressions, and provides reproducibility, which are essential requirements when conceptual accuracy directly affects learning outcomes. This work contributes a model for how quantum educational software can be engineered to remain trustworthy as it evolves.

Third, the pilot study provides some early evidence of how learners interact with QNotation. These preliminary findings reinforce the value of exploratory visualization environments in quantum computing education and offer insight into how such tools may be integrated into learning assessments. The survey templates detailed allow for the study to be replicated with QNotation in the future while also providing inspiration for future studies.

Finally, QGrover demonstrates how the methodology used in QNotation can be extended to algorithm-specific learning tools. By allowing learners to inspect state amplitudes step-by-step through Grover's algorithm, QGrover supports more intuitive understanding of oracle behaviour, diffusion, and amplitude amplification concepts.

To help motivate this thesis, we asked 5 research questions. The following points summarize our contributions to them

1. **How can exploratory learning tools be designed to support the learning of foundational concepts in quantum computing education?** QNotation and QGrover illustrate that intuitive, multi-notation, interactive tools can reduce conceptual barriers and promote hypothesis testing and self-directed learning.

2. **How can exploratory tools be effectively integrated into exercises and homework assignments?** The learning questions in Chapters 3 and 6 demonstrate concrete examples of how instructors can embed exploratory tasks into curricula.

3. **How do learners engage with and learn from such tools in practice?** The anecdotes received from the QNotation pilot study as well as informal feedback point to QNotation being a tool that learners and educators alike are would like to use to explore a variety of introductory quantum concepts.

4. **How can good software engineering principles be incorporated into our tools?** The construction of QNotation and its testng framework are grounded in chapters from the SWEBOK and SOLID principles, illustrating how educational tools can be engineered for correctness and maintainability.

5. **How can our tools be built to align with industry needs?** QNotation and QGrover align with emerging industry needs by teaching foundational quantum skills, notation fluency and Grover's algorithm, and by supporting widely used quantum libraries. QNotation's modular, library-agnostic design further future-proofs the tool, allowing new industry-standard libraries to be integrated with minimal changes.

8.2. Future Work

The contributions presented in this thesis open several promising avenues for future development, expansion, and evaluation. Our aim for this section is to provide points on how to strengthen the pedagogical usefulness of QNotation and QGrover while also broadening their reach as scalable, industry-aligned, and rigorously engineered educational technologies. The following subsections outline concrete next steps for creating additional tools inspired by QGrover, expanding QNotation, and deepening the research agenda through further user studies.

8.2.1. Quantum Algorithms Repository

We believe that creating an interactive quantum algorithm visualization repository, similar to that of Data Structure Visualizations repository [156] from the University of San Francisco and the Visualgo repository [96] from the National University of Singapore. In the following sub-sections we highlight how different quantum protocols and algorithms could be made into applications similar to QNotation, QWalkVis, and QGrover:

8.2.1.1. Superdense Coding

For superdense coding, we recommend its application having two modes: a 'standard' mode and an 'atypical-Alice' mode. The standard mode will show superdense coding how it is typically described in textbooks. Users will choose which Bell state they would like to be sent from Alice to Bob, then they will see the corresponding gates applied to the circuit. Once users are comfortable with the standard version of superdense coding, they can then switch to the atypical-Alice mode of the application. The atypical-Alice mode changes which quantum gates she applies. The task for the user is to figure out which quantum gates Bob should be applying. QNotation would be a great complementary tool for this task as it would allow users to recreate the atypical-Alice circuits and trial and error which gates Bob should apply.

8.2.1.2. Quantum Phase Estimation

For Quantum Phase Estimation, we recommend retrieving two inputs from users: the number of estimation qubits and the desired angle to be estimated. This will allow users to see how the number of estimation qubits dictates the precision of the algorithm's angle estimation. Like with QGrover, we believe that users should be able to click through each component of the algorithm. Being able to see Quantum Phase Estimation as a circuit will allow users to see how adding more estimation qubits exponentially increases the number of controlled gates used by the algorithm.

8.2.2. QNotation

Future work for QNotation should include greater qubit support, increased library support, and increased gate support. We detail how to approach these pieces of work in the following sub-sections:

8.2.2.1. Greater Qubit Support

QNotation currently supports quantum circuits with five qubits or less. While many quantum concepts, protocols, and algorithms can be created with five qubits or less, supporting more qubits will allow for more complex quantum circuits to be visualized. Inspiration could be taken from IBM Quantum's Composer, where each visualization supports a different number of qubits [53]. For QNotation, the matrix notation component could be limited at a lower number of qubits than the circuit and Dirac components due to it being the most verbose component.

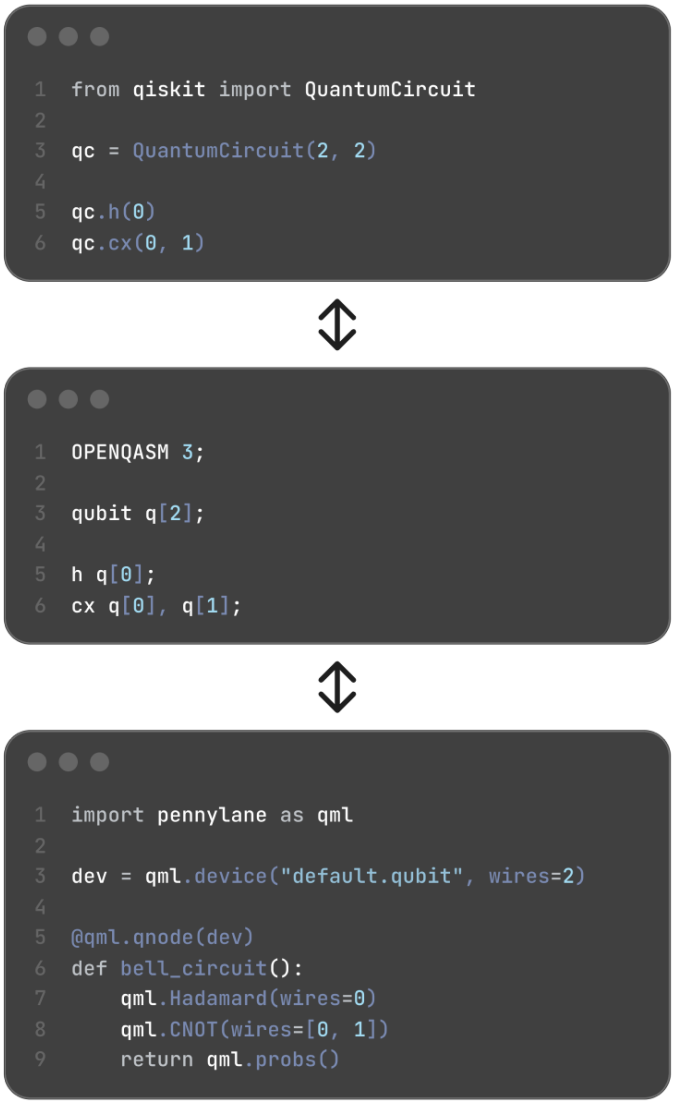
8.2.2.2. Library Support

Using the steps outlined in Sections 3.5, more libraries can be added to QNotation. Some libraries that could be added include `PyQuil` [123] and `QuTiP` [118]. In addition to providing more support, direct translation between libraries (ex. being able to create a quantum circuit in `PennyLane` then be able to view it written in other libraries) could also be implemented. On the backend, this would likely involve converting the given quantum circuit into `OpenQASM 3`, IBM Quantum's assembly language [29], then converting it into the desired library. `Qiskit` [115], `PennyLane` [163], and `Cirq` [8] all support the conversion of their quantum circuits to `OpenQASM 3`.

8.2.2.3. Increased Gate Support

Future iterations of QNotation should prioritize non-neighbouring gate support and varying-sized gate support. The support of non-neighbouring gates will help improve QNotation's ability to support the learning of quantum algorithms. For example, the support of non-neighbouring gates will allow for users to implement algorithms like Quantum Fourier Transform, which can require several non-neighbouring-qubit controlled rotations [10].

Supporting varying-sized gates, such as `PauliRot` for `PennyLane`, will improve QNotation's alignment with industry standards. While gates such as `PauliRot` can be recreated using other gates, as shown in Figure 8.2, allowing them to be used will help QNotation reflect the full capabilities of the libraries implemented within it.



```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)

qc.h(0)
qc.cx(0, 1)
```

```
OPENQASM 3;

qubit q[2];

h q[0];
cx q[0], q[1];
```

```python
import pennylane as qml

dev = qml.device("default.qubit", wires=2)

@qml.qnode(dev)
def bell_circuit():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    return qml.probs()
```

Figure 8.1.|: Conversion between `Qiskit`, `OpenQasm 3`, and `PennyLane`.

A.

```python
import pennylane as qml
from pennylane import numpy as np

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def use_paulirot():
    qml.PauliRot(np.pi/4, "X", wires=[0])
    return qml.probs()
```

B.

```python
import pennylane as qml
from pennylane import numpy as np

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def paulirot_equivalent():
    qml.Hadamard(0)
    qml.RZ(np.pi/4, 0)
    qml.Hadamard(0)
    return qml.probs()
```

Figure 8.2.|: Equivalent `PennyLane` circuits where A shows a `PauliRot` gate being applied, and B shows an equivalent operation. `PauliRot` is not currently supported in QNotation, but `Hadamard` and `RZ` are supported.

8.2.3. QNotation User Study

Subsequent studies should be done on QNotation to help inform its future development. The study presented in Chapter 5 can be used to add more data to the existing dataset. We believe that courses like the University of Victoria and the Quantum Algorithms Institute's Quantum Software Engineering Bootcamp [60] would be an excellent candidate due to its curriculum is similar to that of the Quantum Algorithms and Software Engineering course [101] and due to it being offered on a rolling basis. To receive more immediate feedback on the application, a short survey could be added to the frontend. For this, we propose the following survey:

1. What are you using QNotation for today? (long answer)
2. What is your academic or professional background? (long answer)
3. Please rate the following prompts from 1-10, where 1 would be "Completely Disagree" and 10 would be "Completely Agree"
    a) QNotation helped me learn about circuit, Dirac, and matrix notation (1-10 rating)
    b) QNotation helped me translate between circuit, Dirac, and matrix notation (1-10 rating)
    c) QNotation made my studying more efficient (1-10 rating)
4. What do you like about QNotation? (long answer)
5. What do you not like about QNotation (long answer)

As quantum computing education continues to mature, the tools, frameworks, and findings presented in this thesis offer a foundation that is both practical and forward-looking. QNotation, QGrover, and the methodologies developed alongside them are intended not as static artifacts, but as evolving platforms that will grow with the needs of learners, educators, and industry. The future work outlined above points toward a rich landscape of possibilities from expanded algorithmic visualizations, to deeper empirical evaluation, to more robust software systems. Taken together, these directions chart a path toward a future where exploratory, interactive quantum learning tools are widely accessible, seamlessly integrated into curricula, and engineered with the same rigor as the technologies they aim to teach. While this thesis marks the completion of one stage of that journey, it also represents the starting point for continued development, collaboration, and discovery in the broader effort to make quantum computing education intuitive, engaging, and actionable for all.

Bibliography

[1] CodePen. codepen.io. Accessed November 2025.

[2] CodeSandbox. codesandbox.io. Accessed November 2025.

[3] From Qubits to Quantum Teleportation: A Hands-On Experience for High Schoolers, author="angara, prashanti priya and stege, ulrike and maclean, andrew and markham, tom and knodel, j", booktitle=IEEE Quantum Week 2020 Workshop Abstracts, volume=1, pages=466-487, year=2020, organization=IEEE.

[4] JSFiddle - Code Playground. jsfiddle.net. Accessed November 2025.

[5] Five years ago today, we put the first quantum computer on the cloud. Here's how we did it. www.ibm.com/quantum/blog/quantum-five-years, 2021. Accessed November 2025.

[6] QMsolve: A module for solving and visualizing the Schrödinger equation. github.com/quantum-visualizations/qmsolve, 2025. Accessed November 2025.

[7] Benjamin Agbo, Cerris Morris, Osman Mogdam, Joe Basketts, and Theocharis Kyriacou. A systematic literature review on software applications used to support curriculum development and delivery in primary and secondary education. *Review of Educational Research*, 2024.

[8] Google Quantum AI. Import/export circuits. quantumai.google/cirq/build/interop, 2025. Accessed November 2025.

[9] AlgoMonster. AlgoMonster. algo.monster/, 2024. Accessed November 2025.

[10] Guillermo Alonso. Intro to Quantum Fourier Transform. pennylane.ai/qml/demos/tutorial_qft, 2025. Accessed November 2025.

[11] Prashanti Priya Angara, Saasha Joshi, Ulrike Stege, Ghislain Lefebvre, and Jean-Frédéric Laprade. Demystifying the Quantum Enigmas: A Hands-on Introduction to Quantum Computing. qce.quantum.ieee.org/2023/workshops-program/#wks02. Accessed November 2025.

[12] Prashanti Priya Angara, Paria Naghavi, Saasha Joshi, Ulrike Stege, and Curtis Volin. Careers in Quantum Computing: how to get started with quantum computing—a workshop for highschoolers. In *IEEE International Conference on Quantum Computing and Engineering (QCE22)*. IEEE, 2021.

[13] Prashanti Priya Angara, Ulrike Stege, Andrew MacLean, Tom Markham, and Curtis Volin. Pathways to Quantum: An Introductory Workshop on Quantum Computing for Youth. In *IEEE Quantum Week 2021 Workshop Abstract*. IEEE, 2021.

[14] Prashanti Priya Angara, Ulrike Stege, Andrew MacLean, Hausi A. Müller, and Tom Markham. Teaching quantum computing to high-school-aged youth: A hands-on approach. *IEEE Transactions on Quantum Engineering*, 3:1–15, 2022.

[15] AWS. Amazon Braket. aws.amazon.com/braket/, 2025. Accessed November 2025.

[16] Axios contributors. Axios. axios-http.com/docs/intro. Accessed November 2025.

[17] BCS the Chartered Institute for IT. Record numbers of students choose Computer Science A Level in 2022. 2022.

[18] Pierre Bourque and Richard E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition, 2014. Accessed: 2025-03-XX.

[19] Josephine C. Meyer, Gina Passante, Steven J. Pollock, and Bethany R. Wilcox. Introductory quantum information science coursework at US institutions: Content coverage. *ArXiv*, 2023.

[20] Charles C. Tappert, Ronald I. Frank, Istvan Barabasi, Avery M. Leider, Daniel Evans, and Lewis Westfall. Experience Teaching Quantum Computing. In *2019 ASCUE Proceedings*. 2019 ASCUE Proceedings, 2019.

[21] Carleton. Computer Science B.C.S. Major (20.0 credits). calendar.carleton.ca/undergrad/undergradprograms/computerscience, 2025. Accessed November 2025.

[22] Andrew Chu, Dea Biancarelli, Mari-Lynn Drainoni, James H Liu, Jeffrey I Schneider, Ryan Sullivan, and Alexander Y Sheng. Usability of Learning Moment: Features of an E-learning Tool That Maximize Adoption by Students. *West J Emerg Med*, 21, 2019.

[23] Ruth C. Clark and Richard E. Mayer. *E-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning*. John Wiley & Sons, Hoboken, NJ, 4 edition, 2016.

[24] Classiq. Quantum Program Visualization Tool. docs.classiq.io/latest/user-guide/analysis/quantum-program-visualization-tool/, 2025. Accessed November 2025.

[25] Camosun College. Information and Computer Systems Technology (Diploma). calendar.camosun.ca/preview_program.php?catoid=24&poid=3818, 2025. Accessed November 2025.

[26] Langara College. Associate of Science Degree in Computer Science. langara.ca/programs-and-courses/programs/computer-science/program-curriculum.html, 2025. Accessed November 2025.

[27] Okanagan College. COURSE PLANNING SHEET for Bachelor of Computer Information Systems Degree. www.okanagan.bc.ca/sites/default/files/2025-01/BCIS%20from%202022.pdf, 2022. Accessed November 2025.

[28] Computing Research Association. Cra releases report on surge in computer science enrollments. 2021.

[29] OpenQASM contributors. OpenQASM Live Specification. openqasm.com/, 2025. Accessed November 2025.

[30] Council of Canadian Academies (CCA). Quantum Potential, Ottawa (ON): Expert Panel on the Responsible Adoption of Quantum Technologies. 2023.

[31] Erin A. Crisp and Curtis J. Bonk. Defining the Learner Feedback Experience. *TechTrends*, 2018.

[32] D-WAVE. QUANTUM COMPUTING WITH D-WAVE. www.dwavequantum.com/media/sythedzv/d-wave_quantum-programming-core_syllabus_v3-1.pdf, 2025. Accessed November 2025.

[33] Université de Montréal. IFT 3155/6155 Informatique quantique H22. www.iro.umontreal.ca/~brassard/cours/6155.pdf, 2022. Accessed November 2025.

[34] Université de Montréal. Baccalauréat en informatique Structure du programme. admission.umontreal.ca/programmes/baccalaureat-en-informatique/structure-du-programme/, 2025. Accessed November 2025.

[35] Université de Sherbrooke. BSQ 101 – Projets intégrateurs en programmation quantique. plandecours.dinf.usherbrooke.ca/pdc/2025-1/BSQ101/1/, 2025. Accessed November 2025.

[36] Pascal Debus, Sebastian Issel, and Kilian Tscharke. Quantum Machine Learning Playground. *IEEE Computer Graphics and Applications*, 2024.

[37] Marci Decaro, Raina A. Isaacs, Campbell R. Bego, and Raymond Chastain. Bringing exploratory learning online: problem-solving before instruction improves remote undergraduate physics learning. *Frontiers in Education*, 2023.

[38] DistanceLearning.Institute. Characteristics of Effective Learning Resources: A Guide for Educators. distancelearning.institute/instructional-design/characteristics-effective-learning-resources-guide-educators/, 2024. Accessed November 2025.

[39] Jake Douglass, Susan Schwamberger, Donn Silberman, Connor Teague, Corban Tilleman-Dick, Evangeline Williams, and Newry Corp. Guide to Building a Quantum Technician Workforce. 2024.

[40] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. A Grover-search based quantum learning scheme for classification. *new Journal of Physics*, 2021.

[41] Times Higher Education. Top Universities in Canada 2026. www.timeshighereducation.com/student/best-universities/best-universities-canada, 2025. Accessed November 2025.

[42] Encode. HTTPX. www.python-httpx.org. Accessed November 2025.

[43] Ferryansyah, E Widyawati, and S W Rahayu. The analysis of students' difficulty in learning linear algebra. In *International Conference on Statistics, Mathematics, Teaching, and Research*, volume 2. Journal of Physics: Conference Series, 2018.

[44] Thomas G Wong. Introduction to Classical and Quantum Computing. pages 140–141. Rooted Grove, 2022.

[45] Elizabeth Gire and Edward Price. Structural features of algebraic quantum notations. *Physical Review Physics Education Research*, 11(2), 2015.

[46] GitHub. GitHub Actions. github.com/features/actions, 2025. Accessed November 2025.

[47] Google Quantum AI. cirq.Simulator. quantumai.google/reference/python/cirq/Simulator. Accessed November 2025.

[48] Google Quantum AI. Cirq. quantumai.google/cirq, 2025. Accessed November 2025.

[49] Google Quantum AI. Codebook. quantumai.google/resources, 2025. Accessed November 2025.

[50] Lov k. Grover. A fast quantum mechanical algorithm for database search. *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, 1996.

[51] Peter Hu, Yangqiuting Li, and Chandralekha Singh. Investigating and improving student understanding of the basics of quantum computing. *Phys. Rev. Phys. Educ. Res.*, 20:020108, Aug 2024.

[52] IBM Quantum. Aer - high performance quantum circuit simulation for Qiskit. github.com/Qiskit/qiskit-aer. Accessed November 2025.

[53] IBM Quantum. Composer. quantum-computing.ibm.com/composer, 2023. Accessed November 2025.

[54] IBM Quantum. IBM Quantum Platform. quantum.cloud.ibm.com/, 2025. Accessed November 2025.

[55] IBM Quantum. Learn Quantum Computing. quantum.cloud.ibm.com/learning/en, 2025. Accessed November 2025.

[56] IBM Quantum. Qiskit. qiskit.org, 2025. Accessed November 2025.

[57] IBM Quantum. UnitaryGate. quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.circuit.library.UnitaryGate, 2025. Accessed November 2025.

[58] IBM Quantum. Visualizations. quantum.cloud.ibm.com/docs/en/api/qiskit/visualization, 2025. Accessed November 2025.

[59] IEEE Quantum Week Committee. IEEE Quantum Week 2025. qce.quantum.ieee.org/2025/#, 2025. Accessed November 2025.

[60] Quantum Algorithms Institute. QAI Quantum Software Engineering Bootcamp. www.qai.ca/qseb. Accessed November 2025.

[61] ionel. pytest-cov. pypi.org/project/pytest-cov/. Accessed November 2025.

[62] Megan Ivory and et al. Quantum computing, math, and physics (qcamp): Introducing quantum computing in high schools. *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2023.

[63] Stacey Jeffrey. Quantum Subroutine Composition. *arXiv*, 2022.

[64] Addie Jordon. Step-by-Step: Quantum Walk Implementations and Visualizations, 2023.

[65] Addie Jordon, Austin Hawkins-Seagram, Samantha Norrie, José Ossorio, and Ulrike Stege. QWalkVis: Quantum Walks Visualization Application. *IEEE International Conference on Quantum Computing and Engineering*, 2023.

[66] Tunde Kushimo and Beth Thacker. Investigating Students' Strengths and Difficulties in Quantum Computing. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 03, pages 33–39, 2023.

[67] Université Laval. Structure du programme. www.ulaval.ca/etudes/programmes/baccalaureat-en-informatique#section-structure, 2025. Accessed November 2025.

[68] Peter Y. Lee, James M. Yu, and Ran Cheng. *Mathematical Foundations of Quantum Computing: A Scaffolding Approach*. Polaris, 2025.

[69] Ghislain Lefebvre, Maxime Dion, Jean Frederic Laprade, Tom Mallah, et al. The Quantum Enigmas. In *APS March Meeting Abstracts*, volume 2022, pages T27–009, 2022.

[70] Florence Martin, Lynn Ahlgrim-Delzell, and Kiran Buhrani. Systematic Review of Two Decades (1995 to 2014) of Research on Synchronous Online Learning. *American Journal of Distance Education*, 31:3–19, Jan 2017.

[71] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, Upper Saddle River, NJ, 2003.

[72] Miguel Martín-Sómer, Cintia Casado, and Gema Gómez-Pozuelo. Utilising interactive applications as educational tools in higher education: Perspectives from teachers and students, and an analysis of academic outcomes. *Education for Chemical Engineers*, 46, 2023.

[73] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The Impact of Quantum Computing on Present Cryptography. *ArXiv 1804.00200*, 2018.

[74] Meta. React-Vis: A Composable Chartin Library. uber.github.io/react-vis/. Accessed November 2025.

[75] Microsoft. Build quantum solutions with the Azure Quantum Development Kit. learn.microsoft.com/en-us/azure/quantum/qdk-main-overview, 2025. Accessed November 2025.

[76] MUI. MUI: The React Component Library You Always Wanted. mui.com. Accessed November 2025.

[77] Mathieu Nassif and Martin P. Robillard. A Field Study of Developer Documentation Format. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[78] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2011.

[79] Samantha Norrie and Anthony Estey. QNotation: An Interactive Visual Tool to Lower Learning Barriers in Quantum Computing. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 2, pages 373–374. IEEE, 2023.

[80] Samantha Norrie, Anthony Estey, Hausi Müller, and Ulrike Stege. QGrover: Teaching Grover's Algorithm Through Visual Exploration. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 03, pages 17–24, 2024.

[81] Samantha Norrie, Anthony Estey, Hausi Müller, and Ulrike Stege. QNotation: A Visual Browser-Based Notation Translator for Learning Quantum Computing. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 03, pages 25–33, 2024.

[82] University of Alberta. CMPUT 604 - Quantum Computing for Computing Scientists. www.ualberta.ca/en/computing-science/graduate-studies/course-directory/courses/quantum-computing-for-computing-scientists.html, 2025. Accessed November 2025.

[83] University of Alberta. Major in Computing Science Requirements. calendar.ualberta.ca/preview_program.php?catoid=44&poid=55570#MajorInComputingScienceRequirements, 2025. Accessed November 2025.

[84] University of British Columbia. Computer Science. vancouver.calendar.ubc.ca/faculties-colleges-and-schools/faculty-science/bachelor-science/computer-science, 2024. Accessed November 2025.

[85] University of British Columbia. CPEN 416 Gate-Model Quantum Computing. ece.ubc.ca/courses/cpen-416/, 2025. Accessed November 2025.

[86] University of British Columbia (Okanagan). B.Sc. Major in Computer Science. okanagan.calendar.ubc.ca/faculties-schools-and-colleges/faculty-science/bachelor-science-programs/major-programs/computer-science-bsc#19088, 2025. Accessed November 2025.

[87] University of Calgary. Program Sequencing Guide (Computer Science, Fall 2025). science.ucalgary.ca/sites/default/files/teams/1/USC/program-guides/F25-CPSC-Guide.pdf, 2025. Accessed November 2025.

[88] University of Lethbridge. Computer Science (BSc). www.ulethbridge.ca/career-bridge/computer-science-bsc. Accessed November 2025.

[89] University of Manitoba. Computer Science. umanitoba.ca/science/computer-science. Accessed November 2025.

[90] Memorial University of Newfoundland. Physics 4852: Quantum Information and Computing. www.mun.ca/physics/undergraduates/syllabus/physics-4852-quantum-information-and-computing/. Accessed November 2025.

[91] University of Northern British Columbia. Major in Computer Science. www.unbc.ca/calendar/undergraduate/computer-science#major-in-computer-science, 2025. Accessed November 2025.

[92] University of Ottawa. Major in Computer Science. www.uottawa.ca/faculty-engineering/undergraduate-studies/programs/computer-science/course-sequence, 2025. Accessed November 2025.

[93] University of Ottawa. MAT 4341 QUANTUM COMPUTING (3 UNITS). catalogue.uottawa.ca/en/courses/mat/, 2025. Accessed November 2025.

[94] University of Regina. MATH 416 - Introduction to Quantum Information Theory. banner.uregina.ca:17023/ssbprod/bwckctlg.p_disp_course_detail?cat_term_in=202530&subj_code_in=MATH&crse_numb_in=416. Accessed November 2025.

[95] University of Saskatchewan. Computer Science. programs.usask.ca/arts-and-science/computer-science/bsc-4-computer-science.php#top, 2025. Accessed November 2025.

[96] National University of Singapore. VisuAlgo: visualising data structures and algorithms through animation. visualgo.net/en, 2025. Accessed November 2025.

[97] University of the Fraser Valley. Computing Science (COMP) Course Outlines. www.ufv.ca/calendar/CourseOutlines/PDFs/COMP/, 2025. Accessed November 2025.

[98] University of the Fraser Valley. Computing science major. www.ufv.ca/calendar/current/ProgramsC-E/COMP_MAJOR.html, 2025. Accessed November 2025.

[99] University of Toronto. PHY365H1: Quantum Information. artsci.calendar.utoronto.ca/course/phy365h1, 2025. Accessed November 2025.

[100] University of Victoria. BSc in Computer Science, Major. www.uvic.ca/students/undergraduate/program-planning/program-worksheets/worksheets/ppw-ecs-csci-major.pdf, 2024. Accessed November 2025.

[101] University of Victoria. SENG457: Quantum Algorithms and Software Engineering. heat.csc.uvic.ca/coview/course/2024051/SENG457, 2024. Accessed November 2025.

[102] University of Waterloo. Computer Science Degree. uwaterloo.ca/future-students/programs/computer-science, 2025. Accessed November 2025.

[103] University of Waterloo. CS 467 Introduction to Quantum Information Processing. cs.uwaterloo.ca//current/courses/course_descriptions/cDescr/CS467, 2025. Accessed November 2025.

[104] University of Windsor. Computer Science (Honours). www.future.uwindsor.ca/program/computer-science/. Accessed November 2025.

[105] Francisco José Orts, Gómez, Gloria Ortega Lopez, N.C. Cruz, and E M Garzon. UNDERSTANDING GROVER'S SEARCH ALGORITHM THROUGH A SIMPLE CASE OF STUDY. *11th International Conference on Education and New Learning Technologies*, pages 1730–1737, July 2019.

[106] PennyLane. default.qubit. pennylane.ai/devices/default-qubit. Accessed November 2025.

[107] PennyLane. Intro to Quantum Phase Estimation. pennylane.ai/qml/demos/tutorial_qpe. Accessed November 2025.

[108] PennyLane. What are Bell states? pennylane.ai/qml/glossary/what-are-bell-states. Accessed November 2025.

[109] PennyLane. What is the no-cloning theorem? pennylane.ai/qml/glossary/what-is-the-no-cloning-theorem/. Accessed November 2025.

[110] PennyLane. PennyLane. www.xanadu.ai/products/pennylane/#documentation, 2025. Accessed November 2025.

[111] PennyLane. Quantum Operations. https://docs.pennylane.ai/en/stable/introduction/operations.html, 2025. Accessed November 2025.

[112] Anastasia Perry, Ranbel Sun, Ciaran Hughes, Joshua Isaacson, and Jessica Turner. Quantum Computing as a High School Module. *ArXiv Physics Education*, 2019.

[113] Deyana Peykova and Kosta Garov. DIGITAL TOOLS FOR STEM EDUCATION. *Anniversary International Scientific Conference RESEARCH AND EDUCATION IN MATHEMATICS, INFORMATICS AND THEIR APPLICATIONS*, 2021.

[114] IBM Quantum. Bit-ordering in the Qiskit SDK. quantum.cloud.ibm.com/docs/en/guides/bit-ordering. Accessed November 2025.

[115] IBM Quantum. Exporting to OpenQASM 3. quantum.cloud.ibm.com/docs/en/api/qiskit/qasm3, 2025. Accessed November 2025.

[116] QuEra. Quantum Noise. www.quera.com/glossary/noise, 2025. Accessed November 2025.

[117] QuTiP. Exercises. qutip.org/qutip-tutorials/#visualizations, 2025. Accessed November 2025.

[118] QuTiP. QuTiP. www.npmjs.com/package/@monaco-editor/react, 2025. Accessed November 2025.

[119] Anish Raj, Alaukik Deep, and Rishi Chopra. A Review Of Enhanced Online Live Code Editors. *Proceedings of the KILBY 100 7th International Conference on Computing Sciences 2023 (ICCS 2023)*, 2023.

[120] React. @monaco-editor/react.

[121] John Rieman. A field study of exploratory learning strategies. *ACM Trans. Comput.-Hum. Interact.*, 3(3):189–218, September 1996.

[122] Rigetti. Exercises. pyquil-docs.rigetti.com/en/stable/exercises.html, 2025. Accessed November 2025.

[123] Rigetti. pyQuil. pyquil-docs.rigetti.com/en/stable/, 2025. Accessed November 2025.

[124] Rigetti Computing. Quantum Programming Studio. quantum-circuit.com, 2025. Accessed November 2025.

[125] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Professiona, 2010.

[126] Lisa Shappee. Characteristics to Consider When Evaluating Educational Technology Tools". *C2C Digital Magazine (Spring / Summer 2019)*, 2019.

[127] Alexander Shvets. Dive Into Design Patterns. pages 97–107. Refactoring.Guru, Redwood City, CA, USA, 2018.

[128] Meta Open Source. React. react.dev. Accessed November 2025.

[129] SurveyMonkey. SurveyMonkey. www.uvic.ca/systems/services/pages/surveymonkey.php, 2025. Accessed November 2025.

[130] Louise Turner and Yoan Martha. The Canadian Quantum Ecosystem Report 2024. 2024.

[131] Unitary Foundation. 2024 Quantum Open Source Survey. unitaryfoundation.github.io/survey-2024/. Accessed November 2025.

[132] Brock University. Computer Science – Academic Programs. brocku.ca/programs/undergraduate/computer-science/. Accessed November 2025.

[133] Carleton University. Quantum Computing and Information COMP 4114. outline.scs.carleton.ca/media/2024/F/COMP4114AF2024/COMP-4114-A-F-2024.pdf, 2024. Accessed November 2025.

[134] Concordia University. COEN 691/COEN 498 - Quantum Information Theory and Coding. www.econcordia.com/home/coursedetails.aspx?id=5389, 2025. Accessed November 2025.

[135] Concordia University. Computer Science - General ProgramSeptember Entry. www.concordia.ca/content/dam/ginacody/csse/docs/CS-General-Program.pdf, 2025. Accessed November 2025.

[136] Dalhousie University. Mathematics and Statistics MATH 4680.03/5680.03 : Topics in Logic and Computation An Introduction to Quantum Computation. cdn.dal.ca/content/dam/dalhousie/pdf/faculty/science/math-stats/4680_syllabus_W2020.pdf, 2020. Accessed November 2025.

[137] McMaster University. MATH 3QC3* - Introduction to Quantum Computing. academiccalendars.romcmaster.ca/preview_course_nopop.php?catoid=53&coid=264994, 2025. Accessed November 2025.

[138] Mount Royal University. Bachelor of Science – Computer Science 2023-24. www.mtroyal.ca/ProgramsCourses/FacultiesSchoolsCentres/ScienceTechnology/Advising/_program-planning/2023-24-Computer-Science-Program-Planning-Guide.pdf, 2024. Accessed November 2025.

[139] Mount Royal University. PHYS 3602 - Elementary Quantum Mechanics. catalog.mtroyal.ca/preview_course_nopop.php?catoid=5&coid=6114, 2025. Accessed November 2025.

[140] Ontario Tech University. CSCI 4140U – Quantum Computing Software and Applications. calendar.ontariotechu.ca/preview_course_nopop.php?catoid=62&coid=113255. Accessed November 2025.

[141] Queen's University. Fundamental Computation Program Requirements. www.cs.queensu.ca/undergraduate/programs/options/fundamental-computation.php, 2025. Accessed November 2025.

[142] Simon Fraser University. INTRODUCTION TO QUANTUM ALGORITHMS. www.sfu.ca/outlines.html?2025/spring/macm/476/d100, 2025. Accessed November 2025.

[143] St. Francis Xavier University. Computer Science. Website. Accessed November 2025.

[144] Thompson Rivers University. GUIDELINES AND CHECKSHEET FOR BACHELOR OF COMPUTING SCIENCE. www.tru.ca/__shared/assets/BCS_First___Second_Year_Plan38325.pdf, 2025. Accessed November 2025.

[145] Toronto Metropolitan University. Computer Science Full-Time, Four Year Program. www.torontomu.ca/calendar/2024-2025/programs/science/computer_sci/#!accordion-1595938857886-full-time--four-year-program, 2025. Accessed November 2025.

[146] Toronto Metropolitan University. CPS 688 Advanced Algorithms. www.torontomu.ca/calendar/2024-2025/courses/computer-science/CPS/688/, 2025. Accessed November 2025.

[147] Trinity Western University. COMPUTING SCIENCE MAJOR CHECKLIST (122 s.h.) 2022-23 Academic Calendar. www.twu.ca/sites/default/files/2022-10/2022-23-computing-science-major.pdf, 2023. Accessed November 2025.

[148] Vancouver Island University. CSCI 429 (3) Advanced Topics in Algorithms and Complexity. www.viu.ca/programs/courses/computer-science, note = Accessed November 2025., 2025.

[149] Vancouver Island University. Requirements for a Major. Website, 2025. Accessed November 2025.

[150] Wilfrid Laurier University. CP351: Quantum Computing. bohr.wlu.ca/courses/course.php?courseKey=CP351&termKey=202009. Accessed November 2025.

[151] York University. EECS 4141 Quantum Computing. www.cse.yorku.ca/~roumani/4141.pdf. Accessed November 2025.

[152] York University. BACHELOR OF SCIENCE (BSc) COMPUTER SCIENCE. lassonde.yorku.ca/wp-content/uploads/2022-2023-Degree-Checklist-BSc-Hons-Computer-Science-1.pdf, 2023. Accessed November 2025.

[153] University of British Columbia. K-12 Education. quantumcomputing.ubc.ca/education/k-12-education, 2025. Accessed November 2025.

[154] University of Melbourne. Quantum User Interface. qui.research.unimelb.edu.au, 2025. Accessed November 2025.

[155] University of San Francisco. Breadth-First Search. www.cs.usfca.edu/~galles/visualization/BFS.html, 2025. Accessed November 2025.

[156] University of San Francisco. Data Structure Visualizations. cs.usfca.edu/~galles/visualization/Algorithms.html, 2025. Accessed November 2025.

[157] University of San Francisco. Red/Black Tree. www.cs.usfca.edu/~galles/visualization/RedBlack.html, 2025. Accessed November 2025.

[158] University of Waterloo. Quantum 101. uwaterloo.ca/institute-for-quantum-computing/resources/quantum-101/quantum-mechanics#quantum-superposition, 2025. Accessed November 2025.

[159] Salvadaor Elías Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum Information Processing, Springer*, 2012.

[160] Vercel. Next.js by Vercel - The React Framework. `nextjs.org`. Accessed November 2025.

[161] Wroblewski, Greg and Culp, Laura. Quantum Computing Playground. `experiments.withgoogle.com/quantum-computing-playground`, 2014. Accessed November 2025.

[162] Xanadu. Codebook. `pennylane.ai/codebook`, 2025. Accessed November 2025.

[163] Xanadu. qml.to_openqasm. `docs.pennylane.ai/en/stable/code/api/pennylane.to_openqasm.html`, 2025. Accessed November 2025.

[164] Xanadu. Strawberry Fields. `strawberryfields.ai/`, 2025. Accessed November 2025.

Appendices

Appendix A

Terms

Table A.1.: Quantum terms used in this thesis. We refer readers to the textbooks *Quantum Computation and Quantum Information* [78] and *Mathematical Foundations of Quantum Computing: A Scaffolding Approach* for further details [68].

Term	Description
Superposition	A fundamental quantum principle stating that a qubit can exist in a combination of the basis states $0\rangle$ and $1\rangle$ simultaneously until measured, allowing quantum computers to represent many possibilities at once.
Entanglement	A quantum phenomenon in which two or more qubits become correlated such that the state of one instantaneously affects the state of the other, regardless of distance.
Interference	The process by which quantum probability amplitudes combine constructively or destructively, enabling quantum algorithms to amplify correct outcomes and suppress incorrect ones.
Superdense Coding	A quantum communication protocol that allows two classical bits of information to be transmitted using only one qubit, by leveraging entanglement between sender and receiver.
BB84	The first quantum key distribution (QKD) protocol, developed in 1984, which enables two parties to establish a shared secret key securely by transmitting qubits in non-orthogonal bases.
Quantum teleportation	A protocol that transfers the quantum state of one qubit to another distant qubit using entanglement and classical communication, without physically moving the particle.
Grover's algorithm	A quantum search algorithm that provides a quadratic speedup for finding a marked item in an unsorted database, requiring $O(\sqrt{N})$ queries instead of $O(N)$
Shor's algorithm	A quantum algorithm for integer factorization that runs exponentially faster than the best known classical methods, posing a threat to factoring-based cryptographic systems.
Quantum Fourier Transform	The quantum analogue of the discrete Fourier transform, operating on amplitudes of quantum states and forming the basis for many quantum algorithms such as Shor's.
Quantum Phase Estimation	A core algorithm that estimates the eigenphase of a unitary operator, used in higher-level algorithms like Shor's algorithm and quantum chemistry simulations.
Quantum Approximate Optimization Algorithm	A hybrid quantum-classical algorithm that uses parameterized quantum circuits to find approximate solutions to combinatorial optimization problems.
Variational Quantum Eigensolver	A hybrid algorithm that minimizes a quantum system's energy expectation value using a parameterized circuit and a classical optimizer, commonly used in quantum chemistry.
Quantum annealing	A quantum optimization technique that uses adiabatic evolution to find low-energy states of a problem Hamiltonian, typically implemented in specialized hardware such as D-Wave systems.

Appendix B

Qiskit (2.2.3) gate coverage within QNotation

Table B.1.: Qiskit gate support details

Gate	Supported	Tested
C3SXGate*	–	–
C3XGate*	–	–
C4XGate*	–	–
CCXGate	✓	✓
CCZGate	✓	✓
CHGate	✓	✓
CPhaseGate	✓	✓§
CRXGate	✓	✓§
CRYGate	✓	✓§
CRZGate	✓	✓§
CSdgGate	✓	✓
CSGate	✓	✓
CSwapGate	✓	✓
CSXGate	✓	✓
CU1Gate‡	–	–
CU3Gate‡	–	–
CXGate	✓	✓
CYGate	✓	✓
CZGate	✓	✓
DCXGate	✓	✓
DiagonalGate†	–	–
ExactReciprocalGate†	–	–
FullAdderGate†	–	–
GlobalPhaseGate†	–	–
HalfAdderGate†	–	–

Table B.1 (continued)

Gate	Supported	Tested
HamiltonianGate†	–	–
HGate	✓	✓
IGate	✓	✓
InnerProductGate†	–	–
IntegerComparatorGate†	–	–
iSwapGate	✓	✓
LinearAmplitudeFunctionGate†	–	–
LinearPauliRotationsGate†	–	–
MCMTGate†	–	–
MCPhaseGate†	–	–
MCXGate†	–	–
ModularAdderGate†	–	–
MSGate†	–	–
MultiplierGate†	–	–
PauliEvolutionGate†	–	–
PauliGate†	–	–
PermutationGate†	–	–
PhaseGate	✓	✓§
PhaseOracleGate†	–	–
PiecewiseChebyshevGate†	–	–
PiecewiseLinearRotationsGate†	–	–
PiecewisePolynomialPauliRotationsGate†	–	–
PolynomialPauliRotationsGate†	–	–
QFTGate†	–	–
QuadraticFormGate†	–	–
RC3XGate	✓	✓
RCCXGate	✓	✓
RGate	✓	✓¶
RVGate	✓	✓‖
RXGate	✓	✓§
RXXGate	✓	✓§
RYGate	✓	✓§
RYYGate	✓	✓§
RZGate	✓	✓§
RZXGate	✓	✓§
RZZGate	✓	✓§
SdgGate	✓	✓

Table B.1 (continued)

Gate	Supported	Tested
SGate	✓	✓
SwapGate	✓	✓
SXdgGate	✓	✓
SXGate	✓	✓
TdgGate	✓	✓
TGate	✓	✓
U1Gate‡	–	–
U2Gate‡	–	–
U3Gate‡	–	–
UCGate†	–	–
UCPauliRotGate†	–	–
UCRXGate†	–	–
UCRYGate†	–	–
UCRZGate†	–	–
UGate	✓	✓‖
UnitaryGate†	–	–
WeightedSumGate†	–	–
XGate	✓	✓
XXMinusYYGate	✓	✓
XXPlusYYGate	✓	✓
YGate	✓	✓
ZGate	✓	✓

Appendix C

PennyLane (0.42.3) gate coverage within QNotation

Table C.1.: PennyLane gate support details

Gate	Supported	Tested
CH	✓	✓
CNOT	✓	✓
ControlledPhaseShift, CPhase	✓	✓§
CPhaseShift00	✓	✓§
CPhaseShift01	✓	✓§
CPhaseShift10	✓	✓§
CRot	✓	✓§
CRX	✓	✓§
CRY	✓	✓§
CRZ	✓	✓§
CSWAP	✓	✓
ctrl*	–	–
CY	✓	✓
CZ	✓	✓
ECR	✓	✓
GlobalPhase*	–	–
Hadamard	✓	✓
Identity	✓	✓
IsingXX	✓	✓§
IsingXY	✓	✓§
IsingYY	✓	✓§
IsingZZ	✓	✓§
ISwap	✓	✓
MultiControlledX*	–	–
MultiRZ*	–	–

Table C.1 (continued)

Gate	Supported	Tested
PauliRot*	–	–
PauliX	✓	✓
PauliY	✓	✓
PauliZ	✓	✓
PCPhase*	–	–
PhaseShift	✓	✓
PSWAP	✓	✓
Rot	✓	✓‖
RX	✓	✓§
RY	✓	✓§
RZ	✓	✓§
S	✓	✓
SISWAP, SQISW	✓	✓
SWAP	✓	✓
SX	✓	✓
T	✓	✓
Toffoli	✓	✓
U1	✓	✓§
U2	✓	✓¶
U3	✓	✓‖

Appendix D

Cirq (1.6.1) gate coverage within QNotation

Table D.1.: Cirq gate support details

Gate	Supported	Tested
CCNOT, CCX, TOFFOLI	✓	✓
CCXPowGate	✓	✓
CCZ	✓	✓
CCZPowGate	✓	✓
CNOT	✓	✓
CNotPowGate	✓	✓
CSWAP, CSwapGate, FREDKIN	✓	✓
CZ	✓	✓
CZPowGate	✓	✓
H	✓	✓
HPowGate	✓	✓
ISWAP	✓	✓
ISwapPowGate	✓	✓
MatrixGate	✓	✓
PhasedXPowGate	✓	✓
PhasedXZGate	–	–
QubitPermutationGate	–	–
rx	✓	✓
ry	✓	✓
rz	✓	✓
S	✓	✓
SWAP	✓	✓
SwapPowGate	✓	✓
T	✓	✓
ThreeQubitDiagonalGate	–	–
TwoQubitDiagonalGate	–	–
WaitGate	–	–
X	✓	✓
XPowGate	✓	✓
XX	✓	✓
XXPowGate	✓	✓
Y	✓	✓
YPowGate	✓	✓
YY	✓	✓
YYPowGate	✓	✓
Z	✓	✓
ZPowGate	✓	✓
ZZ	✓	✓
ZZPowGate	✓	✓

Appendix E

QNotation Tests

Table E.1.: Coverage of QNotation tests across libraries

Test	Qiskit	PennyLane	Cirq	Notes
TestEmpty	✓	✓	✓	Valid circuit with no gates
TestTypo	✓	✓	✓	Typo triggers InputError
TestHigherIndexedControlQubit	✓	✓	✓	–
TestNonNeighbouringQubits	✓	✓	✓	–
TestCCX	✓	✓	✓	–
TestCCZ	✓	–	✓	–
TestCH	✓	✓	–	–
TestCPhaseA	✓	✓	–	–
TestCPhaseB	✓	✓	–	–
TestCPhaseC	✓	✓	–	–
TestCPhaseShift00A	–	✓	–	–
TestCPhaseShift00B	–	✓	–	–
TestCPhaseShift00C	–	✓	–	–
TestCPhaseShift01A	–	✓	–	–
TestCPhaseShift01B	–	✓	–	–
TestCPhaseShift01C	–	✓	–	–
TestCPhaseShift10A	–	✓	–	–
TestCPhaseShift10B	–	✓	–	–
TestCPhaseShift10C	–	✓	–	–
TestCROTA	–	✓	–	–
TestCROTB	–	✓	–	–
TestCROTC	–	✓	–	–
TestCRXA	✓	✓	–	–
TestCRXB	✓	✓	–	–
TestCRXC	✓	✓	–	–
TestCRYA	✓	✓	–	–

Continued on next page

Table E.1 (continued)

Test	Qiskit	PennyLane	Cirq	Notes
TestCRYB	✓	✓	–	–
TestCRYC	✓	✓	–	–
TestCRZA	✓	✓	–	–
TestCRZB	✓	✓	–	–
TestCRZC	✓	✓	–	–
TestCS	✓	–	–	–
TestCSdg	✓	–	–	–
TestCSwap	✓	✓	✓	–
TestCSX	✓	–	–	–
TestCX	✓	✓	–	–
TestCY	✓	✓	–	–
TestCZ	✓	✓	–	–
TestDCX	✓	–	–	–
TestIsingXXA	–	✓	–	–
TestIsingXXB	–	✓	–	–
TestIsingXXC	–	✓	–	–
TestIsingXYA	–	✓	–	–
TestIsingXYB	-	✓	–	–
TestIsingXYC	–	✓	–	–
TestIsingYYA	–	✓	–	–
TestIsingYYB	–	✓	–	–
TestIsingYYC	–	✓	–	–
TestIsingZZA	–	✓	–	–
TestIsingZZB	–	✓	–	–
TestIsingZZC	–	✓	–	–
TestISwap	✓	–	✓	–
TestPhaseA	✓	–	–	–
TestPhaseB	✓	–	–	–
TestPhaseC	✓	–	–	–
TestR	✓	✓	–	–
TestRC3X	✓	–	–	–
TestRCCX	✓	–	–	–
TestRVA	✓	–	–	–
TestRVB	✓	–	–	–
TestRXA	✓	✓	✓	–
TestRXB	✓	✓	✓	–

Continued on next page

Table E.1 (continued)

Test	Qiskit	PennyLane	Cirq	Notes
TestRXC	✓	✓	✓	–
TestRXXA	✓	–	–	–
TestRXXB	✓	–	–	–
TestRXXC	✓	–	–	–
TestRYA	✓	✓	✓	–
TestRYB	✓	✓	✓	–
TestRYC	✓	✓	✓	–
TestRYYA	✓	–	–	–
TestRYYB	✓	–	–	–
TestRYYC	✓	–	–	–
TestRZA	✓	✓	–	–
TestRZB	✓	✓	–	–
TestRZC	✓	✓	–	–
TestRZXA	✓	–	–	–
TestRZXB	✓	–	–	–
TestRZXC	✓	–	–	–
TestRZZA	✓	–	–	–
TestRZZB	✓	–	–	–
TestRZZC	✓	–	–	–
TestS	✓	✓	–	–
TestSdg	✓	–	–	–
TestSwap	✓	✓	–	–
TestT	✓	✓	–	–
TestTdg	✓	–	–	–
TestUA	✓	✓	–	U3 is tested here for `PennyLane`
TestUB	✓	✓	–	U3 is tested here for `PennyLane`
TestU1A	–	✓	–	–
TestU1B	–	✓	–	–
TestU1C	–	✓	–	–
TestU2A	–	✓	–	–
TestU2B	–	✓	–	–
TestXXMinusYY	✓	–	–	–
TestXXPlusYY	✓	–	–	–
TestY	✓	✓	–	–

Continued on next page

Table E.1 (continued)

Test	Qiskit	PennyLane	Cirq	Notes
TestZ	✓	✓	–	–

Appendix F

QNotation Study: Start-of-Term Questionnaire

1. Have you taken a physics course before?
    - Yes
    - No
2. Have you taken a computer science course before?
    - Yes
    - No
3. Have you taken a linear algebra course before?
    - Yes
    - No
4. Please select the correct answer for the following question

$$\left(\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\right)\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}0\\ 0\\ 0\\ 1\end{pmatrix}$$

a)

$$\frac{1}{\sqrt{2}}\begin{pmatrix}0\\ 0\\ 1\\ -1\end{pmatrix}$$

b)

$$\frac{1}{\sqrt{2}}\begin{pmatrix}0\\ 0\\ -1\\ -1\end{pmatrix}$$

c)

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}$$

d) I don't know.

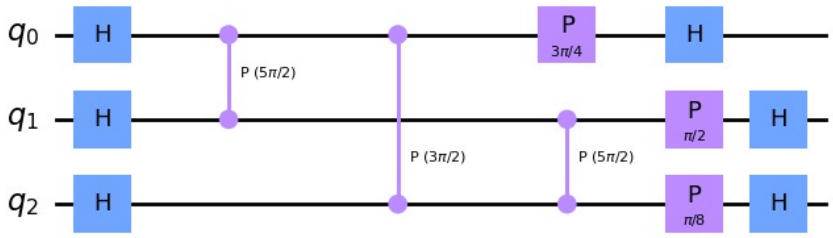


Figure F.1.|: The image for question 5

5. Please select all answers that correctly describe Figure F.1
   - Multiple H gates are applied concurrently
   - The image should be read from right to left
   - Some gates are applied to more than one wire
   - The image is invalid
6. Please select the correct answer to the following question
   $|\psi\rangle = (CZ \otimes I)(H \otimes H \otimes H)|000\rangle$
   a) $\frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)$
   b) $\frac{1}{\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)$
   c) $\frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$
   d) $\frac{1}{\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$
   e) I don't know.

Appendix G

QNotation Study: During Term Questionnaire

1. How many hours did you study for the course this week?
    - 0 hours
    - 1-3 hours
    - 4-6 hours
    - 7-9 hours
    - 10+ hours

2. Did you use QNotation while studying?
    - Yes
    - No
    - Not applicable

3. If you did use QNotation, please explain how you used it. (long answer)
4. QNotation can be used with what I learned this week in class
    - True
    - False

Appendix H

QNotation Study: End-of-Term Questionnaire

Please rate the following statements from 1-10, where 1 would be "Completely Disagree" and 10 would be "Completely Agree"

1. QNotation helped me in the course (1-10 rating)
2. QNotation was equally useful throughout the course (1-10 rating)
3. (if 10 is not given for the previous question) When was QNotation most useful for you? (long answer)
4. QNotation helped me learn about circuit, Dirac, and matrix notation (1-10 rating)
5. QNotation helped me translate between circuit, Dirac, and matrix notation (1-10 rating)
6. QNotation made my studying more efficient (1-10 rating)
7. QNotation allowed me to explore what I've learned in class (1-10 rating)
8. I enjoyed using QNotation (1-10 rating)
9. I felt immersed in my studying while using QNotation (1-10 rating)
10. The results I received from QNotation were worth the effort it took to receive them (1-10 ranking)
11. What do you like about QNotation? (long answer)
12. What do you not like about QNotation (long answer)