



Assignment of bachelor's thesis

Title:	Divide and conquer strategy for representation of classical data using quantum states
Student:	Pavel Slaninka
Supervisor:	Ing. Ivo Petr, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2025/2026

Instructions

Quantum computers might outperform classical computers in some tasks. However, we are faced with the problem of representing the task, i.e. classical data, in a form that a quantum computer can process. Different ways of data representation are suitable for different quantum algorithms, which will stand out, for example, when comparing quantum versions of algorithms known from the field of machine learning.

* The student will research known methods of preparing quantum states using quantum circuits and compare the complexity of the considered methods.

* In his work, the student will focus on the "amplitude encoding" method and its version based on the "divide and conquer" principle presented in [1]. He will also explore the possibilities of optimizing these algorithms in terms of the number of qubits and the depth of the quantum circuit.

* The student will implement the selected methods in the Qiskit package of Python programming language and test them on a quantum computer simulator.

* The student will research metrics used to evaluate the quality of state preparation on real quantum hardware. On appropriately selected instances, the student will perform the evaluation on real quantum hardware available online.

[1] Araujo, I.F., Park, D.K., Petruccione, F. et al. A divide-and-conquer algorithm for quantum state preparation. Sci Rep 11, 6329 (2021). <https://doi.org/10.1038/s41598-021-85474-1>



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Divide and conquer strategy for representation of classical data using quantum states

Pavel Slaninka

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Ivo Petr, Ph.D.

May 17, 2024

Czech Technical University in Prague
Faculty of Information Technology

© 2024 Pavel Slaninka. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Slaninka Pavel. *Divide and conquer strategy for representation of classical data using quantum states*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents



- Acknowledgments** **iv**
- Abstract** **vi**
- Declaration** **viii**
- 1 Introduction** **1**
 - 1.1 Quantum machine learning 2
 - 1.2 Thesis scope 3
 - 1.3 Inspiration 4
- 2 Quantum computing preliminaries** **5**
 - 2.1 Quantum states 5
 - 2.1.1 Bit and qubit 5
 - 2.1.2 Basic concepts 6
 - 2.1.3 General quantum state 9
 - 2.1.4 Measurements 10
 - 2.1.5 Qubit entanglement 13
 - 2.1.6 Global and relative phase 14
 - 2.2 Qubit visualization and Bloch sphere 14
 - 2.3 Quantum gate and quantum circuit 15
 - 2.4 One-qubit quantum gates 16
 - 2.4.1 Gate X 17
 - 2.4.2 Gate Y 18
 - 2.4.3 Gate Z 18
 - 2.4.4 Gate RX 19
 - 2.4.5 Gate RY 20
 - 2.4.6 Gate RZ 22
 - 2.5 Two-qubit quantum gates 23
 - 2.5.1 Gate CX 23
 - 2.5.2 Gate SWAP 24
 - 2.6 State preparation quality evaluation 25
 - 2.6.1 State fidelity 26
 - 2.6.2 Trace Distance 26

3	Data encoding methods	28
3.1	Basis encoding method	28
3.1.1	Definition	29
3.1.2	Properties	30
3.1.3	Usage example	33
3.2	Angle encoding method	34
3.2.1	Definition	37
3.2.2	Properties	38
3.2.3	Usage example	42
3.3	Amplitude encoding method	44
3.3.1	Introduction	44
3.3.2	Detailed example	45
3.3.3	Generalization	53
3.3.4	Properties	57
3.3.5	Divide-and-conquer encoding method	60
3.3.5.1	Introduction and core ideas	60
3.3.5.2	Divide-and-conquer strategy and example	62
3.3.5.3	Algorithm	66
3.3.6	Implications	69
4	Implementation	77
4.1	Repository overview	78
4.1.1	Implementation details	79
4.1.2	Jupyter Notebooks	80
4.2	Pseudocode	82
4.2.1	Basis encoding method	83
4.2.2	Angle encoding method	83
4.2.3	Amplitude encoding method	84
4.2.4	Divide-and-conquer encoding method	86
5	Testing	88
5.1	Empirical complexity comparison	88
5.1.1	No transpilation	94
5.1.2	Transpilation with no optimization	97
5.1.3	Transpilation with best optimization	99
5.2	Basis encoding method test	102
5.3	Angle encoding method test	108
5.4	Amplitude encoding method test	112
5.5	Divide-and-conquer encoding method test	116
6	Conclusion	120
	Bibliography	122
	Repository	128

I would like to express my deepest gratitude to my supervisor, Ivo Petr, for his invaluable guidance and dedicated mentorship during the development of this thesis. His expertise, enthusiasm, and patience have been pivotal in shaping both the direction and execution of my thesis work.

I am also immensely thankful to my family, whose unwavering support and encouragement have sustained me throughout my academic endeavors, even as they have navigated through periods of profound adversity, facing immense challenges with remarkable resilience and strength.

Additionally, I would like to thank my peers for their companionship and assistance during my studies. Their understanding and camaraderie have enriched this challenging journey, thus making it more enjoyable and smoother.

Lastly, I extend my appreciation to all my lecturers and tutors, who have contributed significantly to my academic experience, thereby enabling the realization of this thesis.

Abstract

This thesis focuses on exploring several known quantum data encoding methods. Specifically, four data encoding methods are presented, each utilizing a different approach to the representation of classical data using quantum states. Namely, the basis, angle, amplitude, and divide-and-conquer data encoding methods are discussed, with the divide-and-conquer encoding approach being based on the amplitude encoding method. Each encoding technique is first introduced and explained from a theoretical standpoint, with implications and characteristics discussed in detail. The encoding approaches are compared in terms of their qubit and gate complexities. The thesis comes with a repository containing the Python implementation of each introduced encoding method, with the implementation using the Qiskit library. Each encoding method is tested using a small input on IBM Quantum hardware, and the outcomes are compared with the expected results.

Keywords quantum state preparation, classical data embedding, basis encoding, angle encoding, amplitude encoding, divide-and-conquer algorithm, quantum circuit, Qiskit

Abstrakt

Tato práce se zaměřuje na zkoumání několika známých způsobů kvantového kódování dat. Konkrétně jsou představeny čtyři postupy kódování dat, z nichž každý využívá jiný přístup k reprezentaci klasických dat pomocí kvantových stavů. Konkrétně jsou probírány metody bazického, úhlového a amplitudového kódování dat. Také je prezentována metoda kódování dat na základě principu „rozděl a panuj“, přičemž tato metoda vychází z metody amplitudového kódování. Každý způsob kódování je nejprve teoreticky představen a vysvětlen, přičemž jsou podrobně rozebrány důsledky a vlastnosti. Jednotlivé kódovací strategie jsou porovnány z hlediska počtu potřebných qubitů a hradel. Součástí práce je repozitář obsahující implementaci každé představené strategie kódování. Implementace využívá jazyk Python a knihovnu Qiskit. Všechny kódovací metody jsou testovány prostřednictvím malého vstupu na kvantových počítačích od IBM a výsledky jsou porovnány s očekávanými výsledky.

Klíčová slova příprava kvantového stavu, vkládání klasických dat, bazické kódování, úhlové kódování, amplitudové kódování, algoritmus rozděl a panuj, kvantový obvod, Qiskit

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on May 16, 2024

Chapter 1

Introduction

“Embracing the ethos of effective accelerationism, we must propel humanity forward with a zeal that challenges the very constraints of entropy. As we edge closer to the epoch of singularity, it is not merely about the creation of AGI with an inconceivable level of intelligence. The entire essence of humanity will transcend, not just thwarting the decay and entropy of aging in the process but leading to a state of eternal and infinite bliss. This pivotal juncture in our evolution, driven by the principles of transhumanism, represents a transformative leap towards an existence where limitations are but echoes of our primitive past. Our journey is one of exponential growth, where every step taken is a stride towards a future redefined by our boundless potential.”

– GPT-4o

To continue accelerating the recent expansion of large multimodal generative AI models [1, 2], more advanced computational hardware is required as current transistor technologies approach their physical limits [3]. Although innovations still continue [4, 5], as proven by frequent releases of ever more capable AI models [6], the insufficient capabilities of the currently available computational resources are becoming more of a bottleneck with each passing model release [7, 8]. This is further exacerbated by the fact that the global demand for state-of-the-art AI hardware significantly outpaces the total production capacity [7], thereby hindering the adoption and deployment of these technologies.

Furthermore, it is becoming increasingly challenging to shrink transistors—a 20th-century invention [8]—further down in size while at the same time boosting their energy efficiency [3]. And focusing on the pressing mission of continually improving their energy efficiency is paramount, in part due to the rapidly raising number of data centers being erected across the globe [9, 10, 11], whose total electricity consumption is beginning to spiral out of control [12, 13].

However, along with substantial innovations in current hardware, promising novel computational substrates stemming from technologies such as quantum computing [14, 15] or thermodynamic computing [16] could help the in-

dustry leapfrog forward by overcoming these challenges, potentially ushering in an entirely new era of computation and ubiquitous ASI [17].

1.1 Quantum machine learning

As a leading alternative to classical computing, the field of quantum computing has made profound advances in recent years [18, 19]. Quantum computers allow for leveraging quantum mechanics to surpass some fundamental limitations that classical computers’ transistor-based nature poses. In contrast to algorithms in classical computing, quantum algorithms may utilize quantum phenomena, such as quantum superposition and entanglement, to potentially deliver significant speedups for certain types of calculations [14, 15, 20, 21].

Especially algorithms in the field of quantum ML typically use some information as input. That information, most often in the form of a dataset, must somehow become represented in the state of a quantum system for any quantum algorithm to be able to use it. This process of classical data representation using quantum states, often called “the input problem”, is integral and precedes any other computations [21], since some data must first be encoded into a quantum system for a quantum algorithm to process them. This encoding process can be performed in a multitude of possible ways, possibly significantly affecting subsequent algorithms that use the encoded data [22, 23]. On the other hand, “the output problem” arises when faced with the necessity to extract processed data from a quantum state [21]. In summary, these three actions are commonly performed when conducting ML computations on quantum computers:

1. Encoding classical data into a quantum state in a way that the state fully represents the data.
2. Utilizing the encoded data in a quantum algorithm that performs some quantum operations and processes the encoded data by manipulating the quantum state.
3. Measuring qubits either intermediately as the quantum state evolves or after all computations are finished [24], and interpreting the measurement results.

Each of these three actions holds importance, but this thesis aims to concentrate solely on the first one¹, which is the concept of representing classical data using quantum states. This concept is introduced in greater detail in the following section, where the scope of the thesis is delineated by outlining the main thesis objectives.

¹The third action involves performing qubit measurements. In this thesis, qubit measurements are also performed, but only to evaluate the performance of data encoding methods (that are introduced in this thesis).

1.2 Thesis scope

When given a dataset containing classical data and a quantum algorithm that uses encoded data as input, identifying the optimal data encoding technique to maximize the algorithm’s performance is a nontrivial task, and it is tempting to neglect this process in favor of the “fun part”, which is designing and executing algorithms that use encoded data². This is because, as previously mentioned, the performance of such algorithms can potentially be severely affected by the choice of an encoding method. And encoding methods themselves may vary drastically in the overhead they bring to a quantum circuit [25, 26]. This thesis explores these concepts, with the principal focus of the thesis being to present certain solutions to “the input problem”. One of the solutions introduced is the divide-and-conquer state preparation algorithm [27], which is the main inspiration for this thesis. Specifically, the main goals of the thesis are outlined as follows:

- Covering certain aspects of quantum computing needed in the later parts of the thesis.
- Presenting three common state preparation data encoding methods [25], more precisely these three encoding methods:
 - the basis encoding method
 - the angle encoding method
 - the amplitude encoding method
- Introducing the divide-and-conquer data encoding method [27] that is derived from the amplitude encoding method.
- Discussing the implementation of these four encoding methods.
- Testing the encoding methods on real quantum hardware and evaluating their performance.
- Comparing the encoding methods in terms of their qubit and gate complexities.
- Discussing possible enhancements of the divide-and-conquer state preparation algorithm.

The thesis does not delve into anything that happens after data encoding. It only centers on the representation of classical data using quantum states that are created by using the four encoding methods mentioned in the list above. Ordinarily, as mentioned before, the representation of classical data in a quantum state is just the first step, after which a model uses the encoded data, and then the measurement results are taken and interpreted.

²However, depending on the situation, it might be sufficient to use encoding methods provided by quantum computing software development kits.

1.3 Inspiration

This thesis is heavily inspired by the article named "A divide-and-conquer algorithm for quantum state preparation", written by authors Israel F. Araujo, Daniel K. Park, Francesco Petruccione, and Adenilton J. da Silva [27]. In this article, the authors present a data encoding algorithm based on a divide-and-conquer strategy. The main goal of this thesis, among other objectives, is to explain this algorithm, implement it, compare its computational complexity with other introduced encoding algorithms, and discuss possible improvements to this algorithm.

Quantum computing preliminaries

The second chapter of this thesis provides some of the key theoretical concepts in quantum computing that are fundamental to understanding the subsequent parts of the thesis. Rather than serving as an introduction to quantum mechanics as a whole or as an exhaustive textbook-like introduction to quantum computing, this chapter only introduces certain essential concepts in quantum programming. A thorough introduction to quantum computing is beyond the scope of this thesis; therefore, this chapter may not cover all of the theoretical background needed for the subsequent thesis chapters. The concepts discussed in this chapter are defined and concisely explained using primarily linear algebra. Several sources were used when writing this chapter [25, 28]¹.

2.1 Quantum states

This chapter starts by presenting the fundamental element of information in quantum computing—a quantum bit, also known as **qubit**—and comparing it to the classical bit.

2.1.1 Bit and qubit

In classical computing, the smallest unit of information is called a bit—a binary digit. The capacity of a bit is limited, as its name implies, as it can hold only a limited amount of information. It has the ability to exist in only one of two states at any given time. This behavior becomes apparent in the following note:

¹For practical reasons, the sources are only mentioned here, at the beginning of this chapter.

► Note 2.1 (Bit). A bit can be characterized as a vector of the vector space over the finite field \mathbb{Z}_2 . As a one-dimensional vector over \mathbb{Z}_2 , a bit can be represented simply as a scalar 0 or 1. ◀

In contrast to a bit, which can store one of two possible states, a qubit can store one of infinite possible states (within certain constraints discussed below):

► **Definiton 2.2** (Qubit). *A qubit is a vector of the two-dimensional complex inner product space \mathbb{C}^2 , where \mathbb{C}^2 is also a two-dimensional Hilbert space.* ◀

There is one significant distinction between bits and qubits that is apparent—the ability to store one of the infinite possible states makes the qubit a much more powerful unit of information than the bit. This is one of the biggest advantages quantum computing brings compared to classical computing.

2.1.2 Basic concepts

In order to continue in this introductory chapter, several key quantum computing concepts need to be explained first.

► Note 2.3 (Bra-ket notation and Hilbert space operations). Throughout this entire thesis, elements of Hilbert spaces (denoted as \mathcal{H}) are represented using the notation explained in this note, also known as the **bra-ket** notation.

- $\mathbb{C}^{n,1}$ for column vectors:

This space consists of column vectors with $n \in \mathbb{N}$ rows and 1 column. It is the standard representation of vectors in the Hilbert space \mathcal{H} :

$$\mathbb{C}^{n,1} = \mathbb{C}^n = \left\{ |\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} \mid \psi_i \in \mathbb{C} \right\}.$$

The vectors $|\psi\rangle$ are called **kets** (one such vector is called **ket**).

- $\mathbb{C}^{1,m}$ for row vectors:

This space consists of row vectors with 1 row and $m \in \mathbb{N}$ columns:

$$\mathbb{C}^{1,n} = \{ \langle\phi| = (\phi_1 \ \phi_2 \ \dots \ \phi_m) \mid \phi_i \in \mathbb{C} \}.$$

The vectors $\langle\phi|$ are called **bras** (one such vector is called **bra**).

- Then, when having two kets $|\psi\rangle, |\phi\rangle \in \mathcal{H}^n$, $n \in \mathbb{N}$, the ordinary vector operations can be denoted as:

$$|\psi\rangle = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_n \end{pmatrix}, \quad |\phi\rangle = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_n \end{pmatrix}, \quad |\psi\rangle + |\phi\rangle = \begin{pmatrix} \psi_1 + \phi_1 \\ \vdots \\ \psi_n + \phi_n \end{pmatrix}, \quad \alpha|\psi\rangle = \begin{pmatrix} \alpha\psi_1 \\ \vdots \\ \alpha\psi_n \end{pmatrix}.$$

- And their **inner product** is characterized as:

$$(|\psi\rangle, |\phi\rangle) = \langle\psi| \cdot |\phi\rangle = \langle\psi|\phi\rangle = (\bar{\psi}_1 \ \bar{\psi}_2 \ \cdots \ \bar{\psi}_n) \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \sum_{i=1}^n \bar{\psi}_i \phi_i,$$

where $(|\psi\rangle, |\phi\rangle)$ denotes the inner product and $\bar{\psi}_i$ denotes complex conjugate.

- The **outer product** of $|\psi\rangle$ and $\langle\phi|$, forming an $n \times n$ operator $|\psi\rangle\langle\phi|$ that acts on the Hilbert space $\mathcal{H}^{n,n}$, is characterized as:

$$|\psi\rangle\langle\phi| = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} (\bar{\phi}_1, \bar{\phi}_2, \dots, \bar{\phi}_n) = \begin{pmatrix} \psi_1\bar{\phi}_1 & \psi_1\bar{\phi}_2 & \cdots & \psi_1\bar{\phi}_n \\ \psi_2\bar{\phi}_1 & \psi_2\bar{\phi}_2 & \cdots & \psi_2\bar{\phi}_n \\ \vdots & \vdots & \ddots & \vdots \\ \psi_n\bar{\phi}_1 & \psi_n\bar{\phi}_2 & \cdots & \psi_n\bar{\phi}_n \end{pmatrix}.$$

- Given two vectors $|\psi\rangle \in V$ and $|\phi\rangle \in W$, their **tensor product** is denoted by $|\psi\rangle \otimes |\phi\rangle$ and forms an element of $V \otimes W$.
 - The tensor product of vector spaces V and W , denoted as $V \otimes W$, is a new vector space of the dimensions of V and W .
 - If $\{|v_i\rangle\}$ is a basis for V and $\{|w_j\rangle\}$ is a basis for W , then $\{|v_i\rangle \otimes |w_j\rangle\}$ is a basis for $V \otimes W$.

For example, if $|\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix}$ and $|\phi\rangle = \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}$, their tensor product is:

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} \otimes \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = \begin{pmatrix} \psi_1 \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \\ \psi_2 \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \psi_1\phi_1 \\ \psi_1\phi_2 \\ \psi_2\phi_1 \\ \psi_2\phi_2 \end{pmatrix}.$$

The tensor product is a fundamental operation in quantum computing, allowing the combination of vectors to produce larger vectors. This way, qubits can be combined together, where if k qubits are combined, the resulting vector's length would equal 2^k .

- Given two matrices A and B , where A is an $m \times n$ matrix and B is a $p \times q$ matrix, then their tensor product is called the **Kronecker product**, denoted as $A \otimes B$, producing an $mp \times nq$ matrix given by:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

If $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$, their Kronecker product is:

$$\begin{aligned} A \otimes B &= \begin{pmatrix} a_{11} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{12} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \\ a_{21} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{22} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{pmatrix} = \\ &= \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}. \end{aligned}$$

- Note 2.4 (Computational basis). In a Hilbert space $\mathcal{H} = \mathbb{C}^n$, $n \in \mathbb{N}$ infinitely many possible orthonormal bases exist (bases whose vectors are perpendicular to each other and have a norm equal to one). In this thesis, only the standard basis is used. That means that if, for example, $n = 2$, the standard basis ($|0\rangle, |1\rangle$) is used, where

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

If $n = 4$, the standard basis ($|00\rangle, |01\rangle, |10\rangle, |11\rangle$) is used, where

$$\begin{aligned} |00\rangle &= |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, & |01\rangle &= |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \\ |10\rangle &= |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, & |11\rangle &= |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \end{aligned}$$

In quantum computing, the standard basis vectors are often referred to as **computational basis states**. These states are represented by binary strings where each digit corresponds to the state of one qubit. For an n -qubit system, each qubit can be in either the $|0\rangle$ or $|1\rangle$ state, leading to 2^n possible computational basis states.

Each binary digit in the string represents the state of an individual qubit. The rightmost digit represents the state of the first qubit, the next digit to the left represents the state of the second qubit, and so on. This is known as the **little-endian** convention.

► Note 2.5. This thesis aims to use the little-endian convention in quantum computations most of the time. However, the concept of endianness may be confusing in some parts of the thesis, as many sources other than this thesis explain quantum computing concepts using a different endianness convention, namely the **big-endian** convention, where the rightmost digit in kets represents the first qubit. ◀

For instance, in a 3-qubit system, the state $|100\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle$ means that the first qubit is in state $|1\rangle$, the second qubit is in state $|0\rangle$, and the third qubit is in state $|1\rangle$, and their tensor product is computed.

This binary representation extends naturally to any number of qubits. For an n -qubit system, each computational basis state is a tensor product of n individual qubit states. The general form of an n -qubit computational basis state is:

$$|b_n b_{n-1} \cdots b_2 b_1\rangle = |b_n\rangle \otimes |b_{n-1}\rangle \otimes \cdots \otimes |b_2\rangle \otimes |b_1\rangle,$$

where each $b_i \in \{0, 1\}$ denotes the state of the i -th qubit.

This completes the explanation of computational basis states and their notation. ◀

Getting back to the qubit, each qubit $|\psi\rangle \in \mathcal{C}^2$ can be represented using the above notation in the following way:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \in \mathcal{C}^2, \quad \alpha, \beta \in \mathbb{C}.$$

Thus, the qubit $|\psi\rangle$ can be characterized as being a linear combination of the computational basis states $|0\rangle$ and $|1\rangle$. If $\alpha, \beta \neq 0$, it is said that the qubit $|\psi\rangle$ is in a **superposition** of those basis states. The numbers α and β are called **amplitudes** of the basis states. This characterization of a single qubit can be extended to multiple qubits to create a definition of an arbitrary quantum state.

2.1.3 General quantum state

The following definition describes a quantum state consisting of an arbitrary number of qubits:

► **Definiton 2.6** (Pure quantum state). Let \mathcal{H} be a Hilbert space such that $\mathcal{H} = \mathbb{C}^{2^m} = (\mathbb{C}^2)^{\otimes m}$ and a vector $|\psi\rangle \in \mathcal{H}$ such that $\|\psi\rangle\|^2 = 1$. Then, $|\psi\rangle$ is called a **pure quantum state** (commonly referred to as simply a **state**) on the space \mathcal{H} , where $|\psi\rangle$ consists of m qubits. ◀

► Remark 2.7. Let \mathcal{H} be a Hilbert space such that $\mathcal{H} = \mathbb{C}^{2^m}$, where the standard computational basis is used. Based on the definition above, a state $|\psi\rangle \in \mathcal{H}$ can be characterized as the following linear combination of the standard basis states:

$$|\psi\rangle = \sum_{i=1}^{2^m} \alpha_i |e_i\rangle,$$

where $\alpha_i \in \mathbb{C}$ are complex numbers also known as **amplitudes** and $|e_i\rangle \in \mathcal{H}$ is the i -th computational basis state of the standard basis used by \mathcal{H} . The normalization condition from the definition must be satisfied, meaning that:

$$\langle\psi|\psi\rangle = \sum_{i=1}^{2^n} |\alpha_i|^2 = 1.$$

► **Example 2.8.** For a two-qubit pure quantum state, a general formula can be written as:

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle,$$

where $\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11} \in \mathbb{C}$. ◀

► Note 2.9. The definition defines a *pure* quantum state. However, not every quantum state can be characterized by a vector, as not every quantum state is *pure*. Some quantum states that cannot be characterized by the definition above are referred to as **mixed quantum states**. The concept of *mixed* quantum states goes beyond the scope of the thesis aims. However, it is briefly touched upon in the subsection 3.30. ◀

It is essential to clarify why pure quantum states must be normalized to be valid quantum states. In order to do that, the concept of measurements needs to be explained first.

2.1.4 Measurements

In quantum computing, measurements are a fundamental aspect of extracting information from qubits. A quantum computer performs operations on qubits, evolving the state of the qubits through a series of transformations. However, to retrieve information from these transformed qubits, they must be measured. Measurement is the process by which each measured qubit collapses from a

superposition of its basis states $|0\rangle$ or $|1\rangle$ to one of those states. The probabilities of these outcomes depend on the amplitudes of the quantum state's superposition.

The **Born's rule** provides a way to calculate the probabilities of obtaining each possible outcome from a quantum measurement. For a pure quantum state $|\psi\rangle$ expressed in the computational basis, the probability of measuring a particular basis state $|e_i\rangle$ is given by the square of the magnitude of the corresponding amplitude α_i . This is clearly depicted in the following definition:

► **Definiton 2.10** (Measurement probability). *Let $|\psi\rangle \in \mathbb{C}^{2^m}$, $m \in \mathbb{N}$ be a pure quantum state characterized as:*

$$|\psi\rangle = \sum_{i=1}^{2^m} \alpha_i |e_i\rangle,$$

where $\alpha_i \in \mathbb{C}$ is the amplitude of the i -th computational basis state $|e_i\rangle \in \mathcal{H}$ (it is assumed that the standard basis is in use).

Then, $\forall i \in [1, 2^n] \subset \mathbb{N}$ a number $P_{|e_i\rangle}(|\psi\rangle)$ is defined such that

$$P_{|e_i\rangle}(|\psi\rangle) = |\alpha_i|^2,$$

where $P_{|e_i\rangle}(|\psi\rangle)$ denotes the probability of the state $|\psi\rangle$ collapsing into the i -th basis state $|e_i\rangle$ when all qubits of $|\psi\rangle$ are measured.

Moreover, let $|\phi\rangle \in \mathbb{C}^{2^m}$ be another pure quantum state. Then, a number $P_{|\phi\rangle}(|\psi\rangle)$ is defined such that

$$P_{|\phi\rangle}(|\psi\rangle) = |(\langle\phi|, |\psi\rangle)|^2.$$

The value $P_{|\phi\rangle}(|\psi\rangle)$ represents the probability of obtaining the state $|\phi\rangle$ after all qubits of $|\psi\rangle$ are measured. This equation is called the **Born rule**. ◀

► **Remark 2.11.** The *Born rule* calculates probabilities, and all probabilities are real numbers ranging from 0 to 1. That means that The *Born rule* ensures that the probabilities of all possible outcomes sum to 1, as required by the normalization condition of the quantum state:

$$\sum_{i=1}^{2^n} |\alpha_i|^2 = 1.$$

Below is an example of this characteristic on a single-qubit quantum state $|\psi\rangle \in \mathbb{C}^2$, represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, $\alpha, \beta \in \mathbb{C}$. If this qubit is measured, the probability of measuring the state $|0\rangle$ is equal to $|\alpha|^2$:

$$\begin{aligned} P_{|0\rangle}(|\psi\rangle) &= |(\langle 0|, |\psi\rangle)|^2 = |(\langle 0|, \alpha|0\rangle + \beta|1\rangle)|^2 = \\ &= |\alpha\langle 0|, |0\rangle + \beta\langle 0|, |1\rangle|^2 = |\alpha|^2. \end{aligned}$$

Analogically, the probability of measuring the state $|1\rangle$ is equal to $|\beta|^2$. No other measurement outcomes are possible, and the sum of the probabilities of all possible events must add up to 1. Therefore:

$$P_{|\psi\rangle}(|\psi\rangle) = |(|\psi\rangle, |\psi\rangle)|^2 = |\alpha|^2 + |\beta|^2 = 1.$$

Thus, the state vector representing the qubit $|\psi\rangle$ must be normalized so that its magnitude is 1.

It should also be observed that post-measurement, all amplitudes become zero except for one. Consequently, the qubit ceases to be in a superposition, independent of its initial state. A **collapse** into the state corresponding to the measured value has occurred, and the original state can no longer be reconstructed. The same is true for multi-qubit states. ◀

▶ Note 2.12. In the context of quantum measurements, it should be noted that not all qubits of a quantum state are required to be measured every time. Information can be extracted from only a subset of the qubits. When a subset of qubits is measured, the state of the remaining qubits collapses accordingly, based on the outcome of the measured subset.

If a quantum state $|\psi\rangle \in \mathbb{C}^{2^n}$ is considered, it is a superposition of n qubits. By measuring only a subset of these qubits, the entire state collapses into a state consistent with the measurement outcomes, but only the measured qubits provide direct information. The state of the unmeasured qubits becomes a conditional state, determined by the measurement results.

For example, if $|\psi\rangle$ is a two-qubit quantum state such as:

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle,$$

and only the first qubit is measured, the state collapses to either $|0\rangle$ or $|1\rangle$ for the first qubit. The probabilities of these outcomes are $|\alpha|^2 + |\gamma|^2$ and $|\beta|^2 + |\delta|^2$, respectively. If the first qubit is measured and found to be $|0\rangle$, the state of the system collapses to:

$$|\psi_{*0}\rangle = \frac{\alpha|00\rangle + \gamma|01\rangle}{\sqrt{|\alpha|^2 + |\gamma|^2}}.$$

Here, $|\psi_{*0}\rangle$ represents the conditional state of the second qubit, given that the first qubit was measured as $|0\rangle$.

This process illustrates that measurements can be partial, and the state of the unmeasured qubits remains in a conditional superposition based on the measurement outcomes of the measured qubits. Such selective measurement is crucial in some quantum algorithms (such as the final encoding method presented in this thesis), where only specific information must be extracted. ◀

▶ Note 2.13 (Visualization of probabilities). The probabilities calculated using Born's rule can be effectively visualized using histograms. A histogram represents the distribution of probabilities for different measurement outcomes.

Each bar in the histogram corresponds to a basis state, and the height of the bar represents the probability of the quantum state collapsing to that basis state upon measurement.

For example, if the following two-qubit quantum state is considered:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

the histogram depicting all possible measurement outcomes would have four bars, one for each of the states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. The height of each bar would be given by $|\alpha_{00}|^2$, $|\alpha_{01}|^2$, $|\alpha_{10}|^2$, and $|\alpha_{11}|^2$, respectively.

In one of the last chapters of this thesis (5), histograms are used to present and analyze the results of the measurements for various encoding methods. By visualizing the probabilities, insight can be gained into the behavior of the data encoding methods. Furthermore, the histograms presented in that chapter aid in comparing theoretical predictions with actual experimental results. Any discrepancies between expected and observed outcomes can indicate issues in the state preparation process. ◀

2.1.5 Qubit entanglement

Qubit entanglement is one of the phenomena present in quantum computing. It describes a situation where two or more qubits become correlated in such a way that the state of each qubit cannot be described independently of the state of the others.

▶ **Definiton 2.14** (Entangled quantum state). *A quantum state consisting of multiple qubits is considered **entangled** if it cannot be described as a tensor product of the states of its individual qubits.* ◀

In contrast, a quantum state is **non-entangled**, or **separable** if it can be written as a tensor product of the states of its individual qubits. For example, the state:

$$|010\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle$$

is not entangled, as it is simply the tensor product of individual states.

An example of entangled states would be so-called **Bell states**. There are four *Bell states*, but the most commonly referenced one is the following:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

This state cannot be decomposed into the product of individual qubit states, demonstrating its *entanglement*. In this state, if the first qubit is measured and found to be in state $|0\rangle$, the second qubit will also be in state $|0\rangle$. Similarly, if one of the qubits is measured in state $|1\rangle$, the other qubit would also be in state $|1\rangle$. Some of the encoding methods this thesis introduces prepare states where the qubits are entangled.

2.1.6 Global and relative phase

Valid quantum states are normalized so that they have a magnitude of 1. Let $|\psi\rangle$ be a quantum state. Then, $\forall \alpha \in \mathbb{C}$, the following is true:

$$\|\alpha |\psi\rangle\| = \sqrt{\langle \alpha |\psi\rangle, \alpha |\psi\rangle} = \sqrt{\bar{\alpha}\alpha \langle |\psi\rangle, |\psi\rangle} = |\alpha| \cdot \|\psi\rangle\| = |\alpha|.$$

To ensure that $\alpha |\psi\rangle$ is a valid quantum state, it must be true that $|\alpha| = 1$.

Let there be $|\psi\rangle, |\phi\rangle \in \mathbb{C}^n$ and $\alpha \in \mathbb{C}$ such that $|\alpha| = 1$. Then, the *Born rule* is used on $\alpha |\psi\rangle$ and $|\phi\rangle$, resulting in the following expressions being true:

$$P_{|\phi\rangle}(\alpha |\psi\rangle) = |\langle \phi | (\alpha |\psi\rangle)|^2 = |\alpha \langle \phi | \psi\rangle|^2 = |\alpha|^2 |\langle \phi | \psi\rangle|^2 = |\langle \phi | \psi\rangle|^2 = P_{|\phi\rangle}(|\psi\rangle).$$

The previous equality holds for all possible states $|\phi\rangle$. Thus, the vectors $|\psi\rangle$ and $\alpha |\psi\rangle$ cannot be distinguished by a measurement and therefore are considered identical. The multiplier α where $|\alpha| = 1$ is called the **global phase**.

Therefore, multiplying states by a global phase does not change the measurement outcomes. However, when only some state amplitudes are multiplied (by some number), that may often introduce changes that result in different measurement outcomes. When an amplitude is modified (assuming that the resulting state is still normalized), such a modification is called a change in the **relative phase** because a certain amplitude is being modified *relative to other amplitudes*.

2.2 Qubit visualization and Bloch sphere

As explained in the first section (2.1.1) of this chapter, in classical computing, the state of a bit is straightforward. It does not require special visualization to understand its state, as it can only be in one of two states. However, in quantum computing, qubits represent complex quantum states that can exist in superpositions of 0 and 1. Visualizing these states could significantly aid in understanding the behavior and properties of qubits.

A useful and commonly employed method for visualizing qubit states is called the **Bloch sphere** representation. The Bloch sphere provides a geometrical representation of the pure state of a single qubit. In this representation, any such state can be depicted as a point on the surface of the unit sphere. In such a representation, a single qubit $|\psi\rangle$ can be expressed using two parameters, θ and ϕ , which correspond to the spherical coordinates on the Bloch sphere. This parametrization is given by:

► **Definiton 2.15** (Bloch sphere parametrization). *Let \mathcal{H} represent a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$ and a state $|\psi\rangle \in \mathcal{H}$. The state $|\psi\rangle$ can be represented using the following parametrization:*

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle, \quad \theta \in [0, \pi), \phi \in [0, 2\pi).$$



This parametrization can then be used to visualize the state of a qubit on the unit sphere. In this representation:

- θ is the polar angle, which ranges from 0 to π .
- ϕ is the azimuthal angle, which ranges from 0 to 2π .

The Bloch sphere allows for a clear and intuitive visualization of a qubit's state:

- The north pole of the Bloch sphere represents the state $|0\rangle$.
- The south pole represents the state $|1\rangle$.
- Any point on the surface of the sphere represents a pure qubit state, which is a superposition of the states $|0\rangle$ and $|1\rangle$.

For example:

- When $\theta = 0$ and $\phi = 0$, the qubit is in the state $|0\rangle$.
- When $\theta = \pi$ and $\phi = 0$, the qubit is in the state $|1\rangle$.
- When $\theta = \frac{\pi}{2}$ and $\phi = 0$, the qubit is in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- When $\theta = \frac{\pi}{2}$ and $\phi = \pi$, the qubit is in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Understanding and utilizing the Bloch sphere visualization may help in gaining deeper insights into the nature of qubit states, superposition, and the impact of various quantum operations.

2.3 Quantum gate and quantum circuit

In classical computing, logical gates are utilized when performing bits-based operations. In quantum computing, quantum states and qubits can be manipulated using **unitary operators**:

► **Definiton 2.16** (Unitary operator). *Let \mathcal{H} be a Hilbert space. An operator $U : \mathcal{H} \rightarrow \mathcal{H}$ is called **unitary** if it satisfies the following conditions:*

1. *Preservation of the inner product $\forall x, y \in \mathcal{H}$,*

$$(Ux, Uy) = (x, y)$$

2. *Surjectivity of U , which implies that $\forall y \in \mathcal{H}, \exists x \in \mathcal{H} : Ux = y$.*

These properties ensure that U is also invertible, with the inverse U^{-1} being equal to the Hermitian adjoint U^\dagger (conjugate transpose) of U , hence:

$$UU^\dagger = U^\dagger U = I,$$

where I is the identity operator on \mathcal{H} . ◀

Those unitary operators that are used in quantum computing are then called **quantum gates**. To be more precise:

► **Definiton 2.17** (Quantum gate). *Let there be a 2^n -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$, where $n \in \mathbb{N}$ and a unitary operator $U : \mathcal{H} \rightarrow \mathcal{H}$. U is then called a **quantum gate**.* ◀

In practice, quantum computers can have tens or even hundreds of qubits. Quantum gates, or simply **gates**, are unitary operators that manipulate these qubits. However, typically, gates operate only on a subset of all available qubits, not on all qubits.

► Note 2.18 (Quantum circuit). In a quantum system with n qubits, the initial state of such a system, denoted as $|\psi_0\rangle$, is described as $|\psi_0\rangle = |0\rangle^{\otimes n}$ in the Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$. In this initial state, each qubit is in the state $|0\rangle \in \mathbb{C}^2$. Then, one or more gates may or may not be applied to each qubit or to multiple qubits, producing the final state $|\psi\rangle \in \mathcal{H}$. This results in the use of m unitary operators of various sizes (the maximal size of an operator can be $2^n \times 2^n$ when it affects all qubits by acting on \mathcal{H}), where each operator affects one or more qubits. Due to each gate being unitary, after all gates are applied, the resulting quantum state $|\psi\rangle \in \mathcal{H}$ that represents the entire quantum system of n qubits can be described as one single unitary operator $U : \mathcal{H} \rightarrow \mathcal{H}$ applied on the whole initial state $|\psi_0\rangle$:

$$|\psi\rangle = U |\psi_0\rangle,$$

where the operator U is just a product of all m individual gates (where each gate is appropriately enlarged using the tensor product with the identity operators). The operator U is then referred to as **quantum circuit**. Moreover:

- The number of qubits the operator U affects (in this case n) is called the **width** of the quantum circuit.
 - The **depth** of the quantum circuit is the number of gates that affect the qubit that has the most gates affecting it out of all the qubits. This is a vague description, so the use of this term is contextually dependent. ◀
- Note 2.19. All gates defined in the following two sections are defined in the standard basis, which means the $(|0\rangle, |1\rangle)$ basis for one-qubit operators and the $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$ basis for two-qubit operators. ◀

2.4 One-qubit quantum gates

In this section, the smallest possible gates are introduced. A gate is the smallest possible if it affects only one qubit. These gates are crucial for various quantum applications, as they allow precise control over the state of a single qubit.

Some single-qubit gates are more commonly used than others. For a one-qubit quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \in \mathbb{C}^2$, this section introduces several quantum gates that are among the most widely used².

2.4.1 Gate X

A classical bit b can be inverted using the logical *NOT* gate: $b \xrightarrow{NOT} \neg b$. In quantum computing, an analogous quantum gate to the logical *NOT* gate would need to invert the state of a qubit: $|0\rangle \leftrightarrow |1\rangle$, $|1\rangle \leftrightarrow |0\rangle$. A unitary operator acting this way is labeled as **X**. This gate needs to act as follows: $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$. This behavior is achievable by using the following matrix that represents the gate X :

► **Definiton 2.20** (Gate X). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$. The following quantum gate is defined:*

$$X : \mathcal{H} \rightarrow \mathcal{H}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The figure below depicts this gate being applied to a qubit in a one-qubit quantum circuit:



■ **Figure 2.1** A visualization of the gate X applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.

► **Corollary 2.21.** *Let $|\psi\rangle = (\alpha \ \beta)^T \in \mathbb{C}^2$ be a one-qubit quantum state. Then: $X|\psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$. The gate X does indeed swap the first and the second element of $|\psi\rangle$.*

It should be noted that the gate X possesses several unique properties. It is a *Hermitian operator* ($X^\dagger = X$), it is *involutory* ($X^2 = I$), it has a determinant of -1 and its eigenvalues are ± 1 . This gate is not a single gate with these properties. In fact, this unitary operator is one of three unitary operators that share these exact properties, and they are represented by matrices called the *Pauli matrices*, denoted as $\sigma_x, \sigma_y, \sigma_z$. The gate X is represented by the Pauli matrix σ_x .

²For the sake of brevity, this section is not exhaustive in terms of the number of commonly used gates it introduces. It only introduces the gates that are directly used in the encoding methods presented further in this thesis.

2.4.2 Gate Y

The next Pauli matrix, σ_y , specifies a gate called the gate **Y**:

► **Definiton 2.22** (Gate Y). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$. The following quantum gate is defined:*

$$Y : \mathcal{H} \rightarrow \mathcal{H}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.$$

The image below illustrates the application of this gate to a qubit within a single-qubit quantum circuit:



■ **Figure 2.2** A visualization of the gate Y applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.

► **Corollary 2.23.** *Let $|\psi\rangle = (\alpha \ \beta)^T \in \mathbb{C}^2$ be a one-qubit quantum state. Then: $Y|\psi\rangle = Y(\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} -i\beta \\ i\alpha \end{pmatrix}$.*

The gate Y acts on the state $|\psi\rangle$ in following ways:

- It affects the state in the same way as the gate X does—by switching the elements.
- It negates the first element of the state. Negating the first element's amplitude effectively shifts its relative phase by π radians relative to the second element's relative phase.
- It acts on the whole state with a global phase shift by adding a complex part i to both amplitudes, which corresponds to a phase shift of $\pi/2$ radians.

So, the resulting phase shifts are $3\pi/2 = -\pi/2$ on the first state vector element and $\pi/2$ on the second one.

2.4.3 Gate Z

Lastly, the σ_z matrix completes the set of Pauli matrices. This final Pauli matrix represents the gate **Z**:

► **Definiton 2.24** (Gate Z). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$. The following quantum gate is defined:*

$$Z : \mathcal{H} \rightarrow \mathcal{H}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Presented below is an illustration depicting the application of this gate to a qubit in a quantum circuit containing one qubit:



■ **Figure 2.3** A visualization of the gate Z applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.

► **Corollary 2.25.** Let $|\psi\rangle = (\alpha \ \beta)^T \in \mathbb{C}^2$ be a one-qubit quantum state. Then: $Z|\psi\rangle = Z(\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ -\beta \end{pmatrix}$.

- This gate affects the state by shifting the second element's relative phase by π radians. ◀

The effect of these gates (or any single-qubit gates, for that matter) can be visualized using the Bloch sphere. The effect of any single-qubit quantum gate can be visualized on the Bloch sphere. The reason why the gates X , Y , and Z are called that way is because they are known for rotating quantum states around the X , Y , and Z axes of the Bloch sphere, respectively. However, these rotations are fixed to specific angles, meaning they flip the state of a qubit along the respective axes—for instance, the gate X acts by flipping the state around the X -axis. The gates Y and Z similarly rotate the state around the Y and Z axes, respectively.

While any one-qubit quantum state can be created by using these fixed-angle rotations in combination, it requires careful computation to determine the precise sequence of the gates X , Y , and Z to achieve the desired state. In many quantum algorithms, it is advantageous to generalize these rotations to allow for arbitrary angles. This generalization would not only simplify the process of state preparation, but would also significantly reduce the computational overhead since a single application of a generalized gate could achieve the desired state much more quickly. Such flexibility would allow for more precise control over the rotations. The gates that accomplish this goal exist; they are introduced in the following subsection.

In this subsection, generalized rotational gates are presented that allow the rotation of qubits around the X , Y , and Z axes of the Bloch sphere by a given angle. The first generalized rotational gate is a gate called **RX** , which generalizes the gate X .

2.4.4 Gate RX

The gate RX rotates a qubit state around the X -axis of the Bloch sphere by an arbitrary angle θ . This gate is an essential component in quantum computing as it allows for the creation of superposition states and facilitates the implementation of arbitrary rotations.

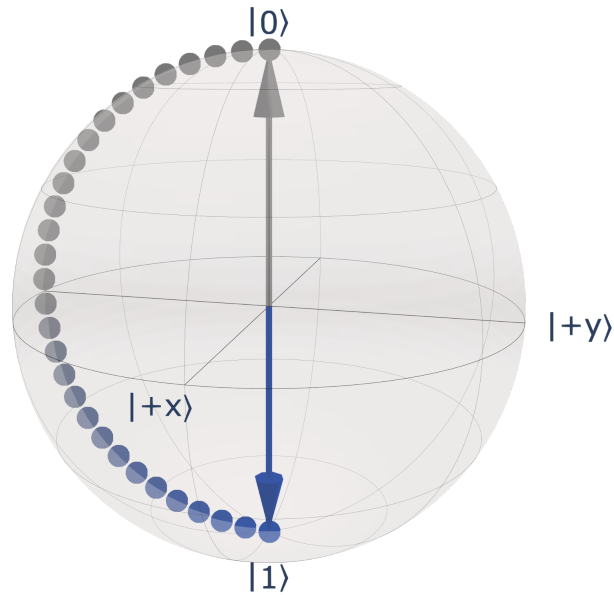
► **Definiton 2.26** (Gate RX). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$ and a given angle $\theta \in \mathbb{R}$. The following quantum gate is defined:*

$$RX(\theta) : \mathcal{H} \rightarrow \mathcal{H}, \quad RX(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}.$$

Below is a diagram showing how this gate is illustrated when applied to a qubit within a single-qubit quantum circuit and also a diagram where the effect of this gate on an initial state $|0\rangle$ is visualized using the Bloch sphere:



■ **Figure 2.4** A visualization of the gate $RX(\pi)$ applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.



■ **Figure 2.5** A visualization using the Bloch sphere of the gate $RX(\pi)$ applied to a qubit in a one-qubit quantum circuit, where the qubit is initially in the state $|0\rangle$ (gray arrow). This outcome is equivalent to the application of the standard gate X on the state $|0\rangle$.

2.4.5 Gate RY

The gate RY performs a rotation around the Y -axis of the Bloch sphere by a given angle θ . This gate, like the gate RX , is crucial for generating superposi-

tions and for enabling arbitrary single-qubit transformations. This capability is particularly useful in quantum algorithms that require rotations in the Y -axis, as is the case in most of the encoding methods presented in this thesis.

► **Definiton 2.27** (Gate R_Y). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$ and a given angle $\theta \in \mathbb{R}$. The following quantum gate is defined:*

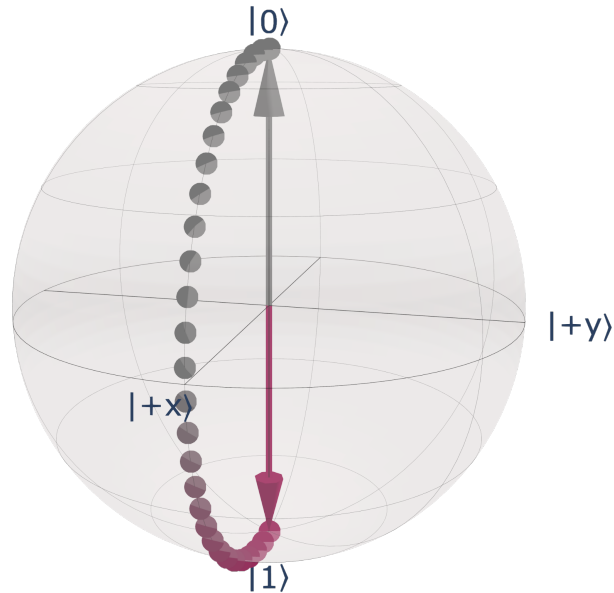
$$R_Y(\theta) : \mathcal{H} \rightarrow \mathcal{H}, \quad R_Y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}.$$



The following figure depicts the representation of this gate when it is used on a qubit in a single-qubit quantum circuit:



■ **Figure 2.6** A visualization of the gate $R_Y\left(\frac{\pi}{2}\right)$ applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.



■ **Figure 2.7** A visualization using the Bloch sphere of the gate $R_Y(\pi)$ applied to a qubit in a one-qubit quantum circuit, where the qubit is initially in the state $|0\rangle$ (gray arrow). This outcome is equivalent to the application of the standard gate Y on the state $|0\rangle$.

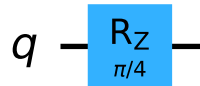
2.4.6 Gate RZ

The gate RZ completes the set of arbitrary single-qubit rotational gates. It is used to perform a rotation around the Z -axis of the Bloch sphere by an angle θ . This gate is particularly important for adjusting the phase of quantum states. The gate RZ changes the relative phase between the basis states $|0\rangle$ and $|1\rangle$, which is crucial in many quantum algorithms that involve phase-sensitive operations.

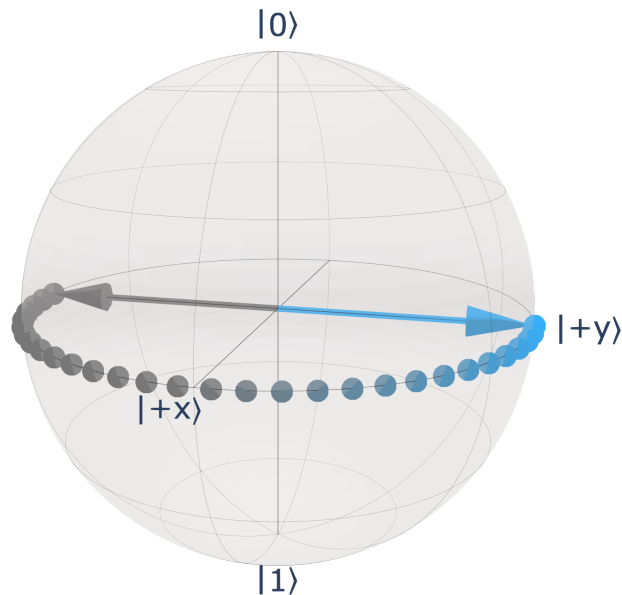
► **Definiton 2.28** (Gate RZ). *Let there be a two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$ and a given angle $\theta \in \mathbb{R}$. The following quantum gate is defined:*

$$RZ(\theta) : \mathcal{H} \rightarrow \mathcal{H}, \quad RZ(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}.$$

The figure below illustrates how this gate is represented when applied to a qubit within a single-qubit quantum circuit:



■ **Figure 2.8** A visualization of the gate $RZ(\frac{\pi}{4})$ applied to a qubit in a one-qubit quantum circuit created using the Qiskit library.



■ **Figure 2.9** A visualization using the Bloch sphere of the gate $RZ(\pi)$ applied to a qubit in a one-qubit quantum circuit, where the qubit is initially in the state $|+y\rangle$ (gray arrow). This outcome is equivalent to the application of the standard gate Z on the state $|+y\rangle$.

Now, all single-qubit gates are defined and established³. Using these gates, any single-qubit quantum state can be prepared. However, if states of multiple qubits need to be prepared, certain multi-qubit states cannot be created using only these single-qubit gates. This limitation arises because the interactions and entanglements required for multi-qubit states extend beyond the capability of single-qubit operations. Therefore, multi-qubit gates exist to facilitate these more complex operations that span more than one qubit. The next section introduces some of these essential multi-qubit gates.

2.5 Two-qubit quantum gates

In quantum computing, the ability to manipulate and entangle multiple qubits is essential to perform more complex computations and implement various quantum algorithms. While single-qubit gates allow for the preparation and manipulation of individual qubit states, they are insufficient for creating entanglement between qubits or for performing operations that involve interactions between multiple qubits. This is where two-qubit gates become crucial.

Two-qubit gates enable the entanglement of qubits, a fundamental aspect in quantum computing that allows qubits to exhibit correlations that are not possible in classical systems. These gates also facilitate operations that involve conditional logic, where the state of one qubit can influence the operation performed on another qubit. Such interactions are vital for implementing some of the data encoding methods discussed in this thesis.

The main purpose of this section is to introduce the most widely known two-qubit gate, presenting its definition, properties, and the role it plays in quantum computing. That gate is referred to as the gate *CX*.

2.5.1 Gate *CX*

In a nutshell, this gate is a two-qubit version of the gate *X*, where the gate *X* is applied to one qubit only when the other qubit is in the state $|1\rangle$. Its name is *CX* because the gate *X* is conditionally applied to one of the qubits, and the letter “C” at the beginning is an abbreviation for the word “controlled”. This gate is also alternatively known as the gate *CNOT*.

The gate *CX* is one of the most fundamental and widely used two-qubit gates in quantum computing. It serves as the quantum analog of the classical XOR gate and plays a crucial role in creating entanglement between qubits.

In this gate, one qubit is designated as the **control qubit** and the other as the **target qubit**. The operation of the gate is such that it applies the gate *X* to the target qubit only when the control qubit is in the state $|1\rangle$. If the control qubit is in the state $|0\rangle$, the target qubit remains unchanged. And when it is said that a control qubit “is in the state”, it means that hypothetically,

³Ones that are used in this thesis.

if measured, the control qubit collapses to either $|0\rangle$ or $|1\rangle$, with the gate operation on the target qubit reflecting on this collapse and either applying the controlled operation or not applying it, based on collapsed state.

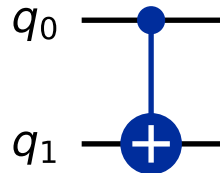
The ability of the gate CX to entangle qubits is a critical aspect of its functionality. When applied to a pair of qubits where the control qubit is in a superposition state, this gate entangles the two qubits. For example, if the control qubit is in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and the target qubit is initially in the state $|0\rangle$, the gate CX transforms the combined state into the entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This gate is formally defined below:

► **Definiton 2.29** (Gate CX). *Let there be a four-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^4$. The following quantum gate is defined:*

$$CX : \mathcal{H} \rightarrow \mathcal{H}, \quad CX = I \otimes |0\rangle\langle 0| + X \otimes |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

► Note 2.30. The definition above assumes that the little-endian convention is used where the least significant qubit is chosen as the control qubit. ◀

The following figure demonstrates the representation of this gate when used in a two-qubit quantum circuit:



■ **Figure 2.10** A visualization of the gate CX applied to a two-qubit Qiskit quantum circuit, with the qubit q_1 serving as the target qubit and the qubit q_0 as the control qubit.

2.5.2 Gate SWAP

The gate called $SWAP$ is another important two-qubit gate in quantum computing. Its primary function is to exchange the states of two qubits. This gate is particularly useful in quantum circuits for rearranging qubits and ensuring that the correct qubits are in the right positions for subsequent operations.

The gate $SWAP$ can be decomposed into a series of three gates CX . This decomposition is significant because it demonstrates that the $SWAP$ operation can be implemented using the more fundamental gate $CNOT$. The sequence of operations for the gate $SWAP$ using gates CX is as follows:

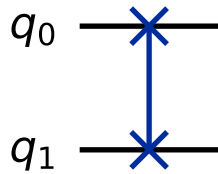
1. The gate CX with the first qubit as the control qubit and the second qubit as the target qubit is applied.
2. Another gate CX is applied with the second qubit as the control qubit and the first qubit as the target qubit.
3. The third gate CX is applied in the same way as the first was.

These three gate applications can be combined into one gate $SWAP$, with its matrix representation defined below:

► **Definiton 2.31** (Gate $SWAP$). *Let there be a four-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^4$. The following quantum gate is defined:*

$$SWAP : \mathcal{H} \rightarrow \mathcal{H}, \quad SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The following figure demonstrates the representation of this gate when used in a two-qubit quantum circuit:



■ **Figure 2.11** A visualization of the gate $SWAP$ applied to a two-qubit quantum circuit created using the Qiskit library.

Thus, the gate $SWAP$ effectively exchanges the states of two given qubits. In summary, this gate is essential for reordering qubits in a quantum circuit. Its implementation using the gates CX highlights the utility of the gate CX as a building block for more complex quantum operations. Understanding this gate is crucial for one of the data encoding methods presented in this thesis.

2.6 State preparation quality evaluation

In quantum computing, it is often necessary to evaluate how different two quantum states are from each other. This evaluation is particularly important for state preparation, where a desired quantum state is specified, and the actual state produced by quantum gates needs to be compared to this target state. Accurate state preparation is crucial for the reliability and efficiency of quantum algorithms and applications that require high-quality state preparation.

One significant application where state preparation quality evaluation is essential is data encoding, which is the focus of this thesis. In such scenarios, the fidelity of the encoded data compared to the expected results directly impacts the performance and accuracy of subsequent algorithms that process the encoded data. For instance, if data is encoded into quantum states for processing or storage, it is vital to ensure that the states have been prepared accurately to maintain data integrity.

In this section, two quality evaluation methods are presented. These methods provide the necessary tools to quantify the extent of differences between quantum states, ensuring that the prepared states meet the desired criteria. The use of these metrics allows for the assessment and evaluation of state preparation techniques, which is a focal point of interest in this thesis.

2.6.1 State fidelity

State fidelity is a widely used metric for quantifying the similarity between two quantum states. It is particularly useful for evaluating the accuracy of state preparation in quantum computing. For two pure quantum states, $|\psi\rangle$ and $|\phi\rangle$, the fidelity F is defined as the absolute square of the inner product between these states:

► **Definiton 2.32.** *Let there be a Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, $n \in \mathbb{N}$ and two pure quantum states $|\psi\rangle, |\phi\rangle \in \mathcal{H}$. Then, the fidelity between these two states $F(|\psi\rangle, |\phi\rangle)$ is defined as follows:*

$$F(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2. \quad \blacktriangleleft$$

This formula provides a measure of how close the state $|\psi\rangle$ is to the state $|\phi\rangle$. The fidelity ranges from 0 to 1, where $F = 1$ indicates that the states are identical, and $F = 0$ indicates that the states are orthogonal and thus completely different.

In the context of state preparation, fidelity can be used to compare a prepared quantum state $|\psi_{\text{prepared}}\rangle$ with the desired target state $|\psi_{\text{target}}\rangle$. High fidelity indicates that the prepared state is very close to the target state, which is crucial for the accuracy and reliability of quantum computations and data encoding processes.

2.6.2 Trace Distance

The trace distance is another important metric that is used to quantify the difference between two quantum states. It is defined as follows:

► **Definiton 2.33.** *Let there be a Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, $n \in \mathbb{N}$ and two pure quantum states $|\psi\rangle, |\phi\rangle \in \mathcal{H}$. Then, the trace distance between these two states $D(|\psi\rangle, |\phi\rangle)$ is defined as follows:*

$$D(|\psi\rangle, |\phi\rangle) = \sqrt{1 - F(|\psi\rangle, |\phi\rangle)}$$

where $F(|\psi\rangle, |\phi\rangle)$ is the fidelity between the two states. ▶

The trace distance ranges from 0 to 1, where $D = 0$ indicates that the states are identical, and $D = 1$ indicates that the states are completely distinguishable. A small trace distance indicates that the prepared state is close to the target state, which is important for ensuring the accuracy of the data representation.

Both fidelity and trace distance provide valuable insights into the quality of quantum state preparation. By using these metrics, the performance of the encoding methods introduced in this thesis is evaluated.

This marks the conclusion of the introductory chapter ⁴.

⁴This chapter has been designed to present certain essential foundational concepts of quantum computing necessary to comprehend the topics explored in the ensuing chapters. Although not all preliminary aspects of quantum computing have been addressed within this chapter, and those that have been introduced were presented in a simplified manner, delving further into the extensive array of basic concepts would not only extend beyond the intended scope of this thesis, but would also offer limited additional informational value to the overall discourse. Therefore, the aim of this chapter was to strike a deliberate balance, providing some requisite knowledge to engage with the discussions that follow.

Data encoding methods

As described in section Thesis scope (1.2), the primary goal of this thesis is to introduce certain data encoding solutions. More precisely, this chapter is fully devoted to introducing these four data encoding methods [25, 27, 26, 29, 30]:

1. The basis encoding method.
2. The angle encoding method.
3. The amplitude encoding method.
4. The divide-and-conquer encoding method (based on the amplitude encoding method).

All of these data encoding techniques are introduced by defining and describing them one by one in the order they appear in the list above. After explaining these encoding methods in this chapter, they are then used in the following chapters, where their implementation is discussed, and their real hardware performance is examined. This chapter starts by presenting the most simple encoding method of all four—the **basis encoding method**.

3.1 Basis encoding method

Let \mathbf{s} be a datapoint in the simplest form possible—a binary string, only containing either 0 or 1 as its elements: $\mathbf{s} = b_1 b_2 b_3 \dots b_n$, where¹ $\forall i \in \hat{n} : b_i \in \{0, 1\}$. *The purpose of the basis encoding method is to represent the string \mathbf{s} in the state of a quantum system.*

► Note 3.1. For the sake of formalism, the definitions contained in this chapter define that the encoding methods encode datapoints that are in the form of vectors of some vector space \mathcal{F}^n , $n \in \mathbb{N}$. ◀

¹The \hat{n} used in this thesis is defined as: $\hat{n} = \{k \in \mathbb{N} \mid 1 \leq k \leq n\}$

The basis encoding method is capable of encoding datapoints that are in the form of vectors of the vector space \mathbb{Z}_2^n , $n \in \mathbb{N}$ over the finite field \mathbb{Z}_2 (all elements of such a vector are equal to either 0 or 1). So instead of the string \mathbf{s} , a vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n$, $n \in \mathbb{N}$ is used as an object that is encoded into the state of a quantum system. This encoding method is simple in that it encodes each input vector element using a unique qubit. That results in this method requiring n qubits to encode an n -dimensional input vector. Moreover, the essence of this encoding method is that each resulting qubit is in one of just two possible states—either in the state $|0\rangle$ or in the state $|1\rangle$. This essence is outlined in the following note:

► Note 3.2 (Basis encoding method purpose). This encoding method works by always creating a state whose state vector is equal to one of the standard **basis** vectors—that is the reason why this encoding method is called the **basis** encoding method. However, it is essential to clarify which exact basis vector this encoding method produces for any given input. The answer is straightforward, as elements of a binary input vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n$, $n \in \mathbb{N}$ are directly used to create the desired state $|\psi\rangle \in \mathbb{C}^{2^n}$ that represents \mathbf{b} :

$$\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \mapsto |b_1 b_2 b_3 \dots b_n\rangle = |\psi\rangle. \quad \blacktriangleleft$$

And to create the resulting $|\psi\rangle$ state, the X gates can be utilized. This fact is more apparent in the definition introduced in the subsection below.

3.1.1 Definition

The whole process of encoding an input using the basis encoding method is encompassed in the following formal definition:

► **Definiton 3.3** (Basis encoding method). *Let there be a 2^n -dimensional Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, where $n \in \mathbb{N}$, a vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n$ and a quantum state $|\psi_0\rangle = |0\rangle^{\otimes n} \in \mathcal{H}$. The following mapping is defined:*

$$\Phi : \mathbb{Z}_2^n \rightarrow U(2^n), \quad \Phi(\mathbf{b}) = \bigotimes_{i=1}^n X'_i,$$

where $U(2^n)$ represents the group of $2^n \times 2^n$ unitary matrices, each acting on the Hilbert space \mathcal{H} , and $\forall i \in \hat{n}$ is X'_i defined as the following unitary operator:

$$X'_i = \begin{cases} X & \text{if } b_i = 1, \\ I & \text{if } b_i = 0, \end{cases}$$

where X is the the unitary operator represented by the Pauli matrix σ_x and I is the identity operator on \mathbb{C}^2 .

Then, the act of encoding \mathbf{b} into the state $|\psi_0\rangle$ using the **basis encoding method** is defined as creating a state $|\psi\rangle \in \mathcal{H}$ by applying the unitary operator $U_{\mathbf{b}} = \Phi(\mathbf{b})$ on the state $|\psi_0\rangle$:

$$U_{\mathbf{b}} |0\rangle^{\otimes n} = |\psi\rangle. \quad \blacktriangleleft$$

► **Theorem 3.4.** *Let there be a 2^n -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$, where $n \in \mathbb{N}$, a vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n$, a quantum state $|\psi_0\rangle = |0\rangle^{\otimes n} \in \mathcal{H}$ and a state $|\psi\rangle \in \mathcal{H}$ created by encoding \mathbf{b} into the state $|\psi_0\rangle$ using the basis encoding method. Then:*

$$|\psi\rangle = |b_1 b_2 \dots b_n\rangle.$$

Proof. Each qubit of the state $|\psi_0\rangle = |0\rangle^{\otimes n}$ is in the state $|0\rangle$, meaning $\forall i \in \hat{n} : |\psi_{0,i}\rangle = |0\rangle$. During the encoding process, the operator $U_{\mathbf{b}}$ acts on the state $|\psi_0\rangle$, resulting in the X'_i gate being applied to each qubit. Thus, the action of the unitary operator $U_{\mathbf{b}}$ on $|0\rangle^{\otimes n}$ can be described as:

$$U_{\mathbf{b}} |0\rangle^{\otimes n} = (X'_1 |0\rangle) \otimes (X'_2 |0\rangle) \otimes \dots \otimes (X'_n |0\rangle).$$

The operator $U_{\mathbf{b}}$ satisfies the unitarity condition, as it is in and of itself a tensor product of the unitary operators X'_i , since the X'_i operation results in either the X gate or the identity I gate being applied to each qubit $|0\rangle$. There are two possible outcomes when applying the gate X'_i to $|0\rangle$:

$$X |0\rangle = |1\rangle, \quad I |0\rangle = |0\rangle.$$

So, $\forall i \in \hat{n}$:

- If $b_i = 1$, then $X'_i = X$ and $X'_i |0\rangle = X |0\rangle = |1\rangle$.
- If $b_i = 0$, then $X'_i = I$ and $X'_i |0\rangle = I |0\rangle = |0\rangle$.

Therefore:

$$U_{\mathbf{b}} |0\rangle^{\otimes n} = |b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_n\rangle = |b_1 b_2 \dots b_n\rangle. \quad \blacksquare$$

Thus, the basis encoding method conditionally uses the X gates based on the elements of a binary input vector, thereby encoding it.

3.1.2 Properties

There are several consequences worth mentioning that stem from the definition above. This subsection is devoted to noting these consequences and certain traits of this encoding method:

► **Remark 3.5 (Basis encoding method definition consequence).** It is assumed that a quantum system is described within an n -dimensional Hilbert space $\mathcal{H} \in \mathbb{C}^n$, $n \in \mathbb{N}$, with \mathcal{H} being equipped with the standard basis² and the initial state of the system is represented by the state $|\psi_0\rangle$, where³ $|\psi_0\rangle = (1 \ 0 \ \dots \ 0)^T$.

²To reiterate, only the standard basis is used throughout this thesis when working with vector spaces and qubit measurements.

³In practice, different quantum systems may have different initial states and may use different bases. In this thesis, it is assumed that the initial state of a quantum system when using the standard basis is always $|\psi_0\rangle = |0\rangle^{\otimes n} \in (\mathbb{C}^2)^{\otimes n}$, $n \in \mathbb{N}$.

This state vector corresponds to the first standard basis vector of the Hilbert space \mathcal{H} , denoted as $|e_1\rangle$, where k -th standard basis vector is denoted as $|e_k\rangle$.

The basis encoding method can be characterized by a transformation that transforms the initial state $|e_1\rangle$ representing the first basis vector to another basis vector of the Hilbert space \mathcal{H} . This transformation can be represented by a unitary operator U such that $U|e_1\rangle = |e_k\rangle$, where $|e_k\rangle$ is one of the basis vectors of \mathcal{H} , and is represented by $|e_k\rangle = (0 \dots 0 1 0 \dots 0)^T$, with the 1 located in the k -th position, where $k \in \mathbb{N}$ satisfies $1 \leq k \leq n$.

The matrix form of the operator U is a permutation matrix which, if applied on the initial state of the system $|e_1\rangle$, results in a swap between the first element and the k -th element of $|e_1\rangle$ to produce $|e_k\rangle$. The operator U is unitary, fulfilling the condition $U^\dagger U = U U^\dagger = I$, where I is the identity matrix on \mathcal{H} , ensuring that the transformation is reversible and preserves the norm of the state vector. The resulting state is one of the standard basis vectors, where all the amplitudes of the state except for one equal zero—so the resulting state is not in a superposition. ◀

The next part summarizes multiple important basis encoding method characteristics:

► **Remark 3.6** (Basis encoding method characteristics). Let there be a binary input vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n$, $n \in \mathbb{N}$ that is encoded using the basis encoding method, resulting in a state $|\psi\rangle$ described within a 2^n -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$.

- As explained in the definition above, this encoding method only uses the X gates to encode the input vector \mathbf{b} . The definition also uses the identity gates I (acting on \mathbb{C}^2), but in practical terms, there is no need to use them in the circuit, as they do not modify the qubit states. Although there might be some real use cases where the identity gates are utilized in quantum computers, they would only introduce errors if applied to qubits in this case.
- The number of gates required to encode \mathbf{b} is always k , where $0 \leq k \leq n$. More specifically, k equals the Hamming weight of the vector \mathbf{b} , denoted by $\text{wt}(\mathbf{b})$, which is defined as the number of elements in \mathbf{b} that equal 1:

$$\text{wt}(\mathbf{b}) = \sum_{i=1}^n b_i,$$

where the sum computes the total count of 1's in the vector \mathbf{b} .

- A maximum of one gate is applied on each qubit, so if $1 \leq k \leq n$, the depth of the resulting quantum circuit is equal to 1. This is an advantageous trait of this encoding method, as the depth of the circuit produced by this method is the lowest possible⁴.

⁴As a circuit with a length of 0 does not modify the initial state in any way, so a depth of at least one is required to alter the initial state.

- The width of quantum circuits produced by this method is where the main disadvantage of this method lies—the number of qubits used in the circuit is the same as the length of \mathbf{b} , meaning the length of the state vector representing the produced state is exponential (2^n) compared to the input vector's length (n).
- Coupled with the fact that only binary vectors can be encoded using this encoding technique, the practicality of this encoding technique is questionable. If, for example, a complex vector is supposed to be encoded using this method, it first needs to be somehow transformed into a binary vector, meaning that the binary vector would either need to be in a higher dimension than the complex vector to store all the information or some information would be lost during the transformation process to ensure the same dimensionality. To write the transformation process more formally:

Let there be an m -dimensional vector space \mathcal{F}^m over a field $\mathcal{F} \neq \mathbb{Z}_2$, where $m \in \mathbb{N}$, and input data in the form of a vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_m)^T \in \mathcal{F}^m$. This input vector \mathbf{v} is not binary; therefore, it is not possible for the basis encoding method to encode it directly. However, if there is a need for this vector to be encoded, the following mapping function must first be used: $f : \mathcal{F}^m \rightarrow \mathbb{Z}_2^n$, $n \in \mathbb{N}$ (it is noteworthy that n can differ from m). The mapping function f is then applied on \mathbf{v} to create a binary vector:

$$f(\mathbf{v}) = \mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)^T \in \mathbb{Z}_2^n.$$

Then, the vector \mathbf{b} (created by transforming \mathbf{v} into \mathbf{b} with the help of f) can be used by the basis encoding method to encode it and create a state that represents it.

- The fact that this encoding method only allows for binary vector encoding means one of the most significant benefits of quantum computing is not utilized. As discussed previously (2.1.1), as opposed to bits, qubits are able to store much more information. In this case, the resulting qubits are either in the state $|0\rangle$ or in the state $|1\rangle$. As a result, the qubits cannot take advantage of qubit superposition (they do not utilize what the Hilbert space offers), which is one of the fundamental principles of quantum computing. Each qubit effectively acts as an ordinary classical bit, thus wasting the potential quantum computing brings.
- Another apparent characteristic of this encoding method is that it produces qubits that are not entangled.
- There is often a need to encode more than just one datapoint. When given a dataset of binary vectors, the entire dataset can be encoded using the basis encoding method if all the vectors in the dataset are concentrated into a single vector. Then, such a concentrated vector can be used as an input. The process is described below:

Let there be a dataset D , where $D = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m\}$, $m \in \mathbf{N}$ is a set of m binary vectors of dimension⁵ $n \in \mathbf{N}$, such that $\forall \mathbf{b}_i \in D : \mathbf{b}_i \in \mathbb{Z}_2^n$.

The following mapping function $f : D \rightarrow \mathbb{Z}_2^{mn}$ is defined as:

$$f(D) = \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \dots \oplus \mathbf{b}_m = \mathbf{b} \in \mathbb{Z}_2^{mn},$$

where \oplus denotes the vector concatenation operation.

The vector \mathbf{b} can then be used as an input in the basis encoding method. ◀

3.1.3 Usage example

The final part of this introduction to the basis encoding method involves demonstrating this encoding method on a short, specific binary input vector:

► **Example 3.7** (Basis encoding method usage example). Let there be:

- A quantum system in a 8-dimensional Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes 3}$, with the system being in the following initial state: $|\psi_0\rangle = |0\rangle^{\otimes 3} = |000\rangle \in \mathcal{H}$.
- A binary input vector $\mathbf{b} = (1\ 0\ 1)^T \in \mathbb{Z}_2^3$.

So, if the input vector is $(1\ 0\ 1)^T \in \mathbb{Z}_2^3$, then the basis encoding method should create a state $|\psi\rangle = |101\rangle \in \mathcal{H}$.

Based on the definition of this encoding method, a unitary operator $U_{\mathbf{b}}$ is created, where $U_{\mathbf{b}} = X \otimes I \otimes X$ and $U_{\mathbf{b}}$ acts on \mathcal{H} . $U_{\mathbf{b}}$ is then used to create the desired state $|\psi\rangle = |101\rangle \in \mathcal{H}$ by applying $U_{\mathbf{b}}$ on the initial state of the quantum system $|\psi_0\rangle = |000\rangle \in \mathcal{H}$ in the following way:

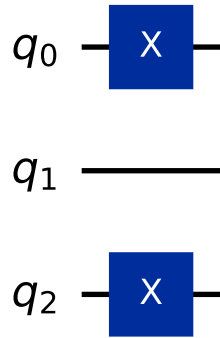
$$U_{\mathbf{b}} |\psi_0\rangle = (X \otimes I \otimes X) |000\rangle, \text{ where}$$

$$(X \otimes I \otimes X) |000\rangle = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |101\rangle = |\psi\rangle$$

The act of applying the operator $U_{\mathbf{b}}$ on the initial state $|000\rangle$ does indeed produce the desired state $|\psi\rangle = |101\rangle$. In practical terms, this can be achieved by applying two X gates in an empty three-qubit quantum circuit, one on

⁵For the sake of brevity, situations where the dataset contains vectors of different lengths are not considered.

the first qubit and one on the third qubit. The implementation algorithm is explained in more detail in the next chapter (4.2.1). Here is a figure depicting such a quantum circuit producing the desired state $|\psi\rangle = |101\rangle$:



■ **Figure 3.1** A visualization of a quantum circuit (created using the Qiskit library) producing the state $|101\rangle \in \mathbb{C}^8$ using the basis encoding method. ◀

3.2 Angle encoding method

The basis encoding method (3.1) is restrictive, as it can encode only binary vectors. Overcoming this limitation would result in more data being encoded in the same amount of qubits, thus providing greater encoding capabilities.

The second data encoding method presented in this chapter is the **angle encoding method**. The purpose of this encoding method is to overcome this restriction of only being able to encode binary vectors. When contrasting the angle encoding method with the basis encoding method, this encoding method exhibits two significant characteristics that define it. One of these characteristics is shared by both methods, while the other one differentiates this method from the basis encoding method:

- Notably, one crucial trait that both of these encoding methods share is that the amount of qubits they require equals the length of a given input vector. So, when input vectors of dimension $n \in \mathbb{N}$ are encoded using these encoding methods, the resulting states produced by both of these encoding methods are represented by 2^n -dimensional state vectors.
- This second characteristic is where both encoding methods differ from each other. Compared to the basis encoding method, this encoding method provides a transformative leap in terms of its encoding capacity—it allows for vectors other than just binary vectors to serve as input vectors. Specifically, it allows for real vectors to be encoded. Instead of only binary vectors $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_m)^T \in \mathbb{Z}_2^m$, $m \in \mathbb{N}$, the method can encode normalized real vectors $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n \in \mathbb{N}$, $\|\mathbf{v}\| = 1$. This means that

given the same input vector length $m = n$, the angle encoding method is able to encode a greater amount of information than the basis encoding method (assuming that \mathbf{b} is encoded using the basis encoding method and \mathbf{v} is encoded using the angle encoding method).

The ability of this encoding approach to encode real vectors originates from the fact that, contrary to the basis encoding method, the angle encoding method manages to utilize the Hilbert space of each qubit to a much larger extent, meaning it takes advantage of qubit superposition. It achieves this by using unitary operators that manipulate qubits in a way that results in the creation of states other than $|0\rangle$ and $|1\rangle$. The effect of unitary operators can be visualized using the Bloch sphere (2.2), where unitary operators “rotate” qubits by some *angle* θ around some Bloch sphere axis, which is why this encoding method is called the *angle* encoding method. The appropriate angles are first calculated based on the input vector elements, and then those angles are used to rotate the qubits with the help of rotational gates. However, it is crucial to explain the process of angle calculation—how are the necessary angles calculated when given an input vector. This process stems directly from the very nature of the angle encoding method itself, which is presented below:

► Note 3.8 (Angle encoding method purpose). Given a real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n \in \mathbb{N}$, where $\|\mathbf{v}\| = 1$, the objective of the angle encoding method is to create a quantum state $|\psi\rangle$ with n qubits in which $\forall i \in \hat{n}$ the probability that the i -th qubit of the state $|\psi\rangle$ collapses into the state $|1\rangle$ during a measurement equals v_i^2 , where v_i is the i -th element of \mathbf{v} . ◀

Thus, the essence of this encoding method is simple—the elements of a given input vector define the desired probabilities of qubits collapsing into the state $|1\rangle$. This is further explored in much greater detail in the following note:

► Note 3.9 (Angle encoding method explanation). Given a real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n \in \mathbb{N}$, where $\|\mathbf{v}\| = 1$, a Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, a quantum system with the initial state being $|\psi_0\rangle = |0\rangle^{\otimes n} \in \mathcal{H}$, a state $|\psi\rangle \in \mathcal{H}$ is created by encoding the vector \mathbf{v} into the state $|\psi_0\rangle$ using the angle encoding method.

The magnitude of the real input vector \mathbf{v} must be equal to 1 to ensure that $\forall v_i \in \mathbf{v} : v_i^2 \in [0, 1] \subset \mathbb{R}$. This is because the probability that any qubit collapses into any possible state when measured can also only range from 0 to 1. Generally, the probability of a measured qubit $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ collapsing into the state $|1\rangle$ equals $P_{|1\rangle}(|\phi\rangle) = |\beta|^2$. Therefore, based on the nature of the angle encoding method, it must be true for the encoded state $|\psi\rangle$ that $\forall i \in \hat{n} : |v_i|^2 = |\beta_i|^2$, where β_i is the second amplitude of the i -th qubit $|\psi_i\rangle = \alpha_i|0\rangle + \beta_i|1\rangle$. This can be described as: $P_{|1\rangle}(|\psi_i\rangle) = |\beta_i|^2 = |v_i|^2$.

So, $\forall i \in \hat{n}$, the i -th qubit $|\psi_{0,i}\rangle$ of the initial state $|\psi_0\rangle = |0\rangle^{\otimes n}$ must somehow be transformed to become the i -th qubit $|\psi_i\rangle$ of the desired state $|\psi\rangle$, which is only possible when a unitary operator U_i (that acts on \mathbb{C}^2) is applied

on $|\psi_{0,i}\rangle$: $U_i |\psi_{0,i}\rangle = |\psi_i\rangle$. In a quantum circuit, there are infinite possible ways to achieve this effect by sequentially applying quantum gates to a qubit. But understandably, the goal is for this encoding process to be as simple and efficient as possible—ideally using a single gate on each qubit such that it can $\forall v_i \in \mathbf{c}$ produce the desired qubit state $|\psi_i\rangle$.

That implies that such a gate must be able to produce the i -th state $|\psi_i\rangle$ for every i -th desired probability v_i^2 . These probabilities can vary, so for this gate to account for every possible probability $v_i^2 \in [0, 1] \subset \mathbb{R}$, the gate must be able $\forall v_i^2 \in [0, 1] \subset \mathbb{R}$ to produce a state $|\phi\rangle$ such that $P_{|1\rangle}(|\phi\rangle) = v_i^2$. This can be achieved by various gates.

As discussed previously in the section One-qubit quantum gates (2.4), the rotational gates RX , RY , and RZ rotate qubits around the X , Y , and Z axes of the Bloch sphere, respectively. Of these, the gates RX and RY are able to create states whose probabilities of collapsing into the state $|1\rangle$ when measured can take any value from the interval $[0, 1] \subset \mathbb{R}$. So, both gates RX and RY are suitable candidates for use in the angle encoding method, since they are commonly known, widely used, and capable of creating any needed qubit state. In this thesis, the gate RY has been chosen. \blacktriangleleft

Now that it is clear that the rotational gate RY can be used to transform each qubit into the desired state, it is important to clarify how to calculate the angle that should be used in the RY rotation for any given input vector element. Let $v_i \in \mathbb{R}$ be an arbitrary input vector element and $|0\rangle \in \mathbb{C}^2$ the initial state of a qubit. The angle encoding method then uses the gate RY to transform the initial state $|0\rangle \in \mathbb{C}^2$ into the desired state $|\psi\rangle \in \mathbb{C}^2$ by applying the unitary operator $RY(\theta)$, $\theta \in \mathbb{R}$ on the initial state $|0\rangle$ in the following way:

$$\begin{aligned} RY(\theta) |0\rangle &= \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) \end{pmatrix} = \\ &= \cos\left(\frac{\theta}{2}\right) |0\rangle + \sin\left(\frac{\theta}{2}\right) |1\rangle = |\psi\rangle, \end{aligned}$$

where, to satisfy the main requirement of this encoding method, the squared second amplitude of the resulting state $|\psi\rangle$ must be equal to the desired probability (because only then the probability of the state $|\psi\rangle$ collapsing into the state $|1\rangle$ when measured equals the desired probability v_i^2):

$$\sin^2\left(\frac{\theta}{2}\right) = v_i^2.$$

This equation needs to be solved for the variable θ , because it is used in the rotation $RY(\theta)$:

$$\begin{aligned} \sin^2\left(\frac{\theta}{2}\right) = v_i^2 &\Leftrightarrow \sin\left(\frac{\theta}{2}\right) = \pm v_i \Leftrightarrow \\ \Leftrightarrow \frac{\theta}{2} = \arcsin(\pm v_i) &\Leftrightarrow \theta = 2 \arcsin(\pm v_i) \end{aligned}$$

Thus, the angle θ used in the gate $RY(\theta)$ can be calculated using the equation $\theta = 2 \arcsin(\pm v_i)$. In terms of qubit measurements, it does not matter whether v_i or $-v_i$ is used in the equation because, as explained later, when measuring the desired state $|\psi\rangle$ in the standard basis, the following is true:

$$P_{|0\rangle}(|\psi\rangle) = 1 - (v_i)^2 = 1 - (-v_i)^2, \quad P_{|1\rangle}(|\psi\rangle) = v_i^2 = (-v_i)^2,$$

meaning that when performing a measurement, the measurement outcomes are the same regardless of whether v_i or $-v_i$ is used. However, as explained later, the resulting state $|\psi\rangle$ is described as:

$$|\psi\rangle = \sqrt{1 - v_i^2} |0\rangle + v_i |1\rangle,$$

and in this case, it does matter if v_i is used or $-v_i$ is used, as the sign can be encoded directly in the quantum state $|\psi\rangle$. This encoding method works by also encoding the sign into the quantum state $|\psi\rangle$, and to achieve this, the original input vector element with its sign is used in the equation when calculating the angles θ , which means that $\theta = 2 \arcsin(\text{sgn}(v_i) \cdot |v_i|) = 2 \arcsin(v_i)$.

3.2.1 Definition

Now, with the whole essence of the angle encoding method thoroughly explained, it is time to present a more formal definition:

► **Definiton 3.10** (Angle encoding method). *Let there be a 2^n -dimensional Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, where $n \in \mathbb{N}$, a vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$ normalized such that $\|\mathbf{v}\| = 1$ and a quantum state $|\psi_0\rangle = |0\rangle^{\otimes n} \in \mathcal{H}$. The following mapping is defined:*

$$\Psi : \mathbb{R}^n \rightarrow U(2^n), \quad \Psi(\mathbf{v}) = \bigotimes_{i=1}^n RY(\theta_i),$$

where $U(2^n)$ denotes the group of $2^n \times 2^n$ unitary matrices, each acting on the space \mathcal{H} , and $\forall i \in \hat{n}$ is $RY(\theta_i)$ defined as the following unitary operator:

$$RY(\theta_i) = \begin{pmatrix} \cos\left(\frac{\theta_i}{2}\right) & -\sin\left(\frac{\theta_i}{2}\right) \\ \sin\left(\frac{\theta_i}{2}\right) & \cos\left(\frac{\theta_i}{2}\right) \end{pmatrix}, \quad \text{where } \theta_i = 2 \arcsin(v_i).$$

Then, the act of encoding \mathbf{v} into the state $|\psi_0\rangle$ using the angle encoding method is defined as creating a state $|\psi\rangle \in \mathcal{H}$ by applying the unitary operator $U_{\mathbf{v}} = \Psi(\mathbf{v})$ on the state $|\psi_0\rangle$:

$$U_{\mathbf{v}} |0\rangle^{\otimes n} = |\psi\rangle. \quad \blacktriangleleft$$

► **Theorem 3.11.** *Let there be a 2^n -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$, where $n \in \mathbb{N}$, a vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$ normalized such that $\|\mathbf{v}\| = 1$, a quantum state $|\psi_0\rangle = |0\rangle^{\otimes n} \in \mathcal{H}$ and a state $|\psi\rangle \in \mathcal{H}$ created by encoding \mathbf{v} into the state $|\psi_0\rangle$ using the angle encoding method. Then $\forall i \in \hat{n} : P_{|1\rangle}(|\psi_i\rangle) = v_i^2$, where $|\psi_i\rangle$ is the i -th qubit of the state $|\psi\rangle$.*

Proof. Each qubit of the state $|\psi_0\rangle = |0\rangle^{\otimes n}$ is in the state $|0\rangle$, meaning $\forall i \in \hat{n} : |\psi_{0,i}\rangle = |0\rangle$. During the encoding process, the operator $U_{\mathbf{v}}$ acts on the state $|\psi_0\rangle$, resulting in the RY gate being applied to each qubit. Thus, the action of the unitary operator $U_{\mathbf{v}}$ on $|0\rangle^{\otimes n}$ can be described as:

$$U_{\mathbf{v}} |0\rangle^{\otimes n} = (RY(\theta_1) |0\rangle) \otimes (RY(\theta_2) |0\rangle) \otimes \cdots \otimes (RY(\theta_n) |0\rangle).$$

The operator $U_{\mathbf{v}}$ satisfies the unitarity condition, as it is in and of itself a tensor product of the unitary operators RY .

So, it is true $\forall i \in \hat{n}$ that the i -th qubit $|\psi_{0,i}\rangle = |0\rangle$ of the state $|\psi_0\rangle$ is acted upon by a gate $RY(\theta_i)$ (where $\theta_i = 2 \arcsin(v_i)$) to create the i -th qubit $|\psi_i\rangle$ of the desired state $|\psi\rangle$ in the following way: $|\psi_i\rangle = RY(\theta_i) |\psi_{0,i}\rangle$. That means:

$$\begin{aligned} RY(\theta_i) |\psi_{0,i}\rangle &= RY(2 \arcsin(v_i)) |0\rangle = \\ &= \cos\left(\frac{2 \arcsin(v_i)}{2}\right) |0\rangle + \sin\left(\frac{2 \arcsin(v_i)}{2}\right) |1\rangle = \\ &= \cos(\arcsin(v_i)) |0\rangle + \sin(\arcsin(v_i)) |1\rangle = \\ &= \sqrt{1 - v_i^2} |0\rangle + v_i |1\rangle = |\psi_i\rangle. \end{aligned}$$

Therefore:

$$\begin{aligned} P_{|0\rangle}(|\psi\rangle) &= P_{|0\rangle} \left(\sqrt{1 - v_i^2} |0\rangle + v_i |1\rangle \right) = \left(\sqrt{1 - v_i^2} \right)^2 = 1 - v_i^2 \\ P_{|1\rangle}(|\psi\rangle) &= P_{|1\rangle} \left(\sqrt{1 - v_i^2} |0\rangle + v_i |1\rangle \right) = v_i^2, \end{aligned}$$

meaning v_i^2 indeed defines the probability of the i -th qubit of the resulting state $|\psi\rangle$ collapsing into the state $|1\rangle$ when measured. ■

Thus, the proof above demonstrates that the angle encoding method works correctly when using the rotational gates $RY(\theta_i)$, with the angles θ_i being calculated using the formula $\theta_i = 2 \arcsin(v_i)$.

3.2.2 Properties

Several notable outcomes arise from the preceding definition. This subsection focuses on discussing these outcomes, and it summarizes some of the specific characteristics of this encoding approach:

► **Remark 3.12.** Let $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n, n \in \mathbb{N}$ be an input vector that is encoded using the angle encoding method, resulting in a state $|\psi\rangle$ described within a 2^n -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$.

- According to the definition, this encoding technique exclusively employs the rotational gates RY to encode the vector \mathbf{v} .

- Based solely on the definition, the number of gates used in the circuit is always equal to n . However, in some cases, there is no need to use a gate on each qubit. Instances can arise when a gate $R_Y(\theta)$ does not modify the state of a qubit (for example, when $\theta = 0$). So, the process of constructing a quantum circuit using this encoding method can be optimized by not utilizing those gates that do not change the state of a qubit.
- Each qubit uses at most one gate, resulting in the quantum circuit depth of 1, which is the minimal possible depth for quantum circuits (those that modify the state in any way). This characteristic is shared by both this encoding method and the basis encoding method.
- The number of qubits this encoding method utilizes matches the length of the input vector \mathbf{v} , which implies that the length of the state vector (which represents the resulting state $|\psi\rangle$) is exponentially larger (2^n) relative to the length of the input vector (n). This is a disadvantageous characteristic that both this encoding method and the basis encoding method share. That suggests that neither of these encoding methods utilizes the exponentiality that stems from the very nature of quantum computing (meaning that a state vector's length grows exponentially in relation to its qubit count).
- The practicality of this encoding method is more significant in comparison with the basis encoding method due to the fact that this method allows for real vector encoding. Allowing for real numbers to be encoded rather than just binary numbers makes this encoding approach much more flexible, as classical datasets often contain numbers other than just binary numbers. In the basis encoding method, a real number would first have to be converted into a binary form to be encoded. However, when this encoding approach is used, a real number can be encoded directly without the need for it to be transformed. Obviously, more complex data structures still have to be transformed into real vectors. A similar transformation (3.6) is discussed in the subsection Properties (3.1.2) of the basis encoding method. The same concept can be employed in this case as well (for example, in a case where a given input vector contains complex numbers).
- Another noticeable feature of this encoding technique is that it results in the qubits not being entangled.
- Even though this encoding approach allows for real vector encoding, it does not mean that this encoding approach is efficient in utilizing what quantum computing offers. Based on the actuality that this encoding method only uses the gates R_Y , it is clear that the Hilbert space is far from being fully utilized. While the basis encoding method produces qubits such that each qubit is in one of just two possible states, the angle encoding method produces qubits such that each qubit is in one of infinite possible states. Despite this fact, all the states that the angle encoding technique

produces form a circle on the surface of the Bloch sphere. An arbitrary two-dimensional quantum state can lie anywhere on the surface of the Bloch sphere. But when the gate RY is used to rotate the initial state $|0\rangle$, it only rotates the state around the Y axis, where the full range of motion forms a circle around the Y axis. The consequence of this is that almost the whole two-dimensional Hilbert space is still wasted, with the resulting qubits being constrained in one circle on the surface of the Bloch sphere. This consequence originates from the fact that this encoding method only encodes real numbers.

Although this encoding method only encodes real numbers, it does not mean that it is not possible to overcome this limitation. In fact, the angle encoding method could be expanded so that it also uses the gates RZ in addition to the gates RY . Hence, it is possible to fully utilize the whole surface of the Bloch sphere by using these two types of rotational gates. Thus, more information can be encoded from each input vector element, resulting in the ability to encode complex numbers.

So, if a complex number $c = re^{i\theta} \in \mathbb{C}$, $r \in \mathbb{R}$, $\theta \in [0, 2\pi] \subset \mathbb{R}$ is given such that $|c|^2 \in [0, 1] \subset \mathbb{R}$, the magnitude r of this given number can be encoded using the gate RY and the phase $e^{i\theta}$ of this number can be encoded using the gate RZ .

- This angle encoding method could also be defined alternatively, where the i -th input vector element would correspond to the probability of the i -th qubit collapsing into the state $|0\rangle$ when measured instead of the state $|1\rangle$. In such a case, the angle calculation equation would be $\theta_i = 2 \arccos(v_i)$.
- In this final list item, the relationship is explained between the input vector elements and the elements of the state vector that represents the state $|\psi\rangle$. Assuming that the standard basis is used with the Hilbert space $\mathcal{H} = \mathbb{C}^{2^n}$, the standard basis in use is comprised of 2^n standard basis states, where $\forall j \in \widehat{2^n}$ the j -th standard basis state is denoted as $|e_j\rangle$ and can be described as the following tensor product of n single-qubit states:

$$|e_j\rangle = \bigotimes_{k=1}^n |b_k\rangle,$$

where b_k is the k -th bit in the binary representation of the number $j - 1$, zero-padded at the beginning to n bits if necessary. So, this tensor product of the qubits $|b_k\rangle$ represents the state $|e_j\rangle$, with each qubit $|b_k\rangle$ equaling either the state $|0\rangle$ or the state $|1\rangle$, depending on the k -th bit b_k .

The state $|\psi\rangle$ can be represented as a superposition of all the standard basis states as follows:

$$|\psi\rangle = \sum_{j=1}^{2^n} \alpha_j |e_j\rangle,$$

where $\alpha_j \in \mathbb{C}$ is the amplitude of the j -th standard basis state. The link between the input vector elements and the amplitudes α_j can then be presented as a sum, where $\forall i \in \hat{n}$:

$$P_{|1\rangle}(|\psi_i\rangle) = v_i^2 = \sum_{e_j: b_k=1} \alpha_j^2, \quad \text{where:}$$

- v_i is the i -th element of the input vector \mathbf{v} ,
- $|\psi_i\rangle$ is the i -th qubit of the resulting state $|\psi\rangle$,
- $P_{|1\rangle}(|\psi_i\rangle)$ is the probability of the i -th qubit of the resulting state $|\psi\rangle$ collapsing into the state $|1\rangle$ when measured (the angle encoding method requires that this probability must be equal to v_i^2),
- α_j is the amplitude of the j -th standard basis state,
- and the sum itself extends over all j 's such that the j -th standard basis state has its k -th bit being equal to 1 ($b_k = 1$ is the k -th bit in the binary representation of the j -th standard basis state, and the condition in the sum requires the bit to be equal to 1).

This can be demonstrated on a three-qubit quantum state:

Let there be an input vector $\mathbf{v} = (v_1 \ v_2 \ v_3)^T \in \mathbb{R}^3$ encoded into a three-qubit quantum state $|\phi\rangle \in \mathbb{C}^8$ using the angle encoding method, where the first, second, and third qubits of the state $|\phi\rangle$ are denoted as $|\phi_1\rangle$, $|\phi_2\rangle$, and $|\phi_3\rangle$, respectively. The state $|\phi\rangle$ is a tensor product of its three qubits:

$$\begin{aligned} |\phi\rangle &= |\phi_1\rangle \otimes |\phi_2\rangle \otimes |\phi_3\rangle = \\ &= (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) \otimes (\alpha_3 |0\rangle + \beta_3 |1\rangle), \\ &\quad \text{where } \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3 \in \mathbb{C}. \end{aligned}$$

The state $|\phi\rangle$ can also be described as the following superposition of the standard basis states (assuming the standard basis on \mathbb{C}^8 is used):

$$\begin{aligned} |\phi\rangle &= \alpha |000\rangle + \beta |001\rangle + \gamma |010\rangle + \delta |011\rangle + \epsilon |100\rangle + \zeta |101\rangle + \eta |110\rangle + \theta |111\rangle, \\ &\quad \text{where } \alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta \in \mathbb{C}. \end{aligned}$$

Then, the following equations are presented, where $P_{|1\rangle}(|\phi_1\rangle)$, $P_{|1\rangle}(|\phi_2\rangle)$, and $P_{|1\rangle}(|\phi_3\rangle)$ are the probabilities of measuring the individual qubits $|\phi_1\rangle$, $|\phi_2\rangle$, and $|\phi_3\rangle$ in the state $|1\rangle$ (it is crucial to note that this thesis uses a convention where the rightmost digit in the ket notation corresponds to the first qubit of a quantum system):

$$\begin{aligned} P_{|1\rangle}(|\psi_1\rangle) &= v_1^2 = |\beta|^2 + |\delta|^2 + |\zeta|^2 + |\theta|^2, \\ P_{|1\rangle}(|\psi_2\rangle) &= v_2^2 = |\gamma|^2 + |\delta|^2 + |\eta|^2 + |\theta|^2, \\ P_{|1\rangle}(|\psi_3\rangle) &= v_3^2 = |\epsilon|^2 + |\zeta|^2 + |\eta|^2 + |\theta|^2. \end{aligned}$$

Due to the basis states being denoted as binary strings, it is possible to generalize the equation $P_{|1\rangle}(|\psi_i\rangle)$ for an i -th qubit $|\psi_i\rangle$. Given an input vector of length $n \in \mathbb{N}$, the resulting state is a superposition of $m = 2^n$ basis states, where the amplitude of the j -th basis state is denoted as α_j . To calculate each probability $P_{|1\rangle}(|\psi_i\rangle)$, exactly $\frac{m}{2}$ amplitudes are needed. This is because each qubit is in the state $|0\rangle$ in half of the basis states and in the state $|1\rangle$ in the other half of the basis states. To calculate $P_{|1\rangle}(|\psi_i\rangle)$, the last i squared amplitudes out of every $2 \cdot i$ amplitudes should be summed. This results in the following equation, where $\forall i \in \hat{n}$:

$$P_{|1\rangle}(|\psi_i\rangle) = \sum_{k=1}^{\frac{m}{2^i}} \sum_{l=0}^{i-1} |\alpha_{2^i k - l}|^2.$$

So for the three-qubit input vector example above, for the $P_{|1\rangle}(|\psi_1\rangle)$, the second, fourth, sixth, and eighth amplitudes were used. For the $P_{|1\rangle}(|\psi_2\rangle)$, the third, fourth, seventh, and eighth amplitude was used. And finally, for the $P_{|1\rangle}(|\psi_3\rangle)$, the fifth, sixth, seventh, and eighth amplitudes were used. ▶

3.2.3 Usage example

The conclusion to this angle encoding technique introduction showcases its application using a short real input vector:

▶ **Example 3.13** (Angle encoding method usage example). Let there be:

- A quantum system in a 4-dimensional Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes 2}$, with the system being in the following initial state: $|\psi_0\rangle = |0\rangle^{\otimes 2} = |00\rangle \in \mathcal{H}$.
- A real input vector $\mathbf{v} = (v_1 \ v_2)^T = \left(\frac{1}{\sqrt{2}} \ \frac{1}{\sqrt{2}}\right)^T \in \mathbb{R}^2$. The magnitude of this vector is equal to 1: $\|\mathbf{v}\| = 1$.

The objective is to encode the input vector \mathbf{v} using the angle encoding method. So, a state $|\psi\rangle$ should be created. Let the first and second qubits of the state $|\psi\rangle$ be denoted as $|\psi_1\rangle$ and $|\psi_2\rangle$, respectively. Then, based on the input vector elements v_1 and v_2 , the probability of each of these qubits collapsing into the state $|1\rangle$ when measured should be equal to:

$$P_{|1\rangle}(|\psi_1\rangle) = v_1^2 = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}, \quad P_{|1\rangle}(|\psi_2\rangle) = v_2^2 = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}.$$

Based on the definition of this encoding method, a unitary operator $U_{\mathbf{v}}$ that acts on \mathcal{H} is created, where $U_{\mathbf{v}} = RY(\theta_1) \otimes RY(\theta_2)$, while the angles θ_1 and θ_2 used in the rotational gates $RY(\theta_1)$ and $RY(\theta_2)$ are calculated as follows:

$$\theta_1 = 2 \arcsin(v_1) = 2 \arcsin\left(\frac{1}{\sqrt{2}}\right) = \frac{\pi}{2}, \quad \theta_2 = \theta_2.$$

$U_{\mathbf{v}}$ is then used to create the desired state $|\psi\rangle \in \mathcal{H}$ by applying $U_{\mathbf{v}}$ on the initial state of the quantum system $|\psi_0\rangle = |00\rangle \in \mathcal{H}$ in the following way:

$$\begin{aligned} U_{\mathbf{v}} |\psi_0\rangle &= (RY(\theta_1) \otimes RY(\theta_2)) |00\rangle = \left(RY\left(\frac{\pi}{2}\right) \otimes RY\left(\frac{\pi}{2}\right) \right) |00\rangle = \\ &= \left(\begin{pmatrix} \cos\left(\frac{\pi}{4}\right) & -\sin\left(\frac{\pi}{4}\right) \\ \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{4}\right) \end{pmatrix} \otimes \begin{pmatrix} \cos\left(\frac{\pi}{4}\right) & -\sin\left(\frac{\pi}{4}\right) \\ \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{4}\right) \end{pmatrix} \right) |00\rangle = \\ &= \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = |\psi\rangle. \end{aligned}$$

This resulting state $|\psi\rangle$ can be described as a superposition of the basis states:

$$|\psi\rangle = \alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle, \quad \text{where } \alpha = \beta = \gamma = \delta = \frac{1}{2}.$$

The probabilities $P_{|1\rangle}(|\psi_1\rangle)$ and $P_{|1\rangle}(|\psi_2\rangle)$ of the individual qubits $|\psi_1\rangle$ and $|\psi_2\rangle$ collapsing into the state $|1\rangle$ when measured are equal to⁶:

$$\begin{aligned} P_{|1\rangle}(|\psi_1\rangle) &= |\beta|^2 + |\delta|^2 = \frac{1}{2}, \\ P_{|1\rangle}(|\psi_2\rangle) &= |\gamma|^2 + |\delta|^2 = \frac{1}{2}. \end{aligned}$$

These probabilities correspond to the input vector probabilities $v_1^2 = \frac{1}{2}$ and $v_2^2 = \frac{1}{2}$, respectively, which implies that the input vector is correctly encoded. The exact implementation algorithm is explained in more detail in the next chapter (in the subsection 4.2.2). Here is a figure depicting a quantum circuit producing the desired state $|\psi\rangle$:

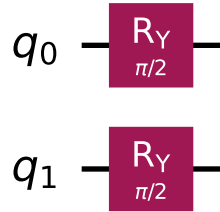


Figure 3.2 A visualization of a quantum circuit (in Qiskit) producing the state $\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \in \mathbb{C}^4$ using the angle encoding method. ▶

⁶As explained earlier, this thesis uses a convention where the rightmost digit in the ket notation corresponds to the first qubit.

3.3 Amplitude encoding method

The angle encoding method (3.1) is an improvement over the basis encoding method (3.2) in that it allows for more information to be encoded in the same number of qubits. However, there is one big issue that still persists—both of these techniques require relatively large numbers of qubits to encode input vectors, where they do not utilize the exponentiality of quantum computing. Although modern quantum computers offer many more qubits than in the past, it still takes a considerable number of qubits to encode vectors using the aforementioned encoding methods. For example, the *IBM Condor* is a quantum processor created by the company *IBM* that offers the highest number of qubits compared to any other quantum processor from *IBM* (as of May 2024) [19]. This quantum processor has exactly 1,121 qubits, meaning that it can theoretically encode at most only 1,121-dimensional input vectors. In practice, datasets may contain vectors of much greater lengths. In situations where the number of available qubits is low, or the input vectors are very long, encoding methods that use fewer qubits are very advantageous.

3.3.1 Introduction

The angle encoding method is not efficient in its quest to encode real vectors. It is possible to store the same amount of information in a lower number of qubits. In a nutshell, this is precisely the purpose of the encoding method that is presented in this section—to use fewer qubits than the number of qubits the encoding approaches introduced in the previous sections require.

When an input vector element $v_1 \in [-1, 1] \subset \mathbb{R}$ is encoded in the angle encoding method, the resulting one-qubit state $|\psi\rangle$ can be described as a following superposition of the standard basis states (the following equation is explained in the subsection Definition (3.2.1) of the angle encoding method):

$$|\psi\rangle = \sqrt{1 - v_1^2} |0\rangle + v_1 |1\rangle.$$

This means that both amplitudes ($\sqrt{1 - v_1^2}$ and v_1) of the superposition $|\psi\rangle$ contain information about only one input vector element v_1 . In other words, each input vector element requires two complex amplitudes for it to become encoded into a qubit. As a result, both amplitudes are wasted to encode just a single piece of information. So, if two amplitudes are available, then it means that theoretically, two different pieces of information can be encoded into one qubit. Thus, a question arises whether it is practically possible to somehow encode two different numbers into one qubit.

Assume a second number is given $v_2 \in \mathbb{R}$. Then it is possible to encode both numbers v_1 and v_2 into one qubit $|\psi\rangle$ in the form presented below:

$$|\psi\rangle = v_1 |0\rangle + v_2 |1\rangle,$$

as long as the following normalization condition is satisfied:

$$v_1^2 + v_2^2 = 1.$$

The normalization condition above must be satisfied because the probabilities of a quantum state collapsing into any state when measured must sum up to 1. So, given any two real numbers $v_1, v_2 \in \mathbb{R}$, they can be encoded into a qubit after they are normalized so that the sum of their squares equals 1. Therefore, if this approach were to be utilized in some encoding method, it would overcome the limitation of the angle encoding method using two qubit amplitudes to encode one input vector element. And such an encoding method exists—it is known as the **amplitude encoding method**. It may be clear already why the word *amplitude* is used in its name. That due to the fact that it uses *amplitudes* of quantum states to encode input vector elements. This way, the amplitude encoding method uses fewer qubits, as it is able to encode more information into each one.

Given $m \in \mathbb{N}$ qubits, a quantum state comprising of m qubits is a superposition of 2^m basis states, each having an amplitude, resulting in 2^m amplitudes being available to encode 2^m different numbers. Thus, given an input vector of length 2^m , this encoding technique produces a state whose state vector's length also equals 2^m , which in turn only requires $\log_2(2^m) = m$ qubits. This logarithmicity is a major benefit, as for the first time, the number of qubits required to encode an input vector is smaller than the number of elements of that input vector. This whole concept of using amplitudes to encode input vector elements is formulated more clearly below:

► Note 3.14 (Amplitude encoding method purpose). Given a real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n \in \mathbb{N}$, with the vector being normalized such that $\|\mathbf{v}\| = 1$, the purpose of the amplitude encoding method is to encode this vector into a state $|\psi\rangle \in \mathbb{C}^n$ of $\lceil \log_2(n) \rceil$ qubits, where the state $|\psi\rangle$ can be characterized as being the following superposition of the standard basis states:

$$|\psi\rangle = \sum_{i=1}^n v_i |e_i\rangle,$$

where $|e_i\rangle$ is the i -th basis state, with its amplitude being v_i (corresponding to the i -th input vector element). ◀

To achieve the desired effect outlined in the note above, a more sophisticated procedure must be used than the one employed by the angle encoding method. This procedure is elucidated in the following few subsections.

3.3.2 Detailed example

Let there be a normalized input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$ that is supposed to be encoded using the amplitude encoding method by creating a state $|\psi\rangle \in \mathbb{C}^m$. The input vector length n must be greater than or equal to 2 since at least one qubit must be used in

any quantum system, and that one qubit has two amplitudes. The resulting state is described as the following superposition of the standard basis states:

$$|\psi\rangle = \sum_{i=1}^n v_i |e_i\rangle,$$

where $|e_i\rangle$ is the i -th basis state, with its amplitude being v_i (corresponding to the i -th input vector element). It must be true that:

$$\| |\psi\rangle \|^2 = \sum_{i=1}^n v_i^2 = 1,$$

because the probabilities of measuring any possible state during a measurement must sum up to 1.

The resulting state $|\psi\rangle$ consists of m qubits. Each qubit of this resulting state must be in the correct state, defined by the input vector elements. This is first illustrated with an example on a short input vector:

► **Example 3.15.** Let $m = 2$. Then, the resulting state $|\psi\rangle$ is made up of two qubits, where the state vector of this state can be characterized as:

$$|\psi\rangle = v_1 |00\rangle + v_2 |01\rangle + v_3 |10\rangle + v_4 |11\rangle.$$

If the state $|\psi\rangle$ is created correctly, the probabilities of the first qubit of this state $|\psi\rangle$ collapsing into the states $|0\rangle$ and $|1\rangle$ when measured are:

$$P_{|*0\rangle}(|\psi\rangle) = v_1^2 + v_3^2, \quad P_{|*1\rangle}(|\psi\rangle) = v_2^2 + v_4^2.$$

The notations $|*0\rangle$ and $|*1\rangle$ are employed to denote the outcomes of a measurement performed on the quantum state $|\psi\rangle$. Specifically, $|*0\rangle$ indicates that the measurement results in the first qubit collapsing into the state $|0\rangle$, with the second qubit subsequently found in either the state $|0\rangle$ or $|1\rangle$. Correspondingly, $|*1\rangle$ signifies that the first qubit collapses into the state $|1\rangle$, independent of the resultant state of the second qubit. Based on the probabilities presented above, the first qubit of the state $|\psi\rangle$ can be characterized as follows:

$$\sqrt{v_1^2 + v_3^2} |0\rangle + \sqrt{v_2^2 + v_4^2} |1\rangle,$$

with the amplitudes being $\sqrt{v_1^2 + v_3^2}$ and $\sqrt{v_2^2 + v_4^2}$, respectively. So, having the first qubit initially in the state $|0\rangle$, it must be acted upon by some unitary operator to produce the state above. As is explained in detail in the previous chapter Angle encoding method (3.2), the rotational gate RY is capable of encoding a real number into a qubit. Because both amplitudes α_1 and β_1 are real numbers, the gate RY can also be used in this encoding method as well, in a similar fashion as in the angle encoding method. So, for the equations above to be true, the gate $RY(\theta_1)$ can be utilized in the following manner:

$$RY(\theta_1) |0\rangle = \cos\left(\frac{\theta_1}{2}\right) |0\rangle + \sin\left(\frac{\theta_1}{2}\right) |1\rangle.$$

This means that for the amplitudes $\cos \frac{\theta_1}{2}$ and $\sin \frac{\theta_1}{2}$, it must be true that:

$$\cos \left(\frac{\theta_1}{2} \right) = \sqrt{v_1^2 + v_3^2}, \quad \sin \left(\frac{\theta_1}{2} \right) = \sqrt{v_2^2 + v_4^2},$$

The angle θ_1 can be calculated using one of the equations above, either with the cosine function or with the sine function; in each case, the correct amplitudes must be chosen. So, solving for the angle θ_1 :

$$\begin{aligned} \cos \left(\frac{\theta_1}{2} \right) = \sqrt{v_1^2 + v_3^2} &\Leftrightarrow \theta_1 = 2 \arccos \left(\sqrt{v_1^2 + v_3^2} \right), \\ \sin \left(\frac{\theta_1}{2} \right) = \sqrt{v_2^2 + v_4^2} &\Leftrightarrow \theta_1 = 2 \arcsin \left(\sqrt{v_2^2 + v_4^2} \right). \end{aligned}$$

This way, the first qubit can be transformed from the initial state $|0\rangle$ to the desired state by using the gate RY in one of two possible ways:

1. $RY \left(2 \arccos \left(\sqrt{v_1^2 + v_3^2} \right) \right) |0\rangle$
2. $RY \left(2 \arcsin \left(\sqrt{v_2^2 + v_4^2} \right) \right) |0\rangle$

By choosing one of these approaches, the state of the first qubit is modified according to the input vector elements. Now that the first qubit is successfully rotated, the second one needs to be rotated, too. This is the point where this encoding method differs from the previous encoding methods. The second qubit cannot be rotated in the same way as the first one because the second qubit depends on the state of the first qubit. There are two possibilities for the first qubit of the resulting state $v_1 |00\rangle + v_2 |01\rangle + v_3 |10\rangle + v_4 |11\rangle = |\psi\rangle$:

- *Possibility 1: the basis states whose first qubit is in the state $|0\rangle$:*

This is true for the basis states $v_1 |00\rangle$ and $v_3 |10\rangle$. In this scenario:

- Out of these both basis states $v_1 |00\rangle$ and $v_3 |10\rangle$, the second qubit is in the state $|0\rangle$ within the basis state $v_1 |00\rangle$.
- Analogically, the second qubit is in the state $|1\rangle$ within the state $v_3 |10\rangle$.

So, for when the first qubit is in the state $|0\rangle$, the probabilities of the second qubit $|\psi_2\rangle$ collapsing into the states $|0\rangle$ and $|1\rangle$ are:

$$P_{|00\rangle}(|*0\rangle) = \frac{v_1^2}{v_1^2 + v_3^2}, \quad P_{|10\rangle}(|*0\rangle) = \frac{v_3^2}{v_1^2 + v_3^2}.$$

The denominator $v_1^2 + v_3^2$ ensures that $P_{|00\rangle}(|*0\rangle) + P_{|10\rangle}(|*0\rangle) = 1$ and suggests that the probabilities $P_{|00\rangle}(|*0\rangle)$ and $P_{|10\rangle}(|*0\rangle)$ are calculated for

the basis states whose first qubit is in the state $|0\rangle$. If the denominator is zero, the probabilities $P_{|00\rangle}(|*0\rangle)$ and $P_{|10\rangle}(|*0\rangle)$ are undefined, and thus the need to account for them in the encoding process is eliminated. In such a case, this scenario *Possibility 1* can be skipped.

To encode the probabilities $P_{|00\rangle}(|*0\rangle)$ and $P_{|10\rangle}(|*0\rangle)$ into the second qubit (currently its state is the initial state $|0\rangle$), a rotation must be applied to it. The angle θ_2 used in the rotation is calculated analogically as previously:

$$\cos\left(\frac{\theta_2}{2}\right) = \sqrt{\frac{v_1^2}{v_1^2 + v_3^2}} \Leftrightarrow \theta_2 = 2 \arccos\left(\sqrt{\frac{v_1^2}{v_1^2 + v_3^2}}\right),$$

$$\sin\left(\frac{\theta_2}{2}\right) = \sqrt{\frac{v_3^2}{v_1^2 + v_3^2}} \Leftrightarrow \theta_2 = 2 \arcsin\left(\sqrt{\frac{v_3^2}{v_1^2 + v_3^2}}\right).$$

However, now, the ordinary R_Y gate is insufficient because this case assumes that the first qubit is in the state $|0\rangle$. In some way, this condition needs to be translated into the circuit. This is where the controlled gates shine; they allow for conditional operations to be performed on qubits, thus entangling them. The gate R_Y is extended so that the first qubit acts as a *control qubit*. Gates that have a qubit acting as a control qubit are referred to as *controlled* gates. In this case, the first qubit is the controlled qubit, conditioned on being in the state $|0\rangle$. So, a controlled gate R_Y is applied to both qubits, where the first qubit acts as a control qubit and the second qubit has a rotation applied to it and is called a *target qubit*. Thus, the second qubit is rotated within the scope of the first qubit being in the state $|0\rangle$, resulting in these qubits becoming entangled.

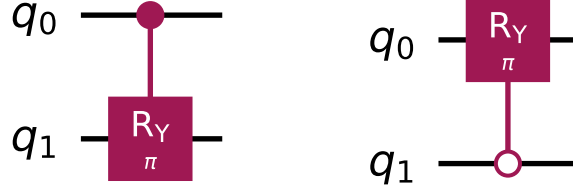
► Note 3.16 (Control qubit state). Ordinarily, a controlled gate is conditioned on its control qubit being in the state $|1\rangle$. In this scenario, the control qubit is assumed to be in the state $|0\rangle$. This issue can be easily fixed by applying the gate X to the control qubit before applying the controlled gate and applying the gate X to the control qubit again after the controlled gate is applied so that the effect of the first gate X is not translated into the final state but is reverted instead. ◀

► Note 3.17 (Controlled gate R_Y). The controlled gate R_Y has the following matrix representation (assuming that little-endian convention is used):

$$\begin{aligned} CRY(\theta) &= |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes RY(\theta) = \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \frac{\theta}{2} & 0 & -\sin \frac{\theta}{2} \\ 0 & 0 & 1 & 0 \\ 0 & \sin \frac{\theta}{2} & 0 & \cos \frac{\theta}{2} \end{pmatrix} \end{aligned}$$

◀

Two different possible setups of controlled gates R_Y are presented in the illustration below:



■ **Figure 3.3** A visualization of two two-qubit quantum circuits (created using the Qiskit library) with a controlled gate R_Y applied to each circuit. The gate on the left targets the qubit q_1 with the qubit q_0 serving as a control qubit whose control state is the state $|1\rangle$. The gate on the right targets the qubit q_0 with the qubit q_1 serving as a control qubit whose control state is in the state $|0\rangle$

- *Possibility 2: the basis states whose first qubit is in the state $|1\rangle$:*
This is an analogy to the scheme described in depth above. This time, the basis states $v_2 |01\rangle$ and $v_4 |11\rangle$ have their first qubit in the state $|1\rangle$, and:
 - Of these states $v_2 |01\rangle$ and $v_4 |11\rangle$, the second qubit is in the state $|0\rangle$ within the basis state $v_2 |01\rangle$.
 - The second qubit is in the state $|1\rangle$ within the state $v_4 |11\rangle$.

So, for the first qubit being in the state $|1\rangle$, the probabilities of the second qubit $|\psi_2\rangle$ collapsing into the states $|0\rangle$ and $|1\rangle$ are:

$$P_{|01\rangle}(|*1\rangle) = \frac{v_2^2}{v_2^2 + v_4^2}, \quad P_{|11\rangle}(|*1\rangle) = \frac{v_4^2}{v_2^2 + v_4^2}.$$

which in turn means the angle θ_3 used in the controlled rotation is calculated as:

$$\cos\left(\frac{\theta_3}{2}\right) = \sqrt{\frac{v_2^2}{v_2^2 + v_4^2}} \Leftrightarrow \theta_3 = 2 \arccos\left(\sqrt{\frac{v_2^2}{v_2^2 + v_4^2}}\right),$$

$$\sin\left(\frac{\theta_3}{2}\right) = \sqrt{\frac{v_4^2}{v_2^2 + v_4^2}} \Leftrightarrow \theta_3 = 2 \arcsin\left(\sqrt{\frac{v_4^2}{v_2^2 + v_4^2}}\right).$$

Again, if the denominator is zero, this process can be skipped. In this scenario, the controlled rotational gate R_Y is applied to both qubits such that the second qubit is the targeted qubit that has the angle θ_3 applied to it using the gate CRY , and the first qubit is the control qubit conditioned on the state $|1\rangle$.

In total, three gates R_Y are used to encode the four-dimensional input vector using the amplitude encoding method, with two of them being controlled and conditioned on the first qubit's state. For an input vector of size 2^m , $2^m - 1$ rotations are applied sequentially to the qubits. To conclude this example, three unitary operators representing the three gates are shown and applied to the qubits that are in the initial state $|0\rangle$. The final state $|\psi\rangle$ is thus created by sequentially applying three unitary operators $U_{RY}(\theta_1)$, $U_{CRY_0}(\theta_2)$, and $U_{CRY_1}(\theta_3)$ on the initial state $|\psi_0\rangle = |0\rangle \otimes |0\rangle$ in the following way: $U_{CRY_1}(\theta_3)U_{CRY_0}(\theta_2)U_{RY}(\theta_1)|\psi_0\rangle = |\psi\rangle$. First, the ordinary gate $RY(\theta_1)$ is applied (a part of the operator $U_{RY}(\theta_1)$), and then the controlled gates are applied (operators $U_{CRY_0}(\theta_2)$ and $U_{CRY_1}(\theta_3)$). The unitary operators are presented below:

$$\begin{aligned}
U_{RY}(\theta_1) &= I \otimes RY(\theta_1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} \cos \frac{\theta_1}{2} & -\sin \frac{\theta_1}{2} \\ \sin \frac{\theta_1}{2} & \cos \frac{\theta_1}{2} \end{pmatrix} = \\
&= \begin{pmatrix} \cos \frac{\theta_1}{2} & -\sin \frac{\theta_1}{2} & 0 & 0 \\ \sin \frac{\theta_1}{2} & \cos \frac{\theta_1}{2} & 0 & 0 \\ 0 & 0 & \cos \frac{\theta_1}{2} & -\sin \frac{\theta_1}{2} \\ 0 & 0 & \sin \frac{\theta_1}{2} & \cos \frac{\theta_1}{2} \end{pmatrix}, \\
U_{CRY_0}(\theta_2) &= (I \otimes X) \cdot CRY(\theta_2) \cdot (I \otimes X) = \\
&= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \frac{\theta_2}{2} & 0 & -\sin \frac{\theta_2}{2} \\ 0 & 0 & 1 & 0 \\ 0 & \sin \frac{\theta_2}{2} & 0 & \cos \frac{\theta_2}{2} \end{pmatrix} \cdot \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) = \\
&= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \frac{\theta_2}{2} & 0 & -\sin \frac{\theta_2}{2} \\ 0 & 0 & 1 & 0 \\ 0 & \sin \frac{\theta_2}{2} & 0 & \cos \frac{\theta_2}{2} \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \\
&= \begin{pmatrix} \cos \frac{\theta_2}{2} & 0 & -\sin \frac{\theta_2}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \sin \frac{\theta_2}{2} & 0 & \cos \frac{\theta_2}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \\
U_{CRY_1}(\theta_3) &= CRY(\theta_3),
\end{aligned}$$

The operator $U_{RY}(\theta_1)$ is just the gate $RY(\theta_1)$ expanded using the identity gate I (acting on the unaffected qubit) to the size 2×2 so that it can be applied to the whole 4-dimensional state vector. The same expansion also happens to the gates X used in the operator $U_{CRY_0}(\theta_2)$. Finally, the equation $|\psi\rangle = U_{CRY_1}(\theta_3)U_{CRY_0}(\theta_2)U_{RY}(\theta_1)|\psi_0\rangle$ can be written as follows:

$$|\psi\rangle = U_{CRY_1}(\theta_3)U_{CRY_0}(\theta_2)U_{RY}(\theta_1) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos \frac{\theta_1}{2} \cos \frac{\theta_2}{2} \\ \sin \frac{\theta_1}{2} \cos \frac{\theta_3}{2} \\ \cos \frac{\theta_1}{2} \sin \frac{\theta_2}{2} \\ \sin \frac{\theta_1}{2} \sin \frac{\theta_3}{2} \end{pmatrix}$$

When the angles are substituted for the corresponding numbers, the resulting state looks like this:

$$\begin{aligned} |\psi\rangle &= \begin{pmatrix} \cos \frac{\theta_1}{2} \cos \frac{\theta_2}{2} \\ \sin \frac{\theta_1}{2} \cos \frac{\theta_3}{2} \\ \cos \frac{\theta_1}{2} \sin \frac{\theta_2}{2} \\ \sin \frac{\theta_1}{2} \sin \frac{\theta_3}{2} \end{pmatrix} = \begin{pmatrix} \cos \frac{2 \arccos \sqrt{v_1^2+v_3^2}}{2} \cos \frac{2 \arccos \sqrt{\frac{v_1^2}{v_1^2+v_3^2}}}{2} \\ \sin \frac{2 \arcsin \sqrt{v_2^2+v_4^2}}{2} \cos \frac{2 \arccos \sqrt{\frac{v_2^2}{v_2^2+v_4^2}}}{2} \\ \cos \frac{2 \arccos \sqrt{v_1^2+v_3^2}}{2} \sin \frac{2 \arcsin \sqrt{\frac{v_3^2}{v_1^2+v_3^2}}}{2} \\ \sin \frac{2 \arcsin \sqrt{v_2^2+v_4^2}}{2} \sin \frac{2 \arcsin \sqrt{\frac{v_4^2}{v_2^2+v_4^2}}}{2} \end{pmatrix} = \\ &= \begin{pmatrix} \sqrt{v_1^2+v_3^2} \sqrt{\frac{v_1^2}{v_1^2+v_3^2}} \\ \sqrt{v_2^2+v_4^2} \sqrt{\frac{v_2^2}{v_2^2+v_4^2}} \\ \sqrt{v_1^2+v_3^2} \sqrt{\frac{v_3^2}{v_1^2+v_3^2}} \\ \sqrt{v_2^2+v_4^2} \sqrt{\frac{v_4^2}{v_2^2+v_4^2}} \end{pmatrix} = \begin{pmatrix} |v_1| \\ |v_2| \\ |v_3| \\ |v_4| \end{pmatrix} \end{aligned}$$

As seen in this outcome, the resulting vector is almost identical to the input vector $\mathbf{v} = (v_1 \ v_2 \ v_3 \ v_4)^T$, it only differs in the fact that it contains the absolute values of the input vector elements. The sign of each input vector element was lost during the encoding process. This fact was not initially mentioned for the sake of simplicity. However, the amplitude encoding technique addresses this concern. This issue can be easily fixed by slightly adjusting the angles used in the rotations. $k = 2^m - 1$ rotations are performed when encoding an input vector of length 2^m using this encoding method. When calculating the last $j = \frac{2^m}{2}$ angles used in the last $j = \frac{2^m}{2}$ gates (for m qubits, always k angles are calculated and k rotational gates are used, so if $k = 3$, $j = 2$, or if $k = 15$,

$j = 8$, etc.), the calculations need to be adjusted to account for the signs of the input vector elements in the following manner:

- Only the last j angles must be adjusted. Of those, when calculating the l -th angle θ (where $l \in [1, j] \subset \mathbb{N}$ is the index of the l -th angle belonging to the group of those last j angles), the following adjustments are needed:
 - If the $(2 \cdot l)$ -th element of the input vector is negative, the l -th angle θ is negated, so the adjusted angle θ_A is calculated as $\theta_A = -\theta$.
 - If the $(2 \cdot l - 1)$ -th element of the input vector is negative, the l -th angle θ needs to be adjusted to create θ_A in the following way: $\theta_A = 2\pi - \theta$.

Both adjustments may occur if both $(2 \cdot l)$ -th and $(2 \cdot l - 1)$ -th elements of the input vector are negative.

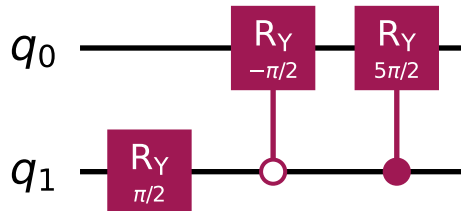
To show these adjustments in practice, this encoding method is demonstrated on a chosen four-dimensional real vector. Let the input vector \mathbf{v} be:

$$\mathbf{v} = \left(\frac{1}{2} \quad \frac{-1}{2} \quad \frac{-1}{2} \quad \frac{-1}{2} \right)^T.$$

Then, the result with the desired angle adjustments (to encode the signs of the input vector elements) would be (adjustments are highlighted in bold):

$$= \begin{pmatrix} \cos \frac{2 \arccos \sqrt{v_1^2 + v_3^2}}{2} & \cos \frac{-1 \cdot 2 \arccos \sqrt{\frac{v_1^2}{v_1^2 + v_3^2}}}{2} \\ \sin \frac{2 \arcsin \sqrt{v_2^2 + v_4^2}}{2} & \cos \frac{2\pi + 2 \arccos \sqrt{\frac{v_2^2}{v_2^2 + v_4^2}}}{2} \\ \cos \frac{2 \arccos \sqrt{v_1^2 + v_3^2}}{2} & \sin \frac{-1 \cdot 2 \arcsin \sqrt{\frac{v_3^2}{v_1^2 + v_3^2}}}{2} \\ \sin \frac{2 \arcsin \sqrt{v_2^2 + v_4^2}}{2} & \sin \frac{2\pi + 2 \arcsin \sqrt{\frac{v_4^2}{v_2^2 + v_4^2}}}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}$$

The adjustments made to the angles result in the input vector being correctly encoded. A quantum circuit that is created by encoding this input vector \mathbf{v} using this approach thoroughly explained in this subsection is shown below:



■ **Figure 3.4** A visualization of a quantum circuit created in Qiskit by encoding the input vector $\left(\frac{1}{2} \quad \frac{-1}{2} \quad \frac{-1}{2} \quad \frac{-1}{2} \right)^T \in \mathbb{R}^4$ using the amplitude encoding method.

This example is now finalized. Although only a four-dimensional vector was showcased, this example conscientiously explained the essence of the amplitude encoding method. The process is the same in its nature even when longer input vectors are used, and the algorithm that implements this process follows this example. This algorithm can be found in the subsection Amplitude encoding method (4.2.3) of the chapter Implementation (4) in the form of pseudocode. ▶

3.3.3 Generalization

The previous example only shows the process of encoding a four-dimensional input vector. However, that process can be generalized to showcase its ability to encode input vectors of larger lengths. The algorithm presented in Amplitude encoding method (4.2.3) is based on this generalization. To understand how this encoding method works in general, and to be able to encode longer than 4-dimensional input vectors, the following two processes need to be explained:

1. The general way to calculate the i -th angle θ needed in the i -th rotation.
2. The general way to characterize the i -th gate of the quantum circuit.

The following definition addresses the first point:

▶ **Definiton 3.18** (i -th angle calculation). *It is assumed that a normalized real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$ is supposed to be encoded using the amplitude encoding method. Then, $n - 1$ angles $\theta_1, \theta_2, \dots, \theta_{n-1}$ are calculated from the elements of \mathbf{v} . Let θ_i be the i -th calculated angle, where $i \in [1, n - 1] \subset \mathbb{N}$. Let:*

1. $o = 2^{m - \lceil \log_2(i) \rceil}$, $p = i + 1 - 2^{\lceil \log_2(i) \rceil}$, $q = 1 + (p - 1) \cdot 2 \cdot o$
2. $S(\mathbf{v}_{[a, b]})$, $1 \leq a \leq b \leq n$ denote the sum of the squared input vector elements, summed from the a -th element to the b -th element:

$$S(\mathbf{v}_{[a, b]}) = \sum_{i=a}^b v_i^2$$

3. $R(\mathbf{v}_{[a, b]}, \mathbf{v}_{[c, d]})$ represent the following number:

$$R(\mathbf{v}_{[a, b]}, \mathbf{v}_{[c, d]}) = \begin{cases} \frac{S(\mathbf{v}_{[a, b]})}{S(\mathbf{v}_{[a, b]}) + S(\mathbf{v}_{[c, d]})} & \text{if } S(\mathbf{v}_{[a, b]}) + S(\mathbf{v}_{[c, d]}) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Then, the angle θ_i is calculated using the following formula:

$$\theta_i = \begin{cases} 2 \arccos \left(\sqrt{S(\mathbf{v}_{[1, 2^{m-1}]})} \right) & \text{if } i = 1, \\ 2 \arccos \left(\sqrt{R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]})} \right) & \text{if } i > 1. \end{cases}$$
▶

► **Example 3.19.** An example calculation is conducted using the above definition for a given 8-dimensional input vector $\mathbf{v} = (v_1 \ v_2 \ v_3 \ v_4 \ v_5 \ v_6 \ v_7 \ v_8)^T \in \mathbb{R}^8$, $\|\mathbf{v}\| = 1$. Let $m = \log_2(8) = 3$ denote the number of qubits required to encode this input vector using the amplitude encoding method. Then, the definition can be utilized to create the following expressions (with these expressions, the 7 required angles can be calculated):

1. For the first angle θ_1 , where $i = 1$:

$$S(\mathbf{v}_{[1, 2^{m-1}]}) = S(\mathbf{v}_{[1, 2^{3-1}]}) = S(\mathbf{v}_{[1,4]}) = v_1^2 + v_2^2 + v_3^2 + v_4^2$$

2. For the second angle θ_2 , where $i = 2$:

$$\begin{aligned} o &= 2^{m - \lfloor \log_2(i) + 1 \rfloor} = 2^{3 - \lfloor \log_2(2) + 1 \rfloor} = 2^{3-2} = 2 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 2 + 1 - 2^{\lfloor \log_2(2) \rfloor} = 3 - 2^1 = 1 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (1 - 1) \cdot 2 \cdot 2 = 1 + 0 \cdot 2 \cdot 2 = 1 \end{aligned}$$

$$\begin{aligned} R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[1,2]}, \mathbf{v}_{[3,4]}) = \\ &= \frac{S(\mathbf{v}_{[1,2]})}{S(\mathbf{v}_{[1,2]}) + S(\mathbf{v}_{[3,4]})} = \frac{v_1^2 + v_2^2}{v_1^2 + v_2^2 + v_3^2 + v_4^2} \end{aligned}$$

3. For the third angle θ_3 , where $i = 3$:

$$\begin{aligned} o &= 2^{m - \lfloor \log_2(i) + 1 \rfloor} = 2^{3 - \lfloor \log_2(3) + 1 \rfloor} = 2^{3-2} = 2 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 3 + 1 - 2^{\lfloor \log_2(3) \rfloor} = 4 - 2^1 = 2 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (2 - 1) \cdot 2 \cdot 2 = 1 + 1 \cdot 2 \cdot 2 = 5 \end{aligned}$$

$$\begin{aligned} R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[5,6]}, \mathbf{v}_{[7,8]}) = \\ &= \frac{S(\mathbf{v}_{[5,6]})}{S(\mathbf{v}_{[5,6]}) + S(\mathbf{v}_{[7,8]})} = \frac{v_5^2 + v_6^2}{v_5^2 + v_6^2 + v_7^2 + v_8^2} \end{aligned}$$

4. For the fourth angle θ_4 , where $i = 4$:

$$\begin{aligned} o &= 2^{m - \lfloor \log_2(i) + 1 \rfloor} = 2^{3 - \lfloor \log_2(4) + 1 \rfloor} = 2^{3-3} = 1 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 4 + 1 - 2^{\lfloor \log_2(4) \rfloor} = 5 - 2^2 = 1 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (1 - 1) \cdot 2 \cdot 1 = 1 + 0 \cdot 2 \cdot 1 = 1 \end{aligned}$$

$$\begin{aligned} R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[1,1]}, \mathbf{v}_{[2,2]}) = \\ &= \frac{S(\mathbf{v}_{[1,1]})}{S(\mathbf{v}_{[1,1]}) + S(\mathbf{v}_{[2,2]})} = \frac{v_1^2}{v_1^2 + v_2^2} \end{aligned}$$

5. For the fifth angle θ_5 , where $i = 5$:

$$\begin{aligned} o &= 2^{m-\lfloor \log_2(i)+1 \rfloor} = 2^{3-\lfloor \log_2(5)+1 \rfloor} = 2^{3-3} = 1 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 5 + 1 - 2^{\lfloor \log_2(5) \rfloor} = 6 - 2^2 = 2 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (2 - 1) \cdot 2 \cdot 1 = 1 + 1 \cdot 2 \cdot 1 = 3 \\ R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[3,3]}, \mathbf{v}_{[4,4]}) = \\ &= \frac{S(\mathbf{v}_{[3,3]})}{S(\mathbf{v}_{[3,3]}) + S(\mathbf{v}_{[4,4]})} = \frac{v_3^2}{v_3^2 + v_4^2} \end{aligned}$$

6. For the sixth angle θ_6 , where $i = 6$:

$$\begin{aligned} o &= 2^{m-\lfloor \log_2(i)+1 \rfloor} = 2^{3-\lfloor \log_2(6)+1 \rfloor} = 2^{3-3} = 1 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 6 + 1 - 2^{\lfloor \log_2(6) \rfloor} = 7 - 2^2 = 3 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (3 - 1) \cdot 2 \cdot 1 = 1 + 2 \cdot 2 \cdot 1 = 5 \\ R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[5,5]}, \mathbf{v}_{[6,6]}) = \\ &= \frac{S(\mathbf{v}_{[5,5]})}{S(\mathbf{v}_{[5,5]}) + S(\mathbf{v}_{[6,6]})} = \frac{v_5^2}{v_5^2 + v_6^2} \end{aligned}$$

7. For the seventh angle θ_7 , where $i = 7$:

$$\begin{aligned} o &= 2^{m-\lfloor \log_2(i)+1 \rfloor} = 2^{3-\lfloor \log_2(7)+1 \rfloor} = 2^{3-3} = 1 \\ p &= i + 1 - 2^{\lfloor \log_2(i) \rfloor} = 7 + 1 - 2^{\lfloor \log_2(7) \rfloor} = 8 - 2^2 = 4 \\ q &= 1 + (p - 1) \cdot 2 \cdot o = 1 + (4 - 1) \cdot 2 \cdot 1 = 1 + 3 \cdot 2 \cdot 1 = 7 \\ R(\mathbf{v}_{[q, q+o-1]}, \mathbf{v}_{[q+o, q+o+o-1]}) &= R(\mathbf{v}_{[7,7]}, \mathbf{v}_{[8,8]}) = \\ &= \frac{S(\mathbf{v}_{[7,7]})}{S(\mathbf{v}_{[7,7]}) + S(\mathbf{v}_{[8,8]})} = \frac{v_7^2}{v_7^2 + v_8^2} \end{aligned}$$

Each of these expressions must then be used in the arccos function along with the square root to calculate the corresponding angle. The angle can also be optionally adjusted to account for the correct signs of the input vector elements, as explained in the example previously. ◀

► Note 3.20. This definition was created by directly translating the implemented algorithm that is introduced in the form of pseudocode in the subsection Amplitude encoding method (4.2.3) of the chapter Implementation (4). So, the pseudocode in that subsection provides an understanding of how this definition is used in practice. Another crucial fact worth noting is that this definition utilizes the same endianness that is used in the implementation. ◀

► Remark 3.21. This definition works for input vectors of sizes $n = 2^m$, $m \in \mathbb{N}$. This is a limitation, as there may be input vectors with lengths other than n . However, this limitation can be easily resolved. If the length of the input vector is $l \in \mathbb{N}$, the input vector can be padded to the nearest power of two that is larger than l by adding enough zeros at the end. ◀

Now, it is time to explore the second point—explaining which exact gates are used in the case of an arbitrarily sized input vector. In the lengthy example presented previously, it was sufficient to use three gates, where two of them were controlled gates CRY . For larger vectors, an ordinary CRY would not be sufficient because more qubits might need to act as control states. The gate CRY can be extended further to condition more than one qubit, resulting in a multi-controlled gate RY that has multiple qubits acting as control qubits.

► Note 3.22. The extension of the controlled RY gate to a multi-controlled version, denoted as $C^n RY(\theta)$, involves the inclusion of additional control qubits. This gate applies the $RY(\theta)$ rotation to the target qubit conditioned on the state of n control qubits being in the state $|1\rangle$. If any control qubit is in the state $|0\rangle$, the identity operation I is applied, leaving the target state unchanged. As was explained earlier (Control qubit state, 3.16), if there is a need for different control states than $|1\rangle$, the gates X need to be applied to the corresponding control qubits. The matrix form for the n -controlled RY gate is of size $2^{n+1} \times 2^{n+1}$, acting on the Hilbert space $\mathcal{H} = \mathbb{C}^{2^{n+1}}$. Its representation utilizes Kronecker products to appropriately enlarge the matrix $RY(\theta)$, ensuring that the rotation is applied to the target qubit only under the specified conditional states. Practical implementations may require decomposition into simpler gates to facilitate execution on real quantum computing devices. ◀

► **Definiton 3.23** (*i -th gate properties*). Let \mathbf{v} be a normalized input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$. For this vector to be encoded using the amplitude encoding method, m qubits are used and $n - 1$ angles are calculated, and therefore, $n - 1$ rotational gates need to be applied, denoted as $C^0 RY(\theta_1)$, $C^1 RY(\theta_2)$, \dots , $C^{n-2} RY(\theta_{n-1})$. Then, the i -th gate $C^{i-1} RY(\theta_i)$, $i \in [1, n - 1] \subset \mathbb{N}$ is fully characterized by having the following properties:

- It targets the $(m - \lceil \log_2(i) \rceil)$ -th qubit, let this qubit be the j -th qubit.
- It has $k = \lfloor \log_2(i) \rfloor$ control qubits.
- If $k > 0$, the control qubits range from the $(j + 1)$ -th qubit to the m -th qubit.
- The control state $|c_p\rangle \in \{|0\rangle, |1\rangle\}$ of the p -th control qubit ($p \in \hat{k}$, the p -th control qubit is the $(m + 1 - p)$ -th qubit overall) is calculated as follows:

$$c_p = \left\lfloor \frac{i \bmod 2^p}{2^{p-1}} \right\rfloor.$$

◀

The algorithm introduced in the subsection Amplitude encoding method (4.2.3) of the chapter Implementation (4) that represents the amplitude encoding method was implemented based on this definition. The pseudocode presented in that subsection illustrates how this definition is implemented in practice.

3.3.4 Properties

Now, with both definitions above, arbitrarily sized input vectors (arbitrary lengths of $n = 2^m$, $m \in \mathbb{N}$) can be encoded using this encoding method. There are certain aspects of this encoding method that are worth mentioning. They are outlined below.

► Remark 3.24. Let $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$ be an input vector that is encoded using the amplitude encoding method, resulting in a state $|\psi\rangle$ described within a 2^m -dimensional Hilbert space $\mathcal{H} = \mathbb{C}^{2^m}$.

- In comparison with the basis and angle encoding methods, the amplitude encoding method is vastly superior in terms of qubit demand. For example, if given a quantum system with a thousand available qubits, theoretically, it is possible to encode an input vector of length 1000 in both basis and angle encoding methods, but an input vector of length 2^{1000} in the amplitude encoding method⁷. This difference in exponentiality effectively renders the basis and angle encoding methods insignificant in terms of how lengthy input vectors they are able to encode.
- It is possible to optimize this encoding method in terms of the number of gates it uses. If a rotational gate has no effect on the qubit it targets, there is no point in using such a gate. For instance, this can occur when the angle θ used in a rotation is equal to 0. Under these circumstances, the application of the gate that uses this angle can be omitted.
- This encoding method typically works for input vectors of lengths equal to some power of two, specifically 2^m , $m \in \mathbb{N}$. However, it is possible to overcome this limitation. Given an input vector of a truly arbitrary length $n \in \mathbb{N}$ (where n is not a power of two), this vector can be encoded using this encoding method if the following procedure is performed:
 1. First, the input vector is enlarged so that its length becomes the nearest power of two larger than n . This is done by appending zeros to the vector's end.
 2. Then, such an enlarged vector can be encoded in the ordinary way.
 3. After it becomes encoded, the resulting state vector can be trimmed so that it only contains the first n elements. This state vector then fully corresponds to the original input vector.

⁷For better comprehension, the number 2^{1000} has 302 digits.

The above process can be performed analogously when zeros are appended to the beginning of the vector, with the state vector then being trimmed from the beginning so that it only contains the last n elements.

- An obvious feature of this encoding method is that it produces states with entangled qubits due to the fact that controlled gates are used.
- The gates that are applied to produce $|\psi\rangle$ closely resemble certain aspects of binary trees. These are summarized below:
 - The last qubit has just one rotation applied to it. The second to last qubit has two rotations applied to it. The subsequent qubit has four rotations applied to it, and so on, until the first qubit is reached and it has $\frac{n}{2}$ rotations applied to it, which is more than half of all rotations applied.
 - The control states used in the controlled rotational gates can be calculated by taking the binary representation of the number denoting the order of the gates that target a specific qubit (with some minor adjustments).

When the angles are calculated for the rotations, they also participate in the process that resembles certain binary tree aspects:


1. The first angle resembles the first binary level with half of the input vector elements used when calculating it.
 2. The second and third angles resemble the second binary tree level, with each having a fourth of the input vector elements in the nominator in their formula that is used to calculate them (the formula can be seen in the definition i -th angle calculation, 3.18).
 3. The fourth to seventh angles (four angles in total) resemble the third binary tree level, with each angle having an eighth of the input vector elements in the nominator of its formula.
 4. Assuming the total angle count is a , a total of $\frac{a+1}{2}$ angles comprise the last binary tree level, with each angle having only one input vector element in the nominator of its formula.
- As mentioned, the major benefit of this encoding method is that it requires a logarithmic number of qubits when comparing the qubit count to the input vector's length. However, there is a trade-off. The number of rotational gates is exponential to the qubit count. For m qubits, $2^m - 1$ rotational gates are utilized. In real quantum computers, it is best to keep both qubit count and gate count requirements at a minimum. This is because the number of available qubits is limited, and each gate in a quantum circuit has the potential to introduce errors. So, the more gates used in a circuit, the more error-prone the encoding process becomes. It is crucial to adhere

to a delicate balance between these two metrics. As seen in the chapter Testing (5), the number of gates used in a circuit has a significant impact on the quality of the resulting state.

To further elucidate, the logarithmic scaling of qubits with respect to the input size represents a substantial advantage in terms of resource efficiency. This characteristic makes this encoding technique particularly appealing for applications involving large datasets, where a linear or polynomial number of qubits would be impractical [28]. Nonetheless, the exponential increase in the number of rotational gates introduces significant complexity. Each additional gate increases the likelihood of decoherence and operational errors, which can degrade the fidelity of the quantum state [31].

Moreover, in practical implementations, the limitations of current quantum hardware must be considered. Quantum error correction and fault-tolerant quantum computing are active areas of development aimed at mitigating these errors [19], but some of the solutions also entail additional qubit overhead [32]. Therefore, trying to optimize quantum circuits to minimize their gate usage without compromising their computational power is essential.

In the broader context of quantum information theory, the trade-off between qubit and gate counts is a pivotal consideration. It influences algorithm designs, hardware development, and overall computational strategies [33]. As quantum technology progresses, achieving an optimal balance will be fundamental to leveraging the full potential of quantum computing. This entails not only reducing the physical gate count but also innovating in error mitigation techniques and circuit simplification methods [19, 34].

Additionally, it is worth noting that different quantum algorithms (that process encoded data) may exhibit varying sensitivities to the gate and qubit counts. Some algorithms may tolerate higher gate counts if they offer significant computational advantages, while others might prioritize minimal gate usage to preserve quantum coherence [15]. Therefore, the specific requirements and constraints of the application domain must be carefully evaluated when choosing the optimal encoding method. This is why all three encoding methods discussed in this thesis so far may be utilized, even though the amplitude encoding method might be preferred in many cases due to its qubit count advantage. However, there may exist some instances when the amplitude encoding method cannot be utilized due to its exponential gate count, whereas the angle and basis encoding methods cannot be utilized due to their limitations. For these cases, a different encoding method might be useful, but one such that it retains many benefits of the amplitude encoding method. This serves as a rationale to introduce the last encoding method discussed in this thesis. This last encoding method originates from the amplitude encoding approach but modifies some of its aspects. This final encoding method is discussed in the next subsection. 

3.3.5 Divide-and-conquer encoding method

This subsection introduces the last encoding method discussed in this thesis. This encoding method is a variation of the amplitude encoding method. It is referred to as the amplitude encoding using a *divide-and-conquer principle*, or concisely called the **divide-and-conquer encoding method** throughout this thesis. The name of this encoding approach stems from the fact that it utilized a divide-and-conquer strategy during the state preparation process. The rationale for presenting this encoding method is mentioned in the last part of the previous subsection (3.3.4). It was mentioned that one of the most significant downsides to the amplitude encoding method is the number of gates it uses to encode input vectors. The divide-and-conquer strategy aims to tackle this issue by encoding the same angles in a different way.

3.3.5.1 Introduction and core ideas

This encoding method starts in the same way as the classical amplitude encoding method—by calculating the required angles used in rotational gates based on given input vector elements. This means that if an input vector \mathbf{v} is given such that $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$, the first step is to calculate $n - 1$ angles, each using the definition i -th gate properties (3.23) presented in the subsection Generalization (3.3.3) of this section. However, from now on, the two encoding methods start to differ. The main disparity lies in the way they encode the calculated angles. While the amplitude encoding technique utilizes multi-controlled rotational gates, this encoding approach works differently. It encodes the angles in exactly the same way as the angle encoding method does. So, having $n - 1$ angles, the angle encoding method would encode each one in a separate qubit using the ordinary rotational gates RY , thus using $n - 1$ qubits. The divide-and-conquer encoding method does the same thing. It uses $n - 1$ qubits, and each one has the gate RY applied to it with the corresponding angle. So, having angles $\theta_1, \theta_2, \dots, \theta_{n-1}$, $n - 1$ rotational gates $RY(\theta_i)$ are applied, where the i -th qubit ($i \in [1, n - 1] \subset \mathbb{N}$) is affected by the gate $RY(\theta_i)$. This is in sharp contrast with the multi-rotational gates used in the angle encoding method. However, this is not the end of the encoding process. If this were the end, this encoding approach would not prepare the desired state. It is vital to explain what state this encoding approach is supposed to create. This is explained in the following note:

► **Note 3.25** (Divide-and-conquer encoding method purpose). Given a real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, with the vector being normalized such that $\|\mathbf{v}\| = 1$, the divide-and-conquer encoding method encodes this input vector into a state $|\psi\rangle \in \mathbb{C}^{2^{n-1}}$ of $n - 1$ qubits. Let M be a set of $\lceil \log_2(n) \rceil$ qubits such that $\forall i \in \{2^k \mid k \in \mathbb{Z}, 0 \leq k < \log_2(n - 1)\}$ is the i -th qubit in the set M . It is assumed that all qubits in M are measured after the vector \mathbf{v} is encoded using this encoding method, and let \mathbf{p}_m be a vector of probabilities of measuring each possible state. Then, the purpose

of this encoding method is for the vector \mathbf{p}_m to be equal to the following superposition of the standard basis states:

$$\mathbf{p}_m = \sum_{j=1}^n v_j^2 |e_j\rangle,$$

where $|e_j\rangle$ is the j -th basis state of the standard basis, with each standard basis vector having a length of n . The amplitudes of these basis states correspond to the squared input vector elements. ◀

There is one crucial fact that is obvious from the note above—not all qubits comprise the output state of this encoding process. This is a clear distinction between this encoding method and all the previous ones. In all previous encoding methods, all qubits participated in the encoding process in a way that they contributed to the desired state vector. Here, only some qubits are directly involved in the representation of the desired state. These distinctive qubit groups have their own terms that describe them:

▶ Note 3.26 (Data qubits and auxiliary qubits). Qubits in quantum circuits are typically classified into two main categories:

- The qubits that are involved in the direct representation of the desired output are referred to as **data qubits**.
- The qubits that are not directly involved in the characterization of the desired output state are called **auxiliary qubits** or **ancillary qubits**. ◀

In all previous data encoding methods, all qubits comprise the desired state, so all are data qubits. However, as clarified in the note above, in this encoding method, only some qubits are data qubits (those that are involved in describing the probability vector \mathbf{p}_m). All other qubits are auxiliary (qubits that are not measured to obtain \mathbf{p}_m). Specifically, each i -th qubit such that i is a power of two is a data qubit, while each other qubit is an ancilla qubit.

The note also highlights the fact that in this encoding approach, more qubits are used in the encoding process than in the case of the amplitude encoding method. Moreover, the number of qubits used to encode an input vector is the same as the number of elements in that input vector, minus one. The rationale behind this encoding method was supposed to be reducing the number of required gates (which holds true, as explained later), but in the process, the number of required qubits rose up to be again linearly aligned to the input vector, as is the case in the basis and encoding methods. If all qubits were to be measured, not only data qubits, the length of the resulting state would have been exponential in comparison with the input vector's length. To preserve the purpose of the input vector acting as the desired state⁸, only a

⁸Which is true in the amplitude encoding method, where the input vector is identical to the desired output vector. This divide-and-conquer approach acts as an alternative version of the amplitude encoding method, so some of the properties of the amplitude encoding method must be retained in this encoding technique.

subset of qubits must act as data qubits; otherwise, the length of the resulting state would have exceeded the length of the input vector. So, to summarize all these findings:

- Only a subset of qubits is used to create the resulting state that corresponds to the squared input vector elements.
- The angles that contain information about the input vector are uniformly distributed across all qubits.

These facts mean that for the angles to participate in the resulting state, they must somehow be represented in the data qubits. So, each angle must, in the end, in some way, connect to one or more data qubits. If two qubits are given, and the state of one of the qubits must be represented in the state of the other qubit, there is only one way to achieve such a goal—through the entanglement. So, after all the angles are encoded using the gates RY , the qubits have to become entangled in a way that when the correct subset of qubits is measured, the probabilities of measuring any possible state correspond to the squared input vector elements.

3.3.5.2 Divide-and-conquer strategy and example

This is where the main point of this encoding method comes in. The name of this encoding method suggests the principle that is used to combine the qubits together. The term “divide-and-conquer principle” generally characterizes the act of dividing a more complex task into subtasks, where each subtask can then be solved more easily when only focusing on solving it, rather than the whole initial problem; then, the solutions to the subproblems can be combined together to form a solution to the whole initial problem. This “divide-and-conquer principle” is utilized in this encoding method after all the angles $\theta_1, \theta_2, \dots, \theta_{n-1}$ are already encoded using the rotational gates. The complex problem that needs to be solved using the divide-and-conquer principle is to entangle all $n - 1$ qubits in a way that only a subset of these $n - 1$ qubits, specifically $\lceil \log_2(n) \rceil$ qubits comprise the desired state. To explain how the divide-and-conquer strategy is used to combine the qubits, a look is first taken back at how the amplitude encoding method works.

It is mentioned in the previous subsection Properties (3.3.4) that the process of encoding input vectors using the amplitude encoding method resembles certain aspects of binary trees. If representing rotational gates as tree nodes, the first gate (the gate RY) is the root of the tree, the second two gates (the gates CRY) are two child nodes of the root node, and they comprise the second binary tree level. The third level then contains four gates C^2RY , the fourth level contains $2^{level-1} = 2^{4-1} = 2^3 = 8 =$ gates $C^{level-1}RY = C^3RY$, and so on. All those gates are applied sequentially, meaning the binary tree uses a breadth-first approach involving adding nodes (gates) level by level, ensuring that all nodes at a given level are added before proceeding to the next depth

level. The angle calculation process also mimics a binary tree by computing the angles in a binary tree-like manner in a top-to-bottom approach, halving the input vector at each level ⁹.

The divide-and-conquer encoding method also adheres to this binarity principle, but with one distinct difference—while the amplitude encoding process utilizes a top-to-bottom tree approach (first encoding the initially calculated angles that represent big chunks of the input vector, and slowly descending by encoding angles that represent smaller and smaller chunks of the input vector until only pairs of input vector elements are represented in the last angles), this encoding method uses a bottom-to-top tree approach (where the first angles that are processed by entangling their respective qubits are the last calculated angles, meaning ones that represent the pairs of the input vector elements; then ascending the tree structure by entangling qubits that represent input vector chunks of larger sizes, until the largest chunk is reached, which is the first angle, and the first angle represents half of the input vector elements).

Now that this introductory explanation is complete, the following two major concepts need to be clarified to precisely characterize the exact encoding process this divide-and-conquer principle employs:

- What kinds of gates are used to entangle the qubits.
- How are the gates used in the circuit to encode a given input vector.

The first point needs to be addressed initially. It was already mentioned that this encoding method first encodes all the calculated angles using the gates RY . There are no more angles left to encode and no other angles are calculated to be used in any rotational gates. The only thing left to do is to appropriately combine the qubits to reach the desired state. The gate CX can be used to achieve this goal, since it entangles qubits. This encoding method does indeed use the gates CX , however, they are just a part of a larger gate that is used—the controlled $SWAP$ gate. So, the answer to the first point is simple—the controlled $SWAP$ gates are used. This gate is introduced in the note below:

► Note 3.27 (Gate $CSWAP$). The controlled $SWAP$ gate (denoted as $CSWAP$) affects three qubits and it has the following properties:

- Two qubits serve as target qubits.
- One qubit serves as the control qubit.
- This gate acts as the classical gate $SWAP$ on the target qubits by swapping their states, but the swapping is conditioned on the state of the control qubit.
- The swap of the targeted qubits occurs only if the control qubit is in the state $|1\rangle$.

⁹This whole explanation is simplified, only intended to convey the notion that the encoding process follows certain binarity principles.

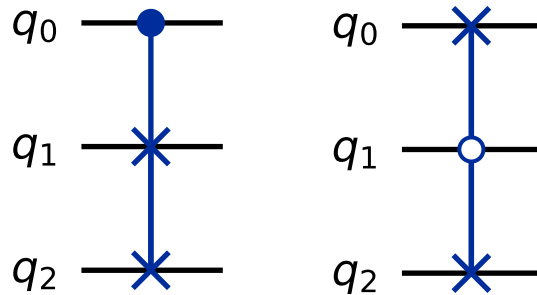
- As with every controlled gate, if a control qubits needs to be conditioned for the state $|0\rangle$, the gates X are applied to that control qubit before and after the main controlled gate.
- This gate is also known as the *Fredkin gate*.

This gate has the following matrix representation:

$$I \otimes I \otimes |0\rangle\langle 0| + \text{SWAP} \otimes |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

assuming the first qubit is the control qubit and the last two qubits are the target qubits.

Two different possible setups of gates *CSWAP* are shown in the illustration below:

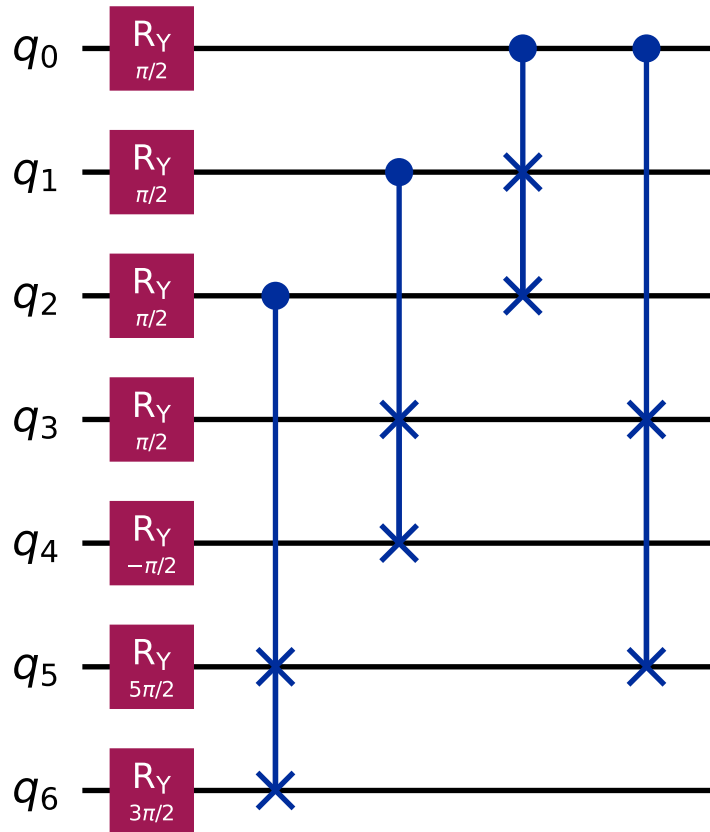


■ **Figure 3.5** A visualization of two three-qubit quantum circuits (created using the Qiskit library) with a controlled gate *SWAP* applied to each circuit. The gate on the left targets the qubits q_1 and q_2 with the qubit q_0 serving as a control qubit whose control state is the state $|1\rangle$. The gate on the right targets the qubits q_0 and q_2 with the qubit q_1 serving as a control qubit whose control state is in the state $|0\rangle$ ◀

The question that now arises is how these gates *CSWAP* are used to prepare the desired state. This can be explained and demonstrated in a small example below:

▶ **Example 3.28** (Divide-and-conquer encoding method usage example). Let $\mathbf{v} = \left(\frac{1}{\sqrt{8}} \frac{1}{\sqrt{8}} \frac{1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{1}{\sqrt{8}}\right)^T \in \mathbb{R}^8$, $\|\mathbf{v}\| = 1$, $n = 8$ be a given normalized 8-dimensional input vector that is supposed to be encoded using the divide-and-conquer encoding method. Then, the quantum circuit created by

encoding this input vector using this method has $k = n - 1 = 7$ qubits in total, $m = \log(n) = 3$ data qubits (the first, second, and fourth qubits are data qubits, because each i -th qubit is a data qubits such that i is a positive power of two smaller than n), and $k - m = 4$ ancillary qubits. Then, the divide-and-conquer encoding method encodes \mathbf{v} by creating the following circuit:



■ **Figure 3.6** A diagram of a Qiskit quantum circuit created by encoding the input vector $\left(\frac{1}{\sqrt{8}} \frac{1}{\sqrt{8}} \frac{1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{-1}{\sqrt{8}} \frac{1}{\sqrt{8}}\right)^T \in \mathbb{R}^8$ using the divide-and-conquer encoding method.

As clearly visible, the seven gates RY comprise the first row of gates, each parallelly applied to a different qubit. The angles used in the rotations are calculated using the definition i -th gate properties (3.23) presented in the subsection Generalization (3.3.3) of this section. Then, the $CSWAP$ gates are applied to the qubits in the following binary tree-like manner:

- The first qubit (q_0) denotes the root of the tree, characterized by the first angle that is calculated using the first four input vector elements.

- The following two qubits comprise the second tree level, where each angle is calculated using two input vector elements¹⁰. The qubit q_1 is the right child node of the root node, while the qubit q_2 is the left child node.
- The last level contains four qubits (the qubits q_3, q_4, q_5, q_6 serving as leaves of the tree), with four angles encoded (to calculate each of these angles, one input vector element is used). The first two of these qubits have the qubit q_1 as their parent node, the last two qubits have the qubit q_2 as their parent node.
- The divide-and-conquer principle combines the qubits from the bottom-up, so, initially, the last two qubits q_5 and q_6 are combined with their parent qubit q_2 and the qubits q_3 and q_4 are combined with their parent qubit q_1 .
- The second-level qubits are combined with the root of the tree.
- The root node is combined with the left children of the second-level qubits.

When the first, second, and fourth qubits (q_0, q_1, q_3) are measured, the resulting vector containing the probabilities of measuring every possible state \mathbf{p}_m equals:

$$\mathbf{p}_m = \sum_{j=1}^8 \frac{1}{8} |e_j\rangle,$$

where $|e_j\rangle$ is the j -th basis state of the standard basis, with each standard basis vector having a length of $n = 8$. This, in turn, fully corresponds to the squared input vector elements, indicating a correct functionality of the encoding method, resulting in the preparation of the desired quantum state:

$$\mathbf{p}_m = \sum_{j=1}^8 \frac{1}{8} |e_j\rangle = \sum_{j=1}^8 v_i^2 |e_j\rangle,$$

where v_i is the j -th input vector element.

This example illustrates the inner workings of this encoding method when given a 8-dimensional input vector. However, the same divide-and-conquer principle is used to encode longer vectors. ◀

3.3.5.3 Algorithm

One practical constraint of this divide-and-conquer principle is that it only works for input vectors that contain four or more elements, where the element count equals some positive power of two. Four elements are required because the gates *CSWAP* act on three qubits, so the quantum circuit needs to have at least three qubits to properly encode the input vector. And three qubits

¹⁰Not counting the parent input vector elements used in the denominator of the expression $\frac{S(\mathbf{v}_{[a, b]})}{S(\mathbf{v}_{[a, b]} + S(\mathbf{v}_{[c, d]})}$, the input vector elements used in the nominator are counted only.

are present in the case when the input vector is four-dimensional. It might be beneficial to generalize some of the properties of this encoding method to summarize how it works in some of the aspects:

► Note 3.29 (Divide-and-conquer encoding method quantum circuit properties). Given a real input vector $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in [2, \infty) \subset \mathbb{N}$, with the vector being normalized such that $\|\mathbf{v}\| = 1$, the divide-and-conquer encoding method encodes this input vector by creating a quantum circuit, where some of the circuit properties are characterized as follows:

- There are a total of $n - 1$ qubits in the circuit.
- Of the $n - 1$ qubits, m qubits are data qubits.
- Of the $n - 1$ qubits, $n - 1 - m$ qubits are ancillary qubits.
- The data qubits comprise a set M , where $\forall i \in \{2^k \mid k \in \mathbb{Z}, 0 \leq k < \log_2(n - 1)\}$ is the i -th overall qubit in the set M .
- The whole quantum circuit represents a state $|\psi\rangle \in \mathbb{C}^{2^{n-1}}$.
- When all the qubits from the set M are measured, the vector of probabilities of measuring each possible state is characterized as:

$$\sum_{j=1}^n v_j^2 |e_j\rangle,$$

where $|e_j\rangle$ is the j -th basis state of the standard basis, with each standard basis vector having a length of n .

- The circuit uses $n - 1$ rotational gates RY , where each qubit has exactly one of these gates applied on it.
- After the gates RY , the qubits are entangled using the gates $CSWAP$, where the number of these gates $CSWAP$ in the circuit is equal to:
- Let $S(x)$ denote the following expression:

$$S(x) = \sum_{d=0}^x 2^d(x - d).$$

Then:

- $S(m)$ equals the total number of all gates used in the circuit.
- $S(m - 1)$ equals the number of the gates $CSWAP$ used in the circuit.
- $S(m) - S(m - 1) = n - 1$ equals the number of the gates RY used in the circuit.



The process of the *CSWAP* gate application needs to be generalized so that it can be used for any given input vector that is normalized, real, and has a dimension of n , where n is a power of two larger than 2. To gain a better understanding of how the divide-and-conquer principle works in a general way, the explanation can be presented in the form of an algorithm that applies all the required gates *CSWAP* based on the length of a given input vector. The algorithm is presented in the form of a pseudocode. Pseudocode representation abstracts from any implementation specifics and language semantics, only focusing on the logic of the underlying algorithm, directly conveying its essence. The pseudocode is presented below:

Pseudocode 1: *CSWAP* application algorithm

Input: Quantum circuit with $n = 2^m - 1 \geq 3$ qubits

```

1 Function get_left_node(index):
2   | return  $2 \cdot \text{index} + 1$ 
3 Function get_right_node(index):
4   | return  $2 \cdot \text{index} + 2$ 
5  $\text{last\_qubit} \leftarrow n$ 
6 for  $\text{current\_qubit} \leftarrow (\text{last\_qubit} - 1)/2$  to 1 by -1 do
7   |  $\text{left\_qubit} \leftarrow \text{get\_left\_node}(\text{current\_qubit})$ 
8   |  $\text{right\_qubit} \leftarrow \text{get\_right\_node}(\text{current\_qubit})$ 
9   | while  $\text{right\_qubit} \leq \text{last\_qubit}$  do
10  |   | Swap  $\text{left\_qubit}$  and  $\text{right\_qubit}$  with  $\text{current\_qubit}$  as
11  |   |   | control qubit using gate CSWAP
12  |   |  $\text{left\_qubit} \leftarrow \text{get\_left\_node}(\text{left\_qubit})$ 
12  |   |  $\text{right\_qubit} \leftarrow \text{get\_left\_node}(\text{right\_qubit})$ 

```

This algorithm utilizes the mentioned divide-and-conquer strategy to apply the required *CSWAP* gates to qubits in a given quantum circuit. To summarize, the divide-and-conquer encoding method works by:

- first calculating the required angles using the input vector elements in the same way as the amplitude encoding method does,
- then encoding the calculated angles in the same way as the angle encoding method does,
- and finally combining the qubits using the algorithm above, with the algorithm utilizing the divide-and-conquer strategy to entangle the qubits.

This algorithm is part of a larger algorithm presented in the subsection Divide-and-conquer encoding method (4.2.4) of the chapter Implementation (4). The algorithm in that subsection represents the entire divide-and-conquer encoding method, not just showcasing how the gates *CSWAP* are applied. Now, with

the entire process of the divide-and-conquer encoding method elucidated, it is beneficial to explore certain notable aspects of the divide-and-conquer encoding method, while also comparing these aspects to those of the amplitude encoding method.

3.3.6 Implications

The last subsection of this entire chapter is devoted to discussing some noteworthy implications that arise from the nature of the divide-and-conquer encoding method. The following points discuss those implications and also explore how they connect to the amplitude encoding method and other encoding methods:

► Remark 3.30. Let $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $n = 2^m$, $m \in \mathbb{N}$, $\|\mathbf{v}\| = 1$ be an input vector that is encoded using the divide-and-conquer encoding method.

- This encoding method utilizes fewer gates in comparison with the classical amplitude encoding method. Specifically, the number of gates used is approximately two times greater than the number of input vector elements, meaning the gate count scales linearly with the input vector length. This is a significant improvement over the amplitude encoding method, where the gate count scales exponentially with the input vector length. However, the number of qubits this encoding method uses also scales linearly, compared to the amplitude encoding method, where the qubit count scales logarithmically. So, in essence, the gate and qubit counts of this encoding technique revert back into the realm of the angle and basis encoding methods, where these counts also increase linearly. There is a clear trade-off between the qubit count and the gate count of this encoding method (more qubits used but fewer gates used) and the amplitude encoding method (fewer qubits used but more gates used). In practical terms, this trade-off means that certain aspects must be considered. For instance, when the qubit count scales linearly, the outcome of this scaling might have adverse effects on quantum computations because the number of available qubits on quantum computers is a limiting factor, given the current technological constraints. The same can be said about the exponential gate counts of the amplitude encoding method. It also poses a significant challenge, as it leads to deeper and more complex quantum circuits that are more error-prone (as applying gates on real quantum hardware may introduce errors in computations [31]) and harder to execute efficiently.

Ideally, both qubit and gate counts should be as low as possible to optimize quantum circuit performance. However, achieving such an ideal scenario where both counts are logarithmical may not be feasible due to the inherent limitations of quantum computing. Typically, if one metric is optimized to be logarithmical, the other tends to be less favorable, often resulting in exponential growth. Therefore, a balanced encoding method that strikes a middle ground between these two extremes could offer significant benefits.

A balanced encoding method would ideally have both qubit and gate counts that are neither logarithmical nor exponential, but lie somewhere in between. Such a method would provide a more practical approach, reducing the overall resource requirements and enhancing the feasibility of performing large-scale quantum computations. The potential to combine the strengths of both the divide-and-conquer and amplitude encoding methods to achieve this balance represents an avenue for further research. Although a perfectly balanced encoding method has not been identified within the scope of the research part of this thesis, the insights gained and presented highlight the potential for further investigation. Combining elements of these two encoding methods to potentially develop a more balanced approach could lead to significant advancements in gate and qubit counts.

The topic of the required qubit and gate counts of all encoding methods is explored in great depth in the section Empirical complexity comparison (5.1) of the chapter Testing (5).

- This encoding method can be made more efficient by reducing the number of gates it uses in one scenario. As mentioned previously, if a rotational gate does not impact the targeted qubit, employing this gate becomes unnecessary. In this case, the gates RY that do not modify initial qubit states can be omitted from the circuit. Moreover, if more such gates are omitted, multiple qubits may be in the state $|0\rangle$ when entering the phase when the gates $CSWAP$ are applied. If some specific gate $CSWAP$ targets two qubits that are both in the state $|0\rangle$, the swapping would result in no change (regardless of the control qubit's control state). In these cases, such gates $CSWAP$ can also be skipped, resulting in a lower gate utilization. The extent of the efficiency boost would be limited by the number of rotations that can be disregarded.
- This encoding method typically works for input vectors of lengths equal to some power of two greater than three, specifically $2^m > 3$, $m \in \mathbb{N}$. Nevertheless, this constraint can be bypassed. Given an input vector of a truly arbitrary length $n \in \mathbb{N}$ (where n is not a power of two and may be smaller than 4), this vector can be encoded using this encoding method if the following procedure is performed:
 1. First, the input vector is enlarged so that its length becomes the nearest power of two larger than $\max\{n, 3\}$. This is done by appending zeros to the vector's end.
 2. Then, such an enlarged vector can be encoded as usual.
 3. If the appropriate qubits are measured, the vector of probabilities of measuring each possible state can be trimmed so that it only contains the first n elements, where each i -th element of this trimmed vector corresponds to the i -th squared element of the original input vector (before it was elongated).

Similarly, this procedure can be applied alternatively by adding zeros to the beginning of the vector instead and subsequently trimming the probability vector from the start to retain only its final n elements.

- An important advantage of the divide-and-conquer encoding method is its potential for parallel computation. This method allows many gates to be executed concurrently (this can be seen in the figure 3.6), speeding up the encoding process. One of the primary areas where parallelism is leveraged in this encoding method is in the first layer, where the gates RY are added to each qubit. These gates are independent of each other, meaning they can be executed simultaneously. Additionally, certain $CSWAP$ gates can also be performed in parallel. When these gates act on disjoint sets of qubits (children of the left node and children of the right node, with both nodes being in the same binary depth), they can be executed concurrently, further enhancing the parallelism of the encoding process.

The benefits of parallel computations are substantial, including [32]:

- Parallel execution of gates leads to a significant reduction in circuit depth. This reduction is crucial for maintaining coherence in quantum operations, as deeper circuits are more prone to errors because of decoherence and other noise factors inherent in quantum systems.
- By executing multiple gates simultaneously, the total time required for the encoding process is decreased. This efficiency gain is particularly valuable in quantum computing, where the duration of quantum operations can be a limiting factor.

In contrast, the amplitude encoding method does not offer the same level of parallelism. In that encoding method, each multi-controlled gate RY must be applied sequentially to encode the input vector correctly. The multi-controlled gates depend on the states of multiple qubits and preceding rotations, requiring a strict sequential application to ensure accurate encoding. This sequential nature leads to deeper circuits with higher gate counts, as each gate must wait for the previous one to be added before it can be added too. The lack of parallelism in the amplitude encoding method results in longer encoding times and higher susceptibility to errors due to the increased circuit depth. Consequently, while the amplitude encoding method is efficient in regard to its qubit usage, the sequential gate application limits its overall efficiency and scalability compared to the divide-and-conquer encoding method.

The basis and angle encoding methods also offer parallelism, as just a single layer of gates comprises each quantum circuit they produce.

- The divide-and-conquer encoding method utilizes a significant number of ancillary qubits, to the extent that more ancillary qubits are used than

data qubits. While these ancilla qubits are necessary for the encoding process, they do not directly participate in the resulting output state. This characteristic leads to inefficient use of quantum resources, as a large portion of the quantum circuit is occupied by qubits that do not contribute to the final encoded state.

This inefficiency is disadvantageous compared to the encoding methods that only contain data qubits, such as the amplitude encoding method, where all qubits are directly involved in the encoding process and the resulting quantum state. This direct involvement ensures that the maximum possible resources are dedicated to the actual data representation, optimizing the use of available qubits.

The use of ancillary qubits is not inherently negative. In fact, ancillary qubits can be beneficial in later computations. These qubits could potentially be repurposed to perform additional quantum operations, thereby enhancing the overall utility of the quantum circuit. However, the main issue with ancillary qubits in the divide-and-conquer encoding method is that they become entangled during the encoding process. Entangled qubits are more difficult to repurpose for subsequent computations due to the correlations established between them and the data qubits. This entanglement restricts the flexibility of the ancillary qubits, limiting their potential utility in subsequent quantum operations.

To overcome this challenge, it would be advantageous to develop a method to unentangle the ancillary qubits after the encoding process. If the divide-and-conquer encoding method could be modified to unentangle the ancillary qubits, it would lead to significant advancements in the efficiency and practicality of this encoding method because it would allow the unentangled qubits to be reused for further quantum computations. This optimization could be particularly beneficial in scenarios where qubit resources are limited, as it would result in the possibility of more complex quantum algorithms being executed with the same hardware.

However, it was not determined within the scope of this thesis whether it is feasible to unentangle the ancillary qubits in this context. The process of unentangling qubits without disrupting the encoded data state poses significant challenges and requires further investigation. Unentangling qubits typically involves applying additional quantum operations that reverse the entangling process without disturbing the encoded information in the data qubits. This topic lies beyond the scope of the thesis objectives and represents an area for possible future research.

- The divide-and-conquer encoding method, as presented in this thesis, is only designed to encode real vectors. This is because it only utilizes the gates RY . This gate is unable to account for the phase information found in complex numbers. However, it is possible to extend this encoding method

so that it encodes complex vectors. That can be achieved by an additional layer of gates employed at the beginning of the encoding process; more specifically, the RZ gates can be used for this purpose. A similar approach is explored in the part 3.12 of the section Angle encoding method (3.2). This approach is reiterated below:

For encoding a complex number $c = re^{i\theta} \in \mathbb{C}$, $r \in \mathbb{R}$, $\theta \in [0, 2\pi] \subset \mathbb{R}$ such that $|c|^2 \in [0, 1] \subset \mathbb{R}$, the absolute value r can be encoded using the gate RY , while the phase $e^{i\theta}$ of the complex number can be encoded using the gate RZ , introduced before or after the gate RY . The gate RZ performs a rotation around the Z-axis of the Bloch sphere, which corresponds to adding a phase to the quantum state. By applying the gate RZ , the phase of the complex number is encoded into the qubit.

A similar approach can also be utilized in the case of the divide-and-conquer encoding method, where a layer of the gates RZ is added before or after the layer of the gates RY . The process of calculating the required angles used in the gates RZ is slightly different compared to the process described in the definition i -th angle calculation (3.18), adopted to account for the retrieval of phase information. The combination of the RY and RZ gates thus allows the encoding of both the magnitude and phase of a complex number. The gates $CSWAP$ applied in the later part of the encoding process would not require any changes, as the only effect of theirs is the combination of qubits using the divide-and-conquer principle.

Such a modification would enable the divide-and-conquer encoding method to handle complex vectors, thus broadening its applicability to a wider range of quantum algorithms that require the encoding of complex data. Implementing this additional layer of the gates RZ does introduce extra computational steps; however, it provides a comprehensive approach to encoding both the amplitude and phase information of complex vectors, making the encoding method more versatile and powerful.

- The idea discussed in this point concerns not only the divide-and-conquer encoding method but all encoding methods. Typically, in the process of encoding classical data into quantum states, it is uncommon to encode merely a single vector. Usually, several vectors require encoding and subsequent processing. Such vectors often comprise some dataset that can contain many vectors. All the encoding methods discussed in this thesis are presented in a way that only allows for the encoding of a single input vector. So, when given a dataset of input vectors, the dataset cannot be directly encoded using the provided encoding method processes. However, that is not to say that it cannot be encoded at all in any way. In fact, there might be many different strategies to tackle this issue. This topic goes beyond the thesis scope and, therefore, is not explored in detail. However, two different approaches that confront this issue are still presented to

illustrate the idea behind how two varying concepts are capable of being utilized to address this challenge.

Let there be a dataset D , where $D = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m\}$, $m \in [2, \infty) \subset \mathbb{N}$ is a set consisting of m data vectors of dimension¹¹ $n \in \mathbb{N}$, such that $\forall \mathbf{v}_i \in D : \mathbf{v}_i \in \mathcal{F}^n$. It is assumed that this dataset is supposed to be encoded using one of the encoding methods introduced in this thesis. Then, let \mathcal{F} be a vector space that is appropriate for the chosen encoding method. Below are presented two distinct approaches to encode D using the chosen encoding method:

- The following mapping function $f_1 : D \rightarrow \mathcal{F}^{mn}$ is defined as:

$$f_1(D) = \bigoplus_{i=1}^m \mathbf{v}_i = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \dots \oplus \mathbf{v}_m = \mathbf{v}_{\mathbf{d}} \in \mathcal{F}^{mn},$$

where \oplus denotes the vector concatenation operation.

The vector $\mathbf{v}_{\mathbf{d}}$ can then be used directly as input in the basis encoding method and after normalization in the other encoding methods.

This approach simply uses concatenation to create the vector $\mathbf{v}_{\mathbf{d}}$ by combining all dataset vectors into one by appending them all one after another. This implies a considerable qubit demand if n or m are large numbers, essentially making this approach practical only for very small datasets, where the dimension of the resulting vector $\mathbf{v}_{\mathbf{d}}$ is at most in the thousands, as the most advanced quantum computers available currently only have four-digit qubit count. Nevertheless, this approach has one advantage—no information is lost during the transformation, as all elements of all dataset vectors are represented in the final output vector $\mathbf{v}_{\mathbf{d}}$.

- The following mapping function $f_2 : D \rightarrow \mathcal{F}^n$ is defined as:

$$f_2(D) = \bigoplus_{i=1}^m \mathbf{v}_i = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \dots \oplus \mathbf{v}_m = \mathbf{v}_{\mathbf{d}} \in \mathcal{F}^n,$$

where \oplus denotes the vector summation operation such that when two vectors are summed, the result is a vector of the same length, where its i -th element is a sum of the i -th element from the first vector and i -th element from the second vector, with the sum of the elements being in accordance with the space \mathcal{F}^n , meaning if for example $\mathcal{F}^n = \mathbb{Z}_2^n$, the sum must be modulo two so that the summation result is also from \mathbb{Z}_2^n .

The vector $\mathbf{v}_{\mathbf{d}}$ can then be used directly as input in the basis encoding method and after normalization in the other encoding methods.

¹¹To keep it concise, cases where the dataset includes vectors of varying dimensions are excluded.

This approach may be better suited for qubit-sensitive applications, where the number of available qubits would not suffice for the first approach. This approach produces an output vector that has the same length as each dataset vector. It may seem that this approach is much better than the first one because of the fact that it produces much shorter output vectors. However, that inherently means that not all the information from the dataset is retained. This is because this transformation effectively acts as a reduction in dimensionality. For example, when these three binary numbers $b_1 = 0, b_2 = 1, b_3 = 1$ are summed, the result is the same as if only the first number were used without summing the next two.

In summary, two distinct approaches were presented: vector concatenation and vector summation. The vector concatenation approach, although ensuring no loss of information, demands a significant number of qubits, making it impractical for large datasets. On the other hand, the vector summation approach is more qubit-efficient but may inherently result in some loss of information due to dimensionality reduction. These strategies highlight the trade-offs involved in dataset encoding and underscore the complexity of adapting classical data for quantum processing. This significant and broad subject is not further investigated as it falls outside the scope of the thesis objectives.

- This last point touches on one paramount aspect of the divide-and-conquer encoding method that was not explained in this thesis previously.

In all previous encoding methods, the state vector of the entire quantum circuit was used to represent the desired state. This approach is convenient because it utilizes the fact that the system is in a *pure state*. Pure states are quantum states that can be described by a single state vector, representing a definite, non-probabilistic state of a given quantum system. However, the divide-and-conquer encoding method may prepare states that cannot be represented using state vectors alone, as the quantum circuits produced by this encoding method may be in the so-called *mixed states*.

The divide-and-conquer encoding method described in this thesis does not rely on the state vector to characterize the resulting quantum state. Instead, it is characterized by a probability vector, which contains the probabilities of measuring each possible state if the appropriate qubits are measured. This is because the use of the divide-and-conquer encoding method may result in a mixed state. And, the crucial fact is that mixed states cannot be adequately represented by a state vector. Mixed states are represented using *density matrices*.

A mixed state arises when a quantum system is not in a single pure state but rather a statistical ensemble of different states. This characteristic

necessitates the use of density matrices, which can encapsulate the probabilistic nature of mixed states. The density matrix of a given quantum state provides its complete description, including all its statistical mixtures. Mathematically, a mixed state is represented by a density matrix ρ , which is a positive semi-definite Hermitian operator with trace equal to one, acting on the Hilbert space of the quantum system. The diagonal elements of the density matrix represent the probabilities of the system being in each corresponding pure state.

To extract the state vector of a pure quantum state that uses ancillary qubits from its density matrix, an operation called *partial trace* is required. The partial trace operation effectively reduces the density matrix by tracing out the degrees of freedom associated with the ancillary qubits, leaving a reduced density matrix that corresponds to the state of the data qubits. This operation produces a reduced density matrix that describes the state of the subsystem of interest. However, this operation cannot produce a state vector when the system is in a mixed state, as mixed states inherently cannot be described by a single state vector. Mixed states require density matrices for their representation.

The use of the probability vector (that can be derived from the density matrix) allows for the characterization of this encoding method in a manner that aligns with the desired outcomes. A probability vector reflects the probabilities of measuring each possible state corresponding to the squared input vector elements.

These concepts were not presented in the thesis earlier because they are outside the primary aims of the thesis. Correctly defining the concepts of mixed states, density matrices, and partial trace operations would introduce complexities that are beyond the level of this thesis. Nevertheless, recognizing that the divide-and-conquer encoding method prepares mixed states is crucial for understanding its operation and potential applications. Even though the divide-and-conquer encoding method's reliance on mixed states and density matrices complicates the representation of quantum states it produces, acknowledging these concepts provides a unique perspective on how varying different data encoding strategies can be.



Implementation

One of the objectives of this thesis is to implement the data encoding approaches presented in the previous chapter—Data encoding methods (3). The implementation of the encoding techniques was carried out using *Python* as the selected programming language. One of the reasons Python was chosen is due to its position of being the most popular programming language in the domains of AI and ML [35]. Python is also the primary programming language in the field of quantum programming, since leading quantum software development kits are available in the form of Python libraries. One of those libraries is *Qiskit*, which is referred to as the “world’s most popular quantum software” [36] by *IBM*—the organization that develops it. “Qiskit is an open-source SDK for working with quantum computers at the level of extended quantum circuits, operators, and primitives”, and it is available on a *GitHub* repository [37].

So, the four encoding methods introduced in the last chapter Data encoding methods (3) were implemented in Python using Qiskit. All code developed as part of this thesis is available in a publicly accessible GitHub repository, with the appendix of this thesis presenting its exhaustive structure and the URL to the repository. This chapter is divided into two main parts:

1. The first part briefly outlines the contents of the repository along with the purpose of the files contained in the repository. This part does not deal with each repository file on an individual level; instead, it describes what code was developed during the creation of this thesis in general, with a rough explanation of its functionality.
2. Each implementation of the encoding method is accompanied by an algorithm that represents the encoding method (the algorithm creates a desired quantum circuit that encodes a given input). In this second part, a pseudocode is provided for each of these algorithms (such a pseudocode only serves as a rough illustration of how an encoding algorithm works, abstracted from any specifics of the Python programming language and the Qiskit library).

4.1 Repository overview

The practical component of this thesis is encapsulated within the aforementioned repository. This section provides a detailed (but not exhaustive) description of the repository that encompasses the practical implementation of the quantum data encoding methods discussed in the chapter Data encoding methods (3). The repository is meticulously organized, designed to provide a comprehensive implementation framework and demonstration of the data encoding methods. The repository is strategically divided into two primary sections, with one section containing the Python implementation of the encoding methods and the other section containing Jupyter Notebooks showcasing those encoding methods. This is apparent from the repository's top-level directory bifurcation into two main subdirectories:

1. `qsp/`
Contains Python modules that implement the encoding methods.
2. `notebooks/`
Houses Jupyter Notebooks that demonstrate the usage and effectiveness of the implemented encoding methods.

The interaction between these two sections is symbiotic; the Python modules define the functional backbone of the encoding methods, while the Jupyter Notebooks serve to illustrate and explain these methods' usage and implications visually. This bifurcation serves several purposes:

- By segregating the implementation code and the demonstration Notebooks, the repository maintains a clean separation of core logic from its demonstrational and visual presentation. This approach enhances code manageability and ensures that modifications in the implementation do not directly impact the Notebooks that showcase the encoding methods.
- The modular nature of the repository allows for each component to be independently developed, tested, and optimized without affecting other parts. This structure is particularly advantageous in a situation where encoding methods may need to be adjusted or extended.
- The dual structure facilitates understanding by providing a clear path from reviewing the implementation code to observing its effects in a visual and interactive manner through the Jupyter Notebooks, or vice-versa.

► Note 4.1. Adoption of Qiskit 1.0 With the recent official release of *Qiskit 1.0* in February 2024, IBM has introduced significant improvements in the performance and stability of its quantum computing SDK, marking a major update from its predecessor versions [38]. By integrating this new version of Qiskit into the repository, the codebase not only leverages these advancements

but also ensures readiness for future updates, adhering to the latest standards. Opting for this latest major release ensures compatibility with upcoming versions of Qiskit and facilitates the potential future expansion of the codebase. More specifically, the code in the repository uses the latest version of Qiskit (as of May 2024), the version 1.1.0. ◀

4.1.1 Implementation details

The `qsp` directory, which forms the core of the implementation, houses several Python modules, each tailored to a specific quantum data encoding method. The structure within this directory is designed to promote object-oriented programming principles, enhancing the flexibility and reusability of the code. The `qsp` directory is in and of itself a Python package capable of being imported and used to encode input vectors using the implemented encoding methods inside the package.

At the foundation of the `qsp` package (an abbreviation of the term *quantum state preparation*) is the script `base.py` that contains the abstract base class `QuantumStatePreparation`. This class houses several common methods used across different encoding methods. This base class provides standardized methods that are essential for encoding input vectors, such as a method that initializes an empty quantum circuit or a method that applies the required measurements. Four subclasses representing the four encoding methods then inherit from this base class and override or extend its methods to implement their respective encoding strategies. The design rationale behind using an inheritance model is supported by these benefits:

- Instances of any encoding class can be created without requiring a deeper understanding of the underlying quantum circuit construction mechanisms and other Qiskit specifics.
- New encoding methods can be added seamlessly by inheriting the base class, adhering to the established method signatures.
- Centralizing common functionality in the base class reduces code duplication and eases maintenance, as changes to the common features need to be made only once.

The abstract parent class `QuantumStatePreparation` is then inherited by these four child classes located in the directory `qsp/encodings`:

1. The class `BasisEncoding` contained in the module `basis.py`.
2. The class `AngleEncoding` contained in the module `angle.py`.
3. The class `AmplitudeEncoding` contained in the module `amplitude.py`.
4. The class `DivideAndConquerEncoding` class contained in the module `divide_and_conquer.py`.

Each of these subclasses focuses on the corresponding encoding method. These subclasses not only use the foundational methods provided by the parent class `base.py` but also introduce additional methods and quantum operations that are unique to their respective encoding techniques. Each of these child classes implements the abstract method `_encode_input_vector()` that must contain an algorithm that encodes a given input vector by utilizing some Qiskit gates. Another method each subclass must implement is the abstract method `_validate_input_vector()`, which validates the input vector and initializes class attributes (such as the qubit count and a list of the measured qubits).

A given input vector can be encoded by creating an instance of one of the available classes (`BasisEncoding`, `AngleEncoding`, `AmplitudeEncoding`, `DivideAndConquerEncoding`) and passing the input vector in the constructor. After encoding the given input vector this way, the base parent class then provides a multitude of essential properties and methods to use:

- Properties of the abstract class `QuantumStatePreparation`:
 - `input_vector`: Returns a copy of the input vector that is encoded in the quantum circuit.
 - `qubit_count`: Returns the number of qubits used in the quantum circuit that is used to encode the input vector. This value is initialized upon validation of the input vector.
 - `quantum_circuit`: Provides access to the quantum circuit that encodes the input vector.
 - `measured_qubits`: Returns a list of indices of the qubits that should be measured to obtain the encoding results when performing a measurement. All encoding methods, except the divide-and-conquer encoding method, require all qubits to be measured.
- Methods of the abstract class `QuantumStatePreparation`:
 - `measure()`: Prepares the quantum circuit for measurement by removing any existing measurements and applying new measurements to the qubits specified by the property `measured_qubits`.
 - `run_aer_simulator(shots, show_plot)`: Runs the quantum circuit that encodes the input vector on the *Aer* simulator with the default options, setting the number of simulation shots and optionally displaying a histogram of the results.
 - `get_statevector()`: Retrieves the state vector of the quantum state that represents the encoded input vector.

4.1.2 Jupyter Notebooks

The Jupyter Notebooks contained within the `notebooks` directory serve a dual purpose: they demonstrate the practical use of the implemented data encoding

methods and analyze their performance. The demonstrations not only bolster understanding of the theoretical concepts discussed in the previous chapter but also provide critical insights into the practical implications and efficiency of these encoding methods. Most of the Notebooks provide an interactive environment with multiple visualizations, where certain parameters can be modified to directly observe the effects of the changes. The Notebooks are categorized as follows:

- The first Notebook, `readme_libraries.ipynb`, is designed to assist in setting up a computational environment for running the demonstrations provided in the subsequent Notebooks. It includes a thorough list of all the necessary Python libraries required to run the Notebooks and use the `qsp` package. The versions of the libraries are specified to ensure compatibility, meaning that a virtual Python environment can be correctly configured to support all the repository features. This setup is crucial for replicating the results shown in the Notebooks.
- The Notebook `circuit_complexity.ipynb` focuses on the computational complexity analysis of the quantum circuits generated by the different encoding methods. It evaluates the number of gates and qubits required by each encoding method to encode input vectors of various lengths. This analysis is essential for understanding the scalability of each encoding method and its suitability for different input vectors. The Notebook provides visualizations in the form of tables that help illustrate the mentioned metrics.
- Lastly, the Notebooks within the `notebooks/encoding_demonstration` subdirectory are meticulously designed to provide a comprehensive experience that showcases the application of each implemented data encoding method. These are the Notebooks contained in this subdirectory:
 - `basis.ipynb`,
 - `angle.ipynb`,
 - `amplitude.ipynb`,
 - `divide_and_conquer.ipynb`.

All Notebooks in this series follow the same structured format. The purpose of each of these Notebooks is to demonstrate the corresponding encoding method on a pseudo-randomly generated input vector. Here is a brief description of the structure that each Notebook utilizes:

1. An input vector of a given length is generated according to the needs of the encoding method.
2. The input vector is then encoded using the corresponding encoding method from the `qsp` package. During the encoding process, a quantum circuit is created, where the state that this circuit produces represents the input vector.

3. A simulation is then performed using Qiskit's default Aer simulator, providing immediate feedback on the circuit's performance and the encoding outcomes.
4. The simulation results are then compared to the expected results calculated from the input vector.
5. A real quantum computer is chosen to run the quantum circuit. A job is then sent to the chosen backend instance, and the measurement results are retrieved when the instance finishes running the circuit.
6. These three types of values are compared:
 - The measurement results obtained by running the circuit on the chosen quantum computer.
 - The simulation results.
 - The expected results calculated from the input vector.

These values are presented in the form of a histogram.

7. Finally, two metrics are calculated to evaluate the quality of the state produced by the chosen quantum hardware (2.6).

These Notebooks include visualizations such as quantum circuit diagrams and histograms of probability distributions. Those visualizations are also used in this thesis text.

4.2 Pseudocode

Following the detailed exposition of the repository structure in the previous section and the theoretical foundations laid out in the previous chapter Data encoding methods (3), this section transitions to a more granular depiction of the encoding methods. Specifically, this part delves into the algorithmic representations of the data encoding techniques in a way where the algorithms found in the package `qsp` are explained through pseudocode. The purpose of presenting pseudocode in this section is to:

- Offer a clear and language-agnostic illustration of the algorithms behind each data encoding method.
- Serve as an essential bridge between the theoretical underpinnings of the encoding methods and their practical implementation using Python and the Qiskit library.
- Abstract away from the syntactic details of Python, instead focusing on the logical flow of the algorithms.

In the following subsections, the algorithm of each implemented encoding method is laid out in a block of pseudocode that encapsulates the essence of

the corresponding data encoding method, illustrating how a given input vector is encoded and offering a holistic view of both the conceptual and practical aspects of the encoding method.

4.2.1 Basis encoding method

The basis encoding method (explained in the section Basis encoding method, 3.1) is one of the simplest yet most fundamental encoding methods utilized to represent classical data using quantum states. The following pseudocode explains the algorithm implemented in the class `BasisEncoding`:

Pseudocode 2: Basis encoding method algorithm

Input: Binary vector $\mathbf{b} = (b_1 \ b_2 \ \dots \ b_n)$ of length n

- 1 Initialize quantum circuit qc with n qubits and initial state $|0\rangle^{\otimes n}$
- 2 **for** $i = 1$ to n **do**
- 3 **if** b_i is 1 **then**
- 4 Apply gate X to i -th qubit of qc
- 5 **return** qc

This algorithm starts by initializing a quantum circuit with as many qubits as there are elements in the input vector. Each qubit is initially in the default state $|0\rangle$. The algorithm then iterates over each element of the input vector, and if the element equals 1, the algorithm applies the gate X (2.4.1) to the corresponding qubit, which flips the qubit from the state $|0\rangle$ to the state $|1\rangle$. This direct mapping of binary elements onto qubits' states exemplifies the encoding method's simplicity and directness, making it easy to implement.

4.2.2 Angle encoding method

The angle encoding method (introduced in the section Angle encoding method, 3.2) is another relatively simple encoding method utilized to encode classical data into quantum states. The pseudocode below outlines the algorithm implemented in the class `AngleEncoding`:

Pseudocode 3: Angle encoding method algorithm

Input: Normalized real vector v of length n

- 1 Initialize quantum circuit qc with n qubits and initial state $|0\rangle^{\otimes n}$
- 2 **for** $i = 1$ to n **do**
- 3 Apply gate RY to i -th qubit of qc with angle $2 \arcsin(v_i)$
- 4 **return** qc

The algorithm begins by setting up a quantum circuit, initializing a number of qubits equal to the number of elements in the input vector. Initially, every qubit is in the state $|0\rangle$. The algorithm then iterates over each qubit and

applies the gate RY to it with an angle calculated as $2 \arcsin(v_i)$ (3.2), where v_i is the corresponding element of the input vector. This implies that the algorithm is similarly simple and straightforward to implement, akin to the one discussed in the preceding subsection.

4.2.3 Amplitude encoding method

The amplitude encoding method (elaborated in the section Amplitude encoding method, 3.3) is more advanced compared to the previous two encoding methods and requires a more sophisticated algorithm. This algorithm systematically applies quantum gates to encode a given normalized input vector containing real elements, where each element is encoded into the corresponding amplitude of a quantum state. This algorithm uses a list of pre-computed angles that determine the rotations necessary to gradually align the quantum state with the input vector. It employs a sequence of multi-controlled rotational gates RY to incrementally build the state, ensuring that each qubit contributes correctly to the desired quantum state configuration. This algorithm is located in the class `AmplitudeEncoding`, and its pseudocode is:

Pseudocode 4: Amplitude encoding method algorithm

Input: Normalized vector v of length $n = 2^m$ containing real numbers

- 1 Initialize quantum circuit qc with m qubits and initial state $|0\rangle^{\otimes m}$
- 2 $angles \leftarrow \text{CalculateAngles}(v)$
- 3 Apply gate RY to m -th qubit with angle $angles[0]$
- 4 $angle_index \leftarrow 1$
- 5 **for** $index = 0$ **to** $qubit_count - 2$ **do**
- 6 $target_qubit \leftarrow m - index - 1$
- 7 $level \leftarrow index + 1$
- 8 $gates_on_this_level \leftarrow 2^{level}$
- 9 **for** $number = 0$ **to** $gates_on_this_level - 1$ **do**
- 10 $angle_to_encode \leftarrow angles[angle_index]$
- 11 $control_qubits \leftarrow \text{range}(target_qubit + 1, m)$
- 12 $control_states \leftarrow \text{list}(\text{BinaryString}(number, level))$
- 13 $\text{ApplyMCRY}(angle_to_encode, target_qubit, control_qubits,$
 $control_states)$
- 14 $angle_index \leftarrow angle_index + 1$
- 15 **return** qc

The algorithm starts by applying a rotation to the last qubit, and then subsequent loops iteratively refine the quantum state by targeting each qubit with multi-controlled RY rotations that depend on the state of the previously configured qubits, thus building the desired quantum state layer by layer (this process is described in greater detail in the section 3.3), going from the last qubit to the first qubit while doubling the number of the rotational gates on

each subsequent qubit. This ordering may seem reversed, but it adheres to the Qiskit conventions [39]. For the sake of clarity, some minor details of lesser significance are omitted from the pseudocode (such as the ability to encode input vectors of sizes other than $n = 2^m$ by padding them with zeros at the end). The complete algorithm is available in the repository.

Now, it is vital to explain the functions `BinaryString`, `ApplyMCRY`, and `CalculateAngles` used in the pseudocode. The function `BinaryString` takes an integer (*number*) and returns its binary representation, padded with zeros if needed so that the returned string contains enough bits (equal to *level*). This string is then broken down to create a list of the controlled states required for each multi-controlled *RY* gate. The function `ApplyMCRY` applies a multi-controlled *RY* gated based on the parameters passed. It appends the gates *X* to the control qubits that have 0 as their control state. And finally, the function `CalculateAngles` calculates the angles used in the rotations. This function is crucial for this algorithm and needs its own pseudocode to be presented:

Pseudocode 5: Calculate angles

Input: Normalized vector v of length $n = 2^m$ containing real numbers

```

1  $probs \leftarrow v^2$ 
2 Initialize  $angles\_probs$  as list of lists
3 Initialize  $angles$  as list
4 for  $binary\_level = 0$  to  $m - 1$  do
5    $probs\_in\_angle \leftarrow n/2^{binary\_level+1}$ 
6   for  $counter = 0$  to  $n - 1$  by  $probs\_in\_angle$  do
7      $angle\_probs\_sum \leftarrow$ 
8       sum of  $probs$ [from  $counter$  to  $counter + probs\_in\_angle$ ]
9     Append  $angle\_probs\_sum$  to  $angles\_probs[binary\_level]$ 
10    if  $counter \bmod 2 = 1$  then
11      | Continue to next iteration
12    if  $binary\_level > 0$  then
13      |  $parent\_probs \leftarrow angles\_probs[binary\_level-1][counter/2]$ 
14      | if  $parent\_probs \neq 0$  then
15        | |  $angle\_probs\_sum \leftarrow angle\_probs\_sum/parent\_probs$ 
16       $angle\_to\_append \leftarrow 2 \arccos(\sqrt{angle\_probs\_sum})$ 
17      if  $binary\_level = m - 1$  then
18        | if  $v[counter + 1] < 0$  then
19          | |  $angle\_to\_append \leftarrow -angle\_to\_append$ 
20        | if  $v[counter] < 0$  then
21          | |  $angle\_to\_append \leftarrow 2\pi - angle\_to\_append$ 
22      Append  $angle\_to\_append$  to  $angles$ 
23 return  $angles$ 

```

To enhance comprehensibility, certain less significant aspects of this algorithm are excluded from this pseudocode as well (such as certain minor optimizations and the possibility of processing input vectors of sizes other than $n = 2^m$). The complete algorithm can be found in the repository, also in the class `AmplitudeEncoding`. In a nutshell, the essence of this function `CalculateAngles` can be summarized as calculating the probability of each qubit collapsing into the state $|0\rangle$ and then using these probabilities in the equation $2 \arccos(\sqrt{\text{angle_probs_sum}})$ to calculate the required angles used in the rotations. This process is also elaborated in the section 3.3.

4.2.4 Divide-and-conquer encoding method

The final encoding method presented in the chapter Data encoding methods (3) is inspired by the classical amplitude encoding method. It uses the divide-and-conquer principle during state preparation. The subsection Divide-and-conquer encoding method (3.3.5) introduces and explains this encoding method. The implementation of this encoding method is housed in the class named `DivideAndConquerEncoding`. It contains an algorithm that uses the divide-and-conquer strategy (3.3.5) to encode a given input vector. This strategy is based on the controlled *SWAP* gates used to create a quantum state that contains entangled qubits and ancillary qubits. The pseudocode of this algorithm is shown below:

Pseudocode 6: Divide-and-conquer encoding method algorithm

Input: Normalized vector v of length $n = 2^m$ containing real numbers

- 1 Initialize quantum circuit qc with $n - 1$ qubits and initial state $|0\rangle^{\otimes n-1}$
- 2 $angles \leftarrow \text{CalculateAngles}(v)$
- 3 **for** $i = 1$ to $n - 1$ **do**
- 4 Apply gate RY to i -th qubit of qc with angle $angles[i - 1]$
- 5 **Function** `get_left_node(index)`:
- 6 **return** $2 \cdot index + 1$
- 7 **Function** `get_right_node(index)`:
- 8 **return** $2 \cdot index + 2$
- 9 $last_qubit \leftarrow n - 1$
- 10 **for** $current_qubit \leftarrow (last_qubit - 1)/2$ to 1 **by** -1 **do**
- 11 $left_qubit \leftarrow \text{get_left_node}(current_qubit)$
- 12 $right_qubit \leftarrow \text{get_right_node}(current_qubit)$
- 13 **while** $right_qubit \leq last_qubit$ **do**
- 14 Swap $left_qubit$ and $right_qubit$ with $current_qubit$ as control qubit using controlled *SWAP* gate
- 15 $left_qubit \leftarrow \text{get_left_node}(left_qubit)$
- 16 $right_qubit \leftarrow \text{get_left_node}(right_qubit)$
- 17 **return** qc

This algorithm extends the traditional amplitude encoding method by integrating controlled *SWAP* operations within a divide-and-conquer strategy, effectively using more qubits but fewer gates than the amplitude encoding method. This hybrid encoding strategy first uses a layer of the gates *RY* for the initial amplitude setup, in which each qubit is rotated. The function `CalculateAngles` presented in the previous pseudocode block is utilized to calculate the required angles used in those rotations. So both this encoding method and the amplitude encoding method use the same approach to calculate the angles used in the rotations. Following the initial rotations, the algorithm orchestrates a series of the gates *CSWAP* (3.5) to reposition the amplitude-encoded information across the qubits in a binary tree traversal fashion, thus entangling the data qubits and the auxiliary qubits (this procedure is introduced in the subsection Divide-and-conquer encoding method (3.3.5) of the chapter Data encoding methods (3)).

► Note 4.2. In all encoding methods except for this one, all qubits should be measured when performing a measurement to evaluate the encoding quality. In this encoding method, there are x qubits, where $x = 2^k - 1$, $k \in [3, \infty) \subset \mathbb{N}$, and each j -th qubit should be measured such that $j = 2^l$, $l \in [3, x] \subset \mathbb{N}$, while all other qubits are ancillary. ◀

► Note 4.3. In the thesis, the divide-and-conquer encoding method algorithm, as presented in the referenced paper [27], was utilized with minimal modifications. The algorithms described in this section that are used by the amplitude encoding method (including the angle calculation algorithm) have been developed from scratch, thereby presenting original work. ◀



Chapter 5

Testing

All the encoding methods this thesis aims to discuss are introduced in the chapter Data encoding methods (3), and their implementation is outlined in the chapter Implementation (4). However, no comprehensive performance evaluation of the encoding methods has yet been presented. This chapter's purpose is exactly that—to test the encoding methods to see their performance and present the results. Specifically, this chapter has these two primary objectives:

1. The first objective is to compare the complexities of all four encoding methods in terms of the qubit and gate counts required to encode input vectors.
2. The implementation of the encoding methods is tested on short random input vectors to evaluate its performance and the quality of the state preparation. This includes testing the implementation on real quantum hardware.

In the chapter Implementation (4), it is explained that Qiskit is used in the implementation of the encoding methods (the motivations for its usage are also discussed therein). Various aspects of this library are mentioned in this chapter, with the official documentation used as a source of information [40]. The first section of this chapter focuses on the first objective.

5.1 Empirical complexity comparison

The objective of this section is to systematically analyze and compare the complexity of the encoding methods. This investigation is driven by the need to understand the efficiency and practicality of those encoding strategies. The comparative analysis focuses on these two primary metrics of the resulting quantum circuits that are produced by the implemented encoding methods:

- The number of qubits used in the circuits.
- The number of gates used in the circuits.

These metrics serve as critical indicators of the resource requirements and computational overhead associated with each encoding method. Regarding the metrics:

- The qubit count is straightforward; it denotes the number of qubits deployed by a quantum circuit.
- The gate count metric is more ambiguous, as it can vary drastically based on what is considered a gate. All gates in a quantum circuit can be characterized by a single gate that acts on the whole quantum state. On the other hand, even the smallest and simplest gate can be written as a product of various other gates, resulting in infinite possibilities to characterize each gate. Due to this ambiguity, this section utilizes three different metrics for gate counts. These are explained below.

Three distinct gate count calculations are utilized to ensure a comprehensive evaluation of the encoding methods. These calculations address different aspects of gate counting and are explained as follows:

1. The first metric counts the gates that are directly added to a quantum circuit during the encoding process when running an algorithm that encodes a given input vector. The algorithms utilized by the implementation of the encoding methods are explained in the form of pseudocode in the chapter Implementation (4). So, for example, in the case of divide-and-conquer encoding method implementation, the gate *CSWAP* (3.5) is counted as being a single gate because the Qiskit's *CSwapGate* [41] is utilized during the encoding process. This approach provides a raw estimate of the number of gates without considering the constraints or specific requirements of the target quantum hardware. This count is useful for understanding the theoretical gate complexity of an encoding method. It serves as a baseline measurement, reflecting the pure algorithmic complexity without any modifications or optimizations. However, in real quantum hardware, only a very limited subset of gates can be used, and all other gates need to be decomposed into such supported gates. So if one *CSWAP* were to be applied in a circuit on a real quantum computer, it would need to be decomposed, resulting in the usage of possibly even ten or more gates, depending on the set of supported gates.
2. As mentioned in the first point, real quantum hardware often only allows certain gates to be used in a quantum circuit. If a circuit uses gates other than the permitted gates, the process referred to as **transpilation** must happen, where the unsupported gates are decomposed into a sequence of gates that are supported. The transpilation process retains the original gate's effects. If a unitary matrix representation of both the original gate and the combined new gates (that are obtained from the transpilation

process) is compared, both matrices would be equal. So, this second metric involves transpiling the quantum circuit to ensure compatibility with quantum hardware. Another purpose of the transpilation process is that it accounts for the specific connectivity patterns of quantum processors. In quantum processors, not all qubits can interact with each other and become connected and entangled. Each quantum processor may have its own framework of qubit connectivity, and if in a quantum circuit, before transpilation, some qubits are connected that have no connection in the processor, the transpilation ensures the resulting quantum circuit is fully capable of being executed on the processor by propagating the entanglement through intermediate qubits.

Each quantum computer may have a different set of natively supported gates and varying qubit connectivity. The transpilation must thus account for a specific quantum computer that is chosen to run a quantum circuit on. So, the resulting gate count after transpilation reflects the actual number of gates that are used when running the original circuit on the chosen quantum hardware.

As explained in the previous chapter Implementation (4), the implementation of the encoding method utilizes Qiskit. This library is developed and maintained by IBM. This library provides the ability to run quantum circuits on quantum computers created by IBM. However, even IBM Quantum machines may differ in their sets of natively supported gates.

So, the most recent IBM processor architecture is used as reference hardware for the transpilation process. As of May 2024, the *IBM Heron* processor family is the latest and most advanced publicly available processor family from IBM, making it a representative model for modern quantum computing capabilities [19].

Specifically, the IBM Heron processor supports these five gates [42]:

- The identity gate, which leaves the qubit state unchanged.
- The gate X (2.1).
- The square root of the gate X , which is a fundamental single-qubit gate used in various quantum algorithms. It is denoted as \sqrt{X} .
- The gate RZ . This gate performs a rotation around the Z -axis of the Bloch sphere by a specified angle, allowing for precise phase adjustments.
- The controlled Z gate, which applies the gate Z to a target qubit only if the control qubit is in the state $|1\rangle$. This gate is fundamental as it is the only supported gate capable of creating entanglement between qubits.

So, when a given circuit is transpiled using this permitted subset of gates, the resulting quantum circuit may differ drastically from the original one,

as all the original unsupported gates must be decomposed. By utilizing the IBM Heron as the reference hardware, the resulting gate count after transpilation offers a practical perspective on the computational overhead required for each encoding method. This metric is crucial for understanding the real-world performance and feasibility of the encoding methods when executed on actual quantum hardware.

3. This final metric further refines the transpiled gate count by considering different optimization levels during the transpilation process.

The transpilation process algorithm can be implemented in various ways, resulting in circuits with different structures and gate counts. This variability arises because each implementation can apply different strategies for optimizing the quantum circuit. The degree of optimization significantly impacts the final gate count, overall performance, and fidelity of the circuit. For example, a basic transpilation might focus on simply mapping the circuit to the hardware's native gates with minimal changes, while a more advanced transpilation could involve extensive reordering of gates, elimination of redundancies, and other sophisticated optimizations [43]. These differences lead to a range of possible resulting circuits, each with its own characteristics and resource requirements.

Variation in optimization levels represents a trade-off between the depth of optimization and the computational resources required to achieve it (transpiling a large circuit using the highest optimization level may be a resource-demanding task). Qiskit's transpilation algorithm offers these four optimization levels [43]:

- a. Level 0 (no optimization): The circuit is transpiled with minimal transformations, primarily ensuring compatibility with the target hardware. Transpilation with no optimization focuses on simply mapping the circuit to the hardware's native gates with minimal changes. It makes the circuit executable without altering its structure significantly, preserving the original sequence of gates as much as possible.
- b. Level 1 (light optimization): Basic optimizations are applied to reduce the gate count and improve the circuit's execution fidelity. This level includes simple techniques such as removing redundant gates and making minor adjustments that do not require extensive computation.
- c. Level 2 (medium optimization): More aggressive optimizations are performed, which might involve significant reordering and transformation of gates. At this level, the algorithm aims to strike a balance between optimization effort and computational efficiency, applying more substantial changes to the circuit to enhance performance while managing resource consumption.
- d. Level 3 (heavy optimization): The most intensive optimizations are applied, potentially leading to substantial changes in the circuit structure

to minimize gate count and maximize performance. This level leverages advanced optimization techniques, thoroughly reworking the circuit to achieve the best possible performance on the target hardware. The process can include extensive gate reordering, merging of operations, and other sophisticated transformations.

This third metric utilizes transpilation with the highest optimization level (level 3). This metric provides a view of the best possible performance of the encoding methods after thorough optimization, showing the minimum gate count achievable on IBM Heron. It represents the theoretical lower bound on the number of gates needed (only Qiskit's transpilation algorithm is considered with its different possible optimization levels).

On the other hand, the metric in the second point uses the lowest optimization level (level 0). That approach gives an insight into the gate count with no optimization, reflecting more closely the raw complexity of the encoding methods. It shows the gate count when the circuit is adapted to the hardware with the least amount of transformation, preserving the initial structure and sequence of operations.

Calculating gate counts for both these levels is beneficial for several reasons. First, it highlights the impact of optimization on gate count and performance. By comparing the gate counts at level 0 and level 3, one can understand how much improvement can be achieved through optimization. Second, it demonstrates the range of possible outcomes, from minimal optimization (indicating the base complexity) to maximum optimization (indicating the optimized complexity). This range provides valuable insights into the efficiency and feasibility of the encoding methods under different optimization strategies.

These three metrics collectively offer a robust framework for evaluating the gate complexity of different encoding methods. By considering both the theoretical and practical aspects of gate counts, this analysis aims to provide a detailed understanding of the resource requirements associated with each encoding method.

The comparative analysis in this section aims to compare the performance of the encoding methods in a way that allows for the most direct comparison of the qubit and gate counts possible. For this reason, it is beneficial to choose a single vector that is encoded using all four encoding methods. However, there is a constraint in the basis encoding method in that it only allows for binary input vectors to be encoded. So, in this comparison, binary vectors are chosen to be encoded so that each encoding method can encode them. More specifically, a length n is given, and then a vector of length n is created, containing only ones as its elements. This is because the basis encoding method only uses gates to encode elements that equal 1; it does not use gates to encode elements that equal 0. In this way, the worst-case scenario for the basis encoding method is

tested. As for the other encoding methods, they would be unable to encode such a vector, because they require the input vector to be normalized. So, the vector is normalized before encoding it with the other three encoding methods. This process of creating vectors is summarized below:

- A number $n \in \mathbb{N}$ is given and an input vector \mathbf{v} is created such that $\mathbf{v} = (1 \ 1 \ \dots \ 1)^T \in \mathbb{R}^n$.
- This vector is then encoded using the basis encoding method.
- The vector \mathbf{v} is normalized so that it can be encoded using the other encoding methods. So, the vector becomes $\mathbf{v} = (v_1 \ v_2 \ \dots \ v_n)^T \in \mathbb{R}^n$, $\|\mathbf{v}\| = 1$.
- This vector is then encoded using the angle, amplitude, and divide-and-conquer encoding methods.
- The resulting qubit and gate counts of all four quantum circuits produced by the encoding process are compared.

For the comparison to be informative, the above process is performed on various input vector lengths, ranging from one-dimensional input vectors up to vectors of dimensions in the thousands. Testing across a wide range of input vector lengths is essential for several reasons:

- By using different input vector lengths, it becomes possible to assess how each encoding method scales with increasing data size. This helps in identifying encoding methods that remain efficient even as the input size grows, which is critical for practical applications where large datasets are common.
- Testing various input lengths facilitates a detailed comparison between the encoding methods. It allows for the identification of methods that are consistently efficient across different scales and those that might excel in specific scenarios but perform poorly in others.
- By simulating large input sizes, this analysis bridges the gap between theoretical complexity and practical constraints. It helps in understanding the limits of current quantum hardware and the challenges that need to be addressed to handle long input vectors effectively.

It is important to note that the gate and qubit counts are calculated without actually running the circuits on quantum computers. This approach is purely empirical in its nature, as it is employed to showcase the theoretical resource requirements for only some specific input vector lengths. Running circuits with a large number of qubits (thousands) and gates (millions) is currently not feasible on existing quantum hardware due to technological limitations. Therefore, this analysis serves as a theoretical framework to understand the potential complexity and resource needs of different encoding methods.

The first comparison, presented below, utilizes the first gate count metric—meaning calculating the gate counts without transpilation.

5.1.1 No transpilation

In the first comparison, no transpilation is performed. (as discussed in the detailed explanation of the first gate count metric). The following table contains the results for input vectors of lengths ranging from $2^0 = 1$ to $2^{11} = 2048$, with the exponent of 2^m being incremented from 0 to 11, resulting in 12 table rows:

Input length	Qubits Basis	Qubits Angle	Qubits Amplitude	Qubits D-a-c	Gates Basis	Gates Angle	Gates Amplitude	Gates D-a-c
1	1	1	1	3	1	1	1	4
2	2	2	1	3	2	2	1	4
4	4	4	2	3	4	4	11	4
8	8	8	3	7	8	8	35	11
16	16	16	4	15	16	16	163	26
32	32	32	5	31	32	32	355	57
64	64	64	6	63	64	64	771	120
128	128	128	7	127	128	128	1667	247
256	256	256	8	255	256	256	3587	502
512	512	512	9	511	512	512	7683	1013
1024	1024	1024	10	1023	1024	1024	16387	2036
2048	2048	2048	11	2047	2048	2048	34819	4083

■ **Figure 5.1** A table containing the number of gates and qubits used by each encoding method when producing a circuit that encodes input vectors of lengths ranging from 1 to 2048 (the lengths are powers of two) with no circuit transpilation performed.

► **Note 5.1.** This table structure is used throughout this whole section, which means that the subsequent tables presented in this section also have this structure. Here is an explanation of the terms used in the table:

- “Input length” is the length of the input vector being encoded.
- “Qubits” represents the number of qubits required by the corresponding encoding method to encode an input vector of a given length.
- “Gates” represents the number of gates required by the corresponding encoding method to encode an input vector of a given length.
- “Basis”, “Angle”, “Amplitude” and “D-a-c” represent that an input vector of a given length is encoded using the basis, angle, amplitude, and divide-and-conquer encoding method, respectively.
- A color scheme is used to differentiate between the two parts of the table, where one part (in blue) contains the qubit counts, and the second part (in red) contains the gate counts. ◀

The results from the table are summarized below; the summarization is divided into two parts. The first part discusses the qubit counts, and the second part discusses the gate counts:

- Qubit count analysis:
 1. Basis encoding method:
 - The qubit count increases linearly with the input length, as each input element is directly mapped to a qubit.
 - For an input length n , the qubit count is n .
 2. Angle encoding method:
 - Similar to the basis encoding method, the qubit count also increases linearly with the input length, leading to a direct 1 : 1 ratio.
 3. Amplitude encoding method:
 - The amplitude encoding shows a more efficient use of qubits, especially for larger input lengths.
 - The qubit count increases logarithmically with the input length, stabilizing at 11 qubits for the largest input length of 2048. This is the only encoding method that utilizes this logarithmicity.
 4. Divide-and-conquer encoding method:
 - The qubit count scales linearly with the input length, starting with 3 qubits for an input length of 1 and rising to 2047 qubits for an input length of 2048.
 - For any input length n larger than 2, the qubit count is $n - 1$.
- Gate count analysis:
 1. Basis encoding method:
 - The gate count grows linearly with the input length, closely following the number of qubits.
 2. Angle encoding method:
 - The gate count mirrors that of the basis encoding method, maintaining a linear increase.
 3. Amplitude encoding method:
 - This encoding method shows a significant increase in gate count, especially for larger input lengths.
 - The gate count grows exponentially, reaching a peak of 34819 gates for an input length of 2048. This reflects the high gate complexity associated with the amplitude encoding method.
 4. Divide-and-conquer encoding method:
 - In this case, the gate count also follows a linear pattern.

- The gate count is approximately twice the length of the input vector.
- The gate count rises to 4083 for the largest input length, indicating a more significant computational burden when encoding longer vectors compared to the basis and angle encoding methods, although much less significant in contrast to the amplitude encoding method.

There are some key insights worth mentioning that stem from the analysis above:

- Efficiency:
 - The amplitude encoding method is the most efficient in terms of qubit usage for larger input lengths but is the least efficient in terms of gate count, reflecting its high computational complexity.
 - The basis and angle encoding methods provide predictable, linear increases in both qubit and gate counts, offering simplicity and ease of implementation.
- Scalability:
 - The amplitude encoding method scales well in terms of the qubit count but poorly in terms of the gate count, making it suitable for scenarios where qubit resources are more constrained than gate operations.
 - The divide-and-conquer encoding method becomes impractical for very large inputs as a result of the significant increase in both the qubit and gate counts.
- Practical applications:
 - For applications requiring minimal qubit usage, the amplitude encoding method might be preferred despite its high gate count.
 - For applications where both qubit and gate resources are considered, the divide-and-conquer approach might offer a balanced trade-off.

► Note 5.2. The qubit counts for the divide-and-conquer encoding approach are shown in terms of both data and ancillary qubits required to encode an input. This is because quantum hardware is limited in the number of qubits it offers, so both data and ancillary qubits should be considered. Nevertheless, for situations where the qubit count is not that important, the divide-and-conquer encoding technique is a clear pick, offering much more efficient gate usage. If, for any reason, only the data qubits are considered, then, for a given input vector, the divide-and-conquer encoding method utilizes the same number of data qubits as the amplitude encoding method. ◀

Now that the first table analysis is concluded, it is time to disclose the results for the second gate metric. The second metric transitions from unmodified to modified quantum circuits, where the modifications allow the circuits to be run on quantum computers.

5.1.2 Transpilation with no optimization

In this second comparison, transpilation is performed on the quantum circuits. However, no optimization is used during the transpilation process (as explained in the comprehensive description of the second gate count metric). The following table also contains the results for input vectors of lengths ranging from $2^0 = 1$ to $2^{11} = 2048$, with the exponent of 2^m being increased from 0 to 11, resulting in the following 12 table rows:

Input length	Qubits Basis	Qubits Angle	Qubits Amplitude	Qubits D-a-c	Gates Basis	Gates Angle	Gates Amplitude	Gates D-a-c
1	1	1	1	3	1	5	5	84
2	2	2	1	3	2	10	5	84
4	4	4	2	3	4	20	55	84
8	8	8	3	7	8	40	543	311
16	16	16	4	15	16	80	2415	834
32	32	32	5	31	32	160	6255	1949
64	64	64	6	63	64	320	18895	4248
128	128	128	7	127	128	640	54095	8915
256	256	256	8	255	256	1280	156367	18318
512	512	512	9	511	512	2560	424655	37193
1024	1024	1024	10	1023	1024	5120	1088719	75012
2048	2048	2048	11	2047	2048	10240	2671823	150719

■ **Figure 5.2** A table containing the number of gates and qubits used by each encoding method when producing a circuit that encodes input vectors of lengths ranging from 1 to 2048 (the lengths are powers of two) with circuit transpilation performed, where the transpilation process does not utilize any optimization.

This table reflects the results after the same circuits from the first table have been transpiled with no optimization to the native gate set of the IBM Heron processor. Transpilation does not alter qubit counts, as the circuits would change if the qubit count were to be altered. The only differences between the two tables are observed in the gate counts, which dramatically increase after transpilation to the IBM Heron gates. This increase highlights the additional complexity introduced by the need to decompose unsupported gates into sequences of native gates. Comparing this table to the previous table, which showed the counts for non-transpiled circuits (5.1), the most significant insights into the impact of transpilation on the gate counts are outlined below:

1. Basis encoding method:
 - Non-transpiled: the gate counts grow linearly, equal to the input length.
 - Transpiled: the gate counts are linearly proportional to the input length.

2. Angle encoding method:

- Non-transpiled: the gate counts also grow linearly.
- Transpiled: the gate counts still grow linearly but with a steeper slope. For example, the gate count for the input length of 2048 increases from 2048 (non-transpiled) to 10240 (transpiled).

3. Amplitude encoding method:

- Non-transpiled: the gate counts increase exponentially, reaching 34819 for the input length of 2048.
- Transpiled: the gate counts escalate dramatically to 2671823 for the same final input length, indicating a substantial increase in computational complexity due to transpilation.

4. Divide-and-conquer encoding method:

- Non-transpiled: the gate counts show a steady linear increase, reaching 4083 for an input length of 2048.
- Transpiled: the gate counts surge to 150719, reflecting the added complexity of decomposing gates into the IBM Heron's natively supported gate set.

The impacts the transpilation process brings can be summarized in the following way:

- Increased gate complexity:
 - The most striking impact of transpilation is the significant increase in the gate counts across all encoding methods (except for the basis encoding method, where the gates X do not have to be decomposed further). This increase is due to the decomposition of unsupported gates into sequences of native gates supported by the IBM Heron processor.
 - The gate counts increase exactly five times in the angle encoding method. They increase up to a hundred times in the amplitude and divide-and-conquer encoding methods, more than doubling with each subsequent input vector length (after the input length of 4).
- Method-specific effects:
 - The amplitude encoding method experiences the most drastic increase in the gate count after transpilation, highlighting the high computational complexity of this method when mapped to the native gate set of a quantum processor.
 - The divide-and-conquer encoding method also shows a substantial increase in gate count, but less extreme compared to the amplitude encoding method, making it relatively more efficient post-transpilation.

- Practical considerations:
 - The substantial increase in gate counts due to transpilation underscores the importance of considering hardware-specific constraints when designing and evaluating quantum circuits. Theoretical efficiency must be balanced with practical feasibility on actual quantum hardware.
 - The choice of encoding method should factor in the potential overhead introduced by transpilation, particularly when requiring high-quality state preparation (since a high gate count can lead to significant errors).

This detailed comparison between non-transpiled and transpiled circuits showcase the critical role of the transpilation process in quantum computing. While the non-transpiled results provide insights into the theoretical gate complexity of encoding methods, the transpiled results reflect the practical realities of running these methods on real quantum hardware. The significant increase in the gate counts after transpilation highlights the need for careful consideration of hardware constraints and the potential trade-offs between theoretical and practical performance. The good news is that this extreme increase in the gate counts is not unavoidable, as seen in the third table below.

5.1.3 Transpilation with best optimization

The third and final table is presented below:

Input length	Qubits Basis	Qubits Angle	Qubits Amplitude	Qubits D-a-c	Gates Basis	Gates Angle	Gates Amplitude	Gates D-a-c
1	1	1	1	3	1	2	0	33
2	2	2	1	3	2	6	3	37
4	4	4	2	3	4	16	6	38
8	8	8	3	7	8	32	225	139
16	16	16	4	15	16	64	975	370
32	32	32	5	31	32	128	2686	863
64	64	64	6	63	64	256	8767	1879
128	128	128	7	127	128	512	25042	3940
256	256	256	8	255	256	1024	69877	8094
512	512	512	9	511	512	2048	184246	16431
1024	1024	1024	10	1023	1024	4096	461961	33135
2048	2048	2048	11	2047	2048	8192	1116202	66573

■ **Figure 5.3** A table containing the number of gates and qubits used by each encoding method when producing a circuit that encodes input vectors of lengths ranging from 1 to 2048 (the lengths are powers of two) with circuit transpilation performed, where the transpilation process utilizes the highest possible optimization level.

In this final comparison, the quantum circuits undergo the transpilation process using the highest optimization level available (the level 3, as detailed in the thorough overview of the third gate count metric). The same input vector lengths are used in this table, too, producing the 12 table rows seen above. Unlike the second table (5.2), which presented the gate counts for circuits transpiled with no optimization, this third table shows the results after applying the highest level of optimization during the transpilation process. Summarizing the third table's results reveals the significant impact of the chosen transpilation level on the gate counts:

1. Basis encoding method:

- No changes are introduced in this case either, due to the fact that the IBM Heron's set of natively supported gates contains the gate X , meaning the transpilation process omits these gates.

2. Angle encoding method:

- The gate counts are reduced by one-fifth compared to no when no optimization is used, meaning a saving of 20% of all gates (starting from the third input vector). But the number is still four times larger than when no transpilation is performed.

3. Amplitude encoding method:

- The last gate count is reduced from 2671823 to 1116202, which is a reduction of more than 58%. This is a significant decrease compared to when no optimization is performed, but still much higher than in the non-transpiled scenario.

4. Divide-and-conquer encoding method:

- The gate count for the final input vector length is decreased from 150719 to 66573, which is a reduction of more than 55%, similar to the reduction in the amplitude encoding method.

Several important observations arise from the preceding commentary, signifying the following broader transpilation aspects:

■ Impact of transpilation without optimization:

- The gate counts increase dramatically when circuits are transpiled without any optimization. This rise is due to the decomposition of unsupported gates into sequences of native gates, leading to a significant increase in the number of operations required.
- The absence of optimization results in higher gate counts, reflecting the raw complexity of implementing the circuits directly on the IBM Heron processor.

- Impact of high-level optimization:
 - High-level optimization during transpilation significantly reduces the gate counts compared to no optimization. This reduction is achieved through advanced techniques that streamline the circuit, remove redundancies, and optimize gate usage.
 - Despite the reduction, the gate counts for optimized circuits remain higher than the non-transpiled scenarios, indicating that some complexity is inherent to the hardware constraints and cannot be fully mitigated.
- Comparative analysis:
 - The angle encoding method benefits from optimization, showing linear growth in gate counts that are reduced compared to no optimization.
 - The amplitude encoding method shows the highest gate counts in all scenarios, reflecting its computational intensity. However, optimization substantially lowers the gate count, making it more feasible for practical applications.
 - A similar optimization boost as in the amplitude encoding method is brought to the divide-and-conquer encoding approach, making it more efficient post-transpilation.
- Practical considerations:
 - Optimization during the transpilation process plays a crucial role in managing the gate counts, making high-level optimization essential for practical quantum computing applications.
 - The choice of optimization level impacts the feasibility and performance of quantum circuits on quantum hardware. This aspect is not explored in this section, as it would require quantum computers using large numbers of qubits and gates. However, it is a point of interest because the chosen transpilation algorithm may have a significant impact on quantum algorithms that later process the encoded data.
 - When transpiling with the highest optimization level, it took significantly longer to complete the process compared to transpiling with no optimization level. This suggests that the highest level of optimization incurs significant overhead when applied to large circuits.

This comprehensive comparison highlights the effects of transpilation and its optimization in quantum computing. While non-transpiled results provide theoretical insights, transpiled results with no and high optimization levels reflect the practical realities of implementing quantum circuits on real hardware. High-level optimization mitigates the complexity introduced by transpilation to some extent, making it a vital step for efficient quantum computing. The whole comparative analysis section is now finished, and the stage is set to proceed to other sections, where the encoding methods are tested individually.

5.2 Basis encoding method test

In this section and in the next three sections, the corresponding encoding methods are tested on an individual basis. Each test is performed by encoding a pseudo-randomly generated input vector using the corresponding encoding method and then evaluating the created quantum circuit. All test results presented in these sections originate from the Jupyter Notebooks located in the repository (4). Qiskit was used to perform the tests. One of the objectives of this thesis is to conduct experiments on real quantum computers. This has been achieved by using the Qiskit library to run quantum circuits on IBM Quantum devices. The findings obtained from these experiments are presented in these sections. However, due to the number of pages of this thesis being already large at this point, the results are not commented on in depth. The mentioned Jupyter Notebook explain in detail the exact rationale behind the performed operations. Here, only the raw results are presented without commentary.

Each one of these last four sections has the same structure. The purpose of this structure is to present the experiment results in an effective and brief way. This is achieved using histograms (the use of histograms is explained Visualization of probabilities (2.13)). The exact structure of each section is described as follows:

- First, the input vector used during the encoding process is presented. Each input vector was natively generated in Python, thus resulting in pseudo-random input vector. Data points generated in Python are referred to as *arrays*. In these sections, the term *array* is also used to describe the input vectors to better align with Python programming conventions (denoted using the squared brackets “[]”).
- A diagram is shown of the quantum circuit created by encoding the generated array using the corresponding implemented encoding method.
- Measurements of the appropriate qubits were performed using the Qiskit’s default simulator, and the simulation results are shown in the form of histograms.
- A histogram is shown, where its bars compare these three values:
 - The simulation results.
 - The expected simulation results in the form of probabilities, where the probabilities are calculated from the generated array. The elements of the generated array can be used to calculate the desired ideal encoding outcomes. For example, in the amplitude encoding method, the resulting state vector elements obtained by encoding the input array should be equal to the generated input array elements. However, due to the fact

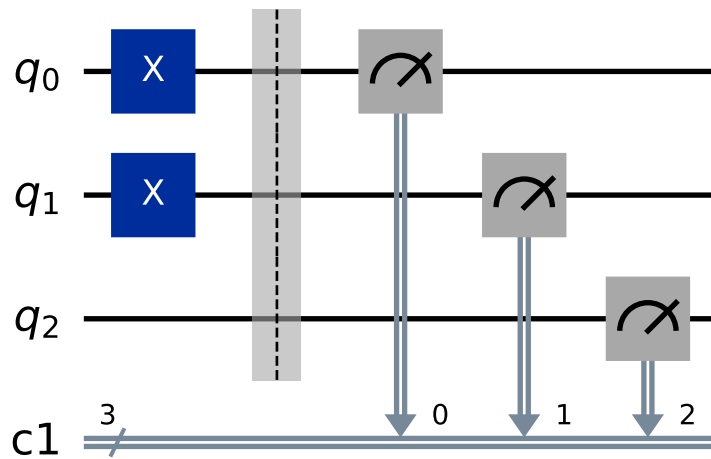
that in the divide-and-conquer encoding method only probabilities are used to define the desired encoding outcomes, only the probabilities are shown in the histogram. For most encoding methods, this just means that the squared input array's elements are displayed.

- The probabilities calculated from the resulting state vector of the created quantum circuit's quantum state. In the case of the divide-and-conquer encoding method, the probabilities are retrieved using the density matrix.
 - The results of the real quantum experiments are shown in a histogram. The measurement outcomes are plotted along with the simulation outcomes and the desired probabilities calculated from the input array.
 - The calculated fidelity and trace distance values are shown. The fidelity (defined in 2.6) is calculated using two pure quantum states. There are no quantum states available to use straight away, they must be calculated manually. One of these pure states is derived from the generated input array (most often, the array's elements are just squared). The other pure state is obtained by approximation, where the probability distribution of quantum states is inferred from the measurement results (measurement results obtained by running the circuit on the chosen quantum computer) by normalizing the measurement results. This process does not produce a real quantum state; it is just an approximation of what the quantum state (in the form of a probability vector) could have been right before the measurement on the quantum computer. The trace distance is calculated on the basis of the fidelity value.
- Note 5.3. These sections are intentionally kept very brief and concise (to prevent the thesis from being excessively lengthy), so only the raw testing results are presented without much commentary, explanation, and mathematical result confirmation. These tests serve as a proof-of-concept, indicating that the implementation can be used to encode inputs on real quantum computers. The repository (4) contains Jupyter Notebooks that explain the testing process in great depth. These sections just contain the results presented in those Notebooks. ◀

In this specific example, the generated array ended up being:

[1, 1, 0].

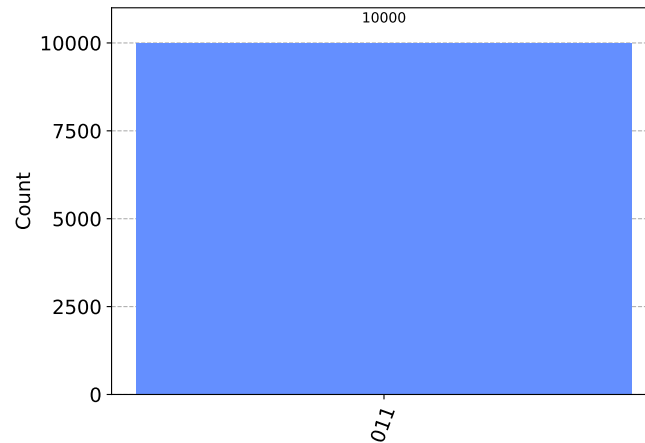
If this input is encoded using the implemented algorithm, this is the resulting quantum circuit:



■ **Figure 5.4** An illustration of a quantum circuit created by using the implemented basis encoding method to encode the following input array: [1, 1, 0].

This corresponds fully to the desired quantum circuit, indicating this input array is encoded correctly.

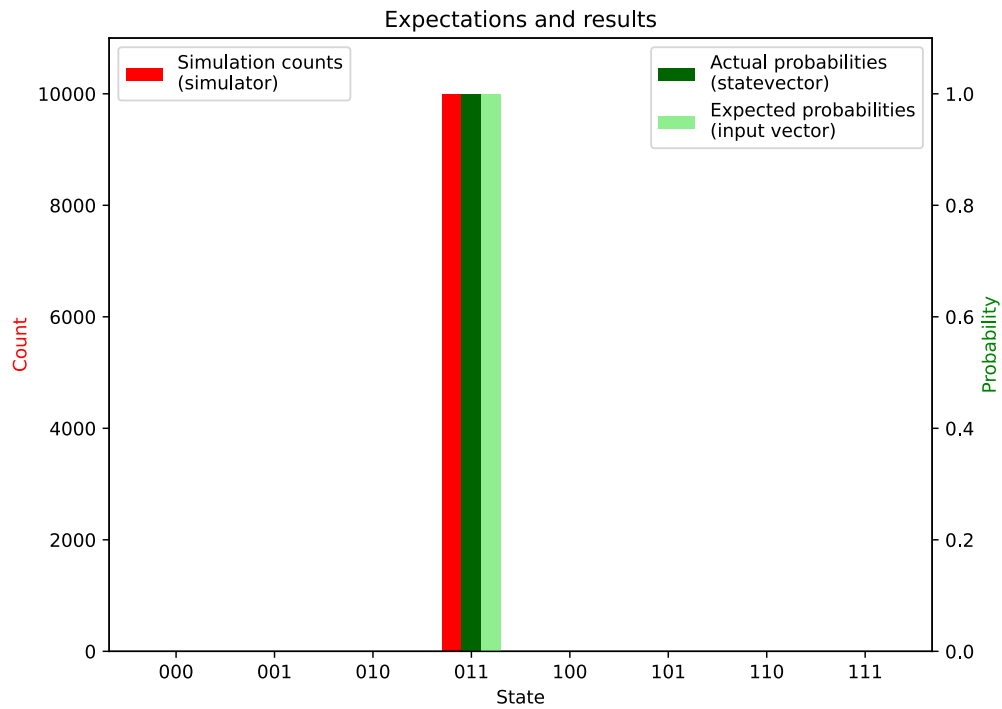
A basic simulation was performed where all qubits were measured using the Qiskit's simulator. The simulation allows for a quick test of the created quantum circuit. The following results were obtained:



■ **Figure 5.5** A histogram containing the measurement results of a simulation, where the simulation was performed by measuring the appropriate qubits of a circuit created by using the implemented basis encoding method to encode the following input array: [1, 1, 0].

Due to the nature of the basis encoding method, the simulation results should always consist of just one state, where 100% of the measurements correspond to the state defined by the input array elements. This holds true in this case.

Next, the simulation results are shown in a histogram, along with the probability distributions calculated from the quantum state vector and the expected probabilities calculated from the input array:



■ **Figure 5.6** A histogram comparing the measurement outcomes of a simulation with the distribution of probabilities of the created quantum state, as well as the expected probabilities calculated from an input array. The values were obtained from a circuit created by using the implemented basis encoding method to encode the following input array: [1, 1, 0].

As can be seen in this histogram, the simulation results and the state vector probabilities fully correspond to the desired probabilities derived from the generated input array.

The final chart shown displays the measurement results obtained by running this quantum circuit on an actual quantum computer. In this specific case, the *ibm_osaka* quantum computer was used to execute this circuit. It should be noted that slight differences in the measurement results are to be expected due to the noise and errors inherent to quantum computers. Below is a histogram that compares the actual results retrieved from the quantum computer with the numbers already shown in the histogram above:

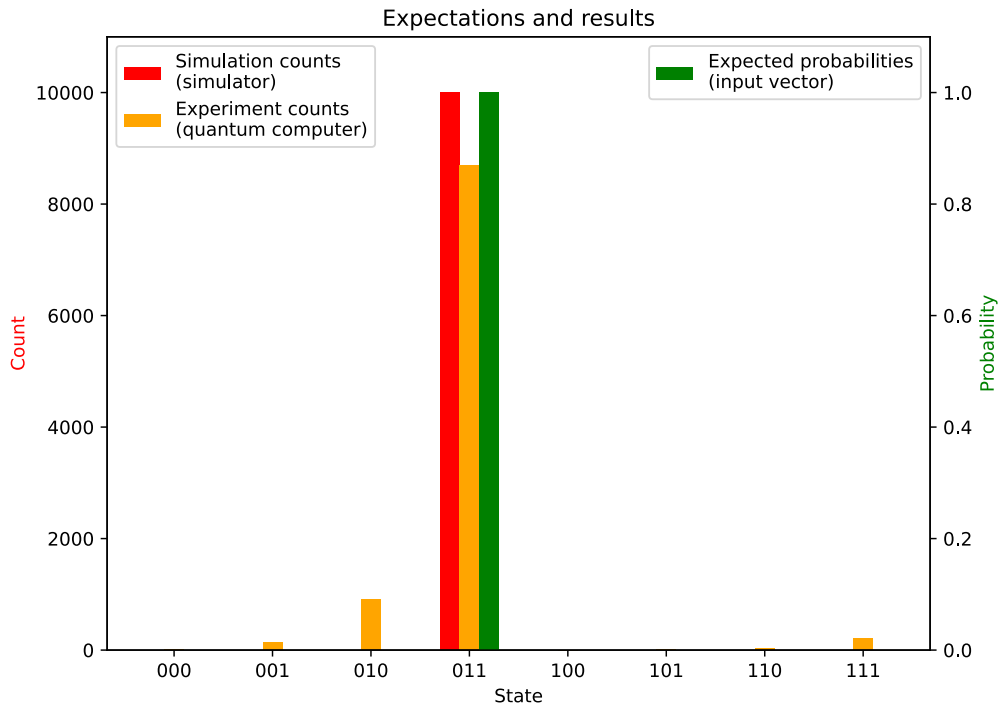


Figure 5.7 A histogram comparing the measurement outcomes of a simulation with the measurement results obtained by running a quantum circuit on the quantum computer *ibm_osaka* and the expected distribution of probabilities calculated from an input array. The values were obtained from a circuit created by using the implemented basis encoding method to encode the following input array: $[1, 1, 0]$.

As seen in the histogram, the quantum computer produced slightly different results compared to the simulation, where the state 010 ended up being relatively numerous in the outcomes (along with the states 001 and 111 to a smaller extent). This illustrates the fact that quantum computers are affected by various phenomena that decrease the quality of the state.

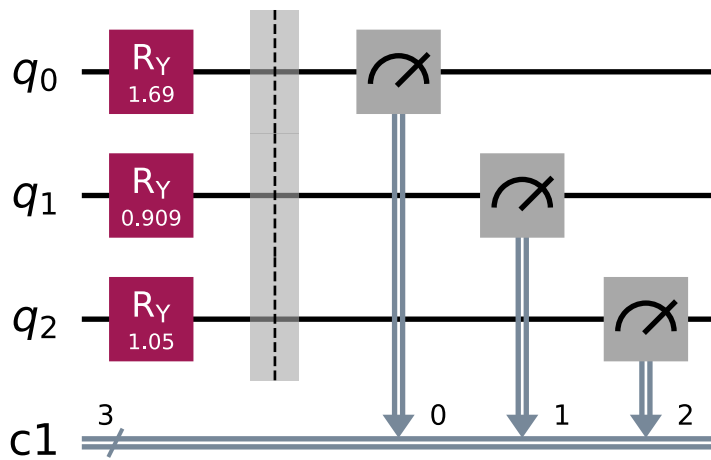
The resulting fidelity between the states is $\simeq 0.87$. This fidelity indicates that the states are quite similar, although not perfectly identical. This may be considered an acceptable result, considering the presence of noise and errors in quantum computations. The trace distance is $\simeq 0.36$, suggesting that there is some distinguishability between the two states, but they are still reasonably close. This reinforces the fidelity result, indicating that the encoding method performs well on the quantum computer.

5.3 Angle encoding method test

This example demonstrates the implementation of the angle encoding method by encoding the following randomly generated input array:

$$[0.74651424, 0.43896263, 0.5000283].$$

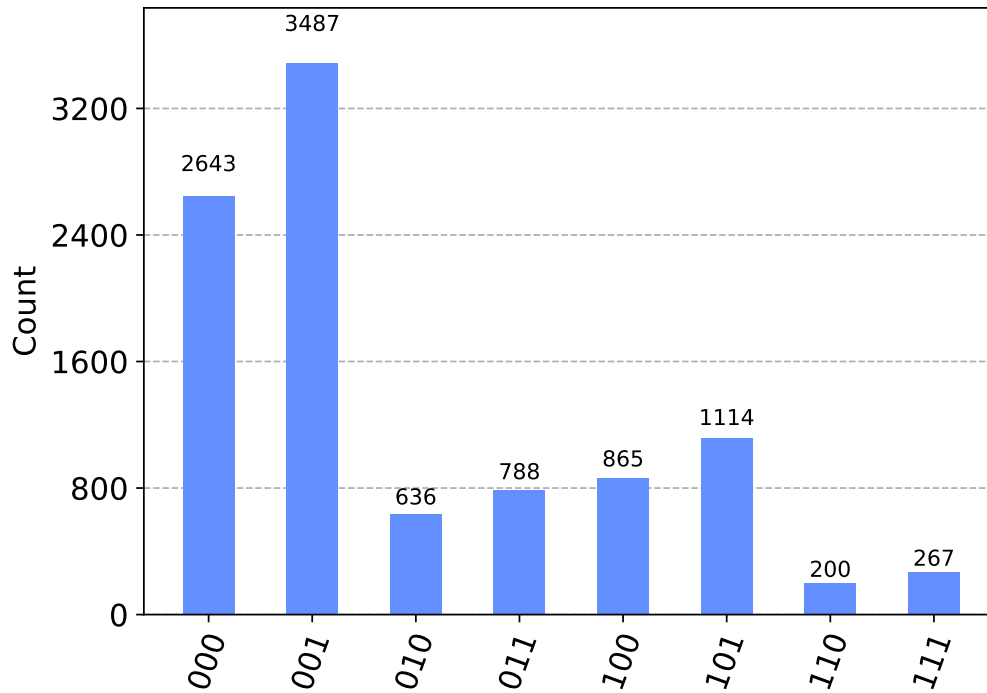
Using the implemented algorithm to encode this input yields the following quantum circuit:



■ **Figure 5.8** An illustration of a quantum circuit created by using the implemented angle encoding method to encode the following input array: $[0.74651424, 0.43896263, 0.5000283]$.

This circuit also corresponds to the desired quantum circuit, meaning that the implementation correctly encoded this input array.

A basic simulation was conducted, measuring all qubits through the simulator provided by Qiskit. The subsequent results were as follows:



■ **Figure 5.9** A histogram containing the measurement results of a simulation, where the simulation was performed by measuring the appropriate qubits of a circuit created by using the implemented angle encoding method to encode the following input array: $[0.74651424, 0.43896263, 0.5000283]$.

The outcomes of the simulation are displayed in a histogram, accompanied by the probability distributions derived from the quantum state vector and the desired probabilities computed from the input array:

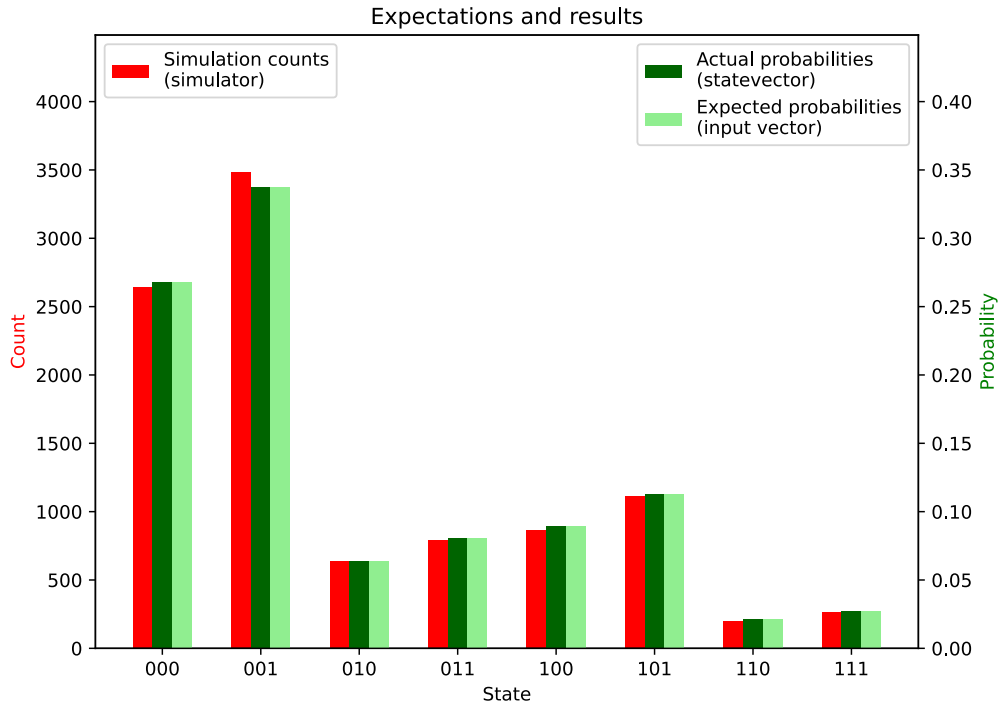


Figure 5.10 A histogram comparing the measurement results of a simulation with the distribution of probabilities of the created quantum state, as well as the expected probabilities calculated from an input array. The values were obtained from a quantum circuit created by using the implemented angle encoding method to encode the following input array: $[0.74651424, 0.43896263, 0.5000283]$.

The histogram clearly illustrates that the simulation outcomes and the state vector probabilities are in almost complete agreement with the expected probabilities obtained from the input array produced. The discrepancies come from the fact that the simulator introduces some uncertainty inherent to quantum computers.

The last diagram presented illustrates the measurement outcomes obtained by executing this quantum circuit on a real quantum computer. The quantum computer *ibm_osaka* was used in this example as well. Here is a histogram that displays the actual outcomes obtained from the quantum computer alongside the data previously depicted in the above histogram:



■ **Figure 5.11** A histogram comparing the measurement outcomes of a simulation with the measurement results obtained by running a quantum circuit on the quantum computer *ibm_osaka* and the expected distribution of probabilities calculated from an input array. The values were obtained from a circuit created by using the implemented angle encoding method to encode the following input array: $[0.74651424, 0.43896263, 0.5000283]$.

The histogram illustrates that the measurement outcomes from the quantum computer varied to some extent from those of the simulation. However, for the most part, they correspond, and this result can be taken as an empirical validation that the implemented encoding method worked correctly on this specific input array.

The resulting fidelity between the states is $\simeq 0.99$. This fidelity indicates that the states are extremely similar, almost identical. This is an excellent result, suggesting minimal noise and errors in the quantum computations. The trace distance is $\simeq 0.07$, suggesting that there is very little distinguishability between the two states, and that they are very close to each other. This reinforces the fidelity result, indicating that the encoding method performs exceptionally well on the quantum computer.

5.4 Amplitude encoding method test

This sample illustrates how the amplitude encoding technique is applied to encode the following input array¹:

$$[0.57221017, -0.21990234, -0.19589899, -0.18398457, \\ -0.4941013, 0.13438299, -0.27722379, -0.46145838].$$

Applying the implemented algorithm on this input results in the subsequent quantum circuit:

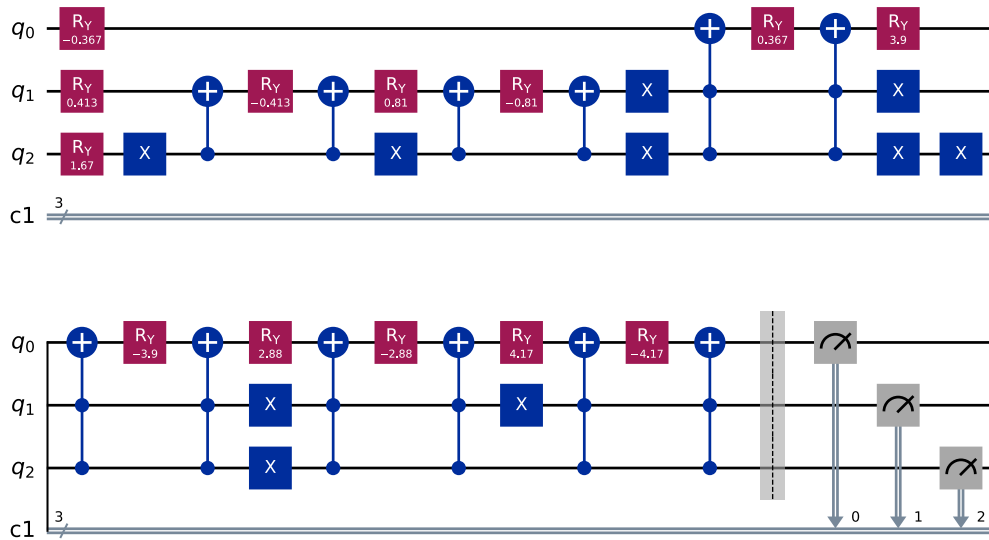
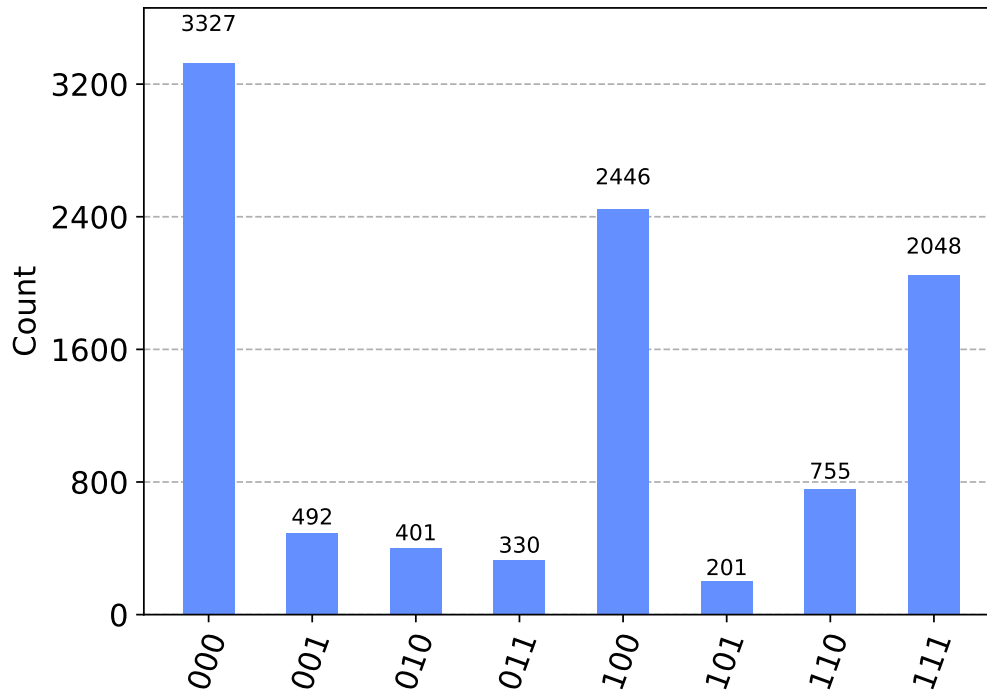


Figure 5.12 An illustration of a quantum circuit created by using the implemented amplitude encoding method to encode the following input array:
 $[0.57221017, -0.21990234, -0.19589899, -0.18398457, \\ -0.4941013, 0.13438299, -0.27722379, -0.46145838].$

This circuit also matches the intended quantum circuit, indicating that the implementation accurately encoded this input array. The controlled gates R_Y are decomposed into smaller gates to facilitate implementation using Qiskit.

¹In this case, a longer input was generated because this encoding method uses logarithmic amount of qubits compared to the input length to encode input vectors.

A simulation was performed, measuring all qubits using the simulator supplied by Qiskit. The ensuing outcomes were as follows:



■ **Figure 5.13** A histogram containing the measurement results of a simulation, where the simulation was performed by measuring the appropriate qubits of a quantum circuit created by using the implemented amplitude encoding method to encode the following input array:
[0.57221017, -0.21990234, -0.19589899, -0.18398457,
-0.4941013, 0.13438299, -0.27722379, -0.46145838].

The results of the simulation are presented in a histogram, along with the probability distributions obtained from the quantum state vector and the target probabilities calculated from the input array:

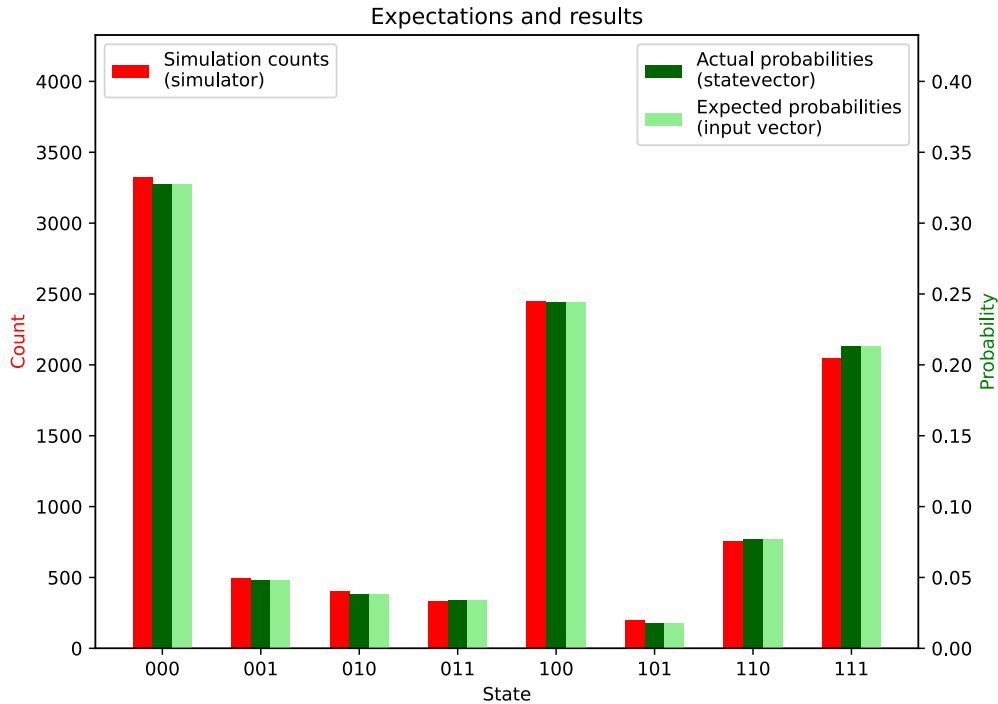


Figure 5.14 A histogram comparing the measurement results of a simulation with the distribution of probabilities of the created quantum state, as well as the expected probabilities calculated from an input array. The values were obtained from a quantum circuit created by using the implemented amplitude encoding method to encode the following input array:
 $[0.57221017, -0.21990234, -0.19589899, -0.18398457, -0.4941013, 0.13438299, -0.27722379, -0.46145838]$.

The histogram effectively demonstrates that the results of the simulation and the probabilities of the state vector largely conform to the anticipated probabilities derived from the generated input array. The variances are due to the uncertainty introduced by the simulator, which is typical in quantum computing environments. The variances may be a little higher in this case due to the fact that this encoding method uses more gates.

The final diagram shown in this section represents the results of measurements from running this quantum circuit on an actual quantum computer. This time, the most advanced quantum computer *ibm_torino* was employed for these executions, as this computer uses better error-correction techniques which is crucial for circuits that contain many gates. Below is a histogram that shows the results from this quantum computer compared with the data shown in the previous histogram:

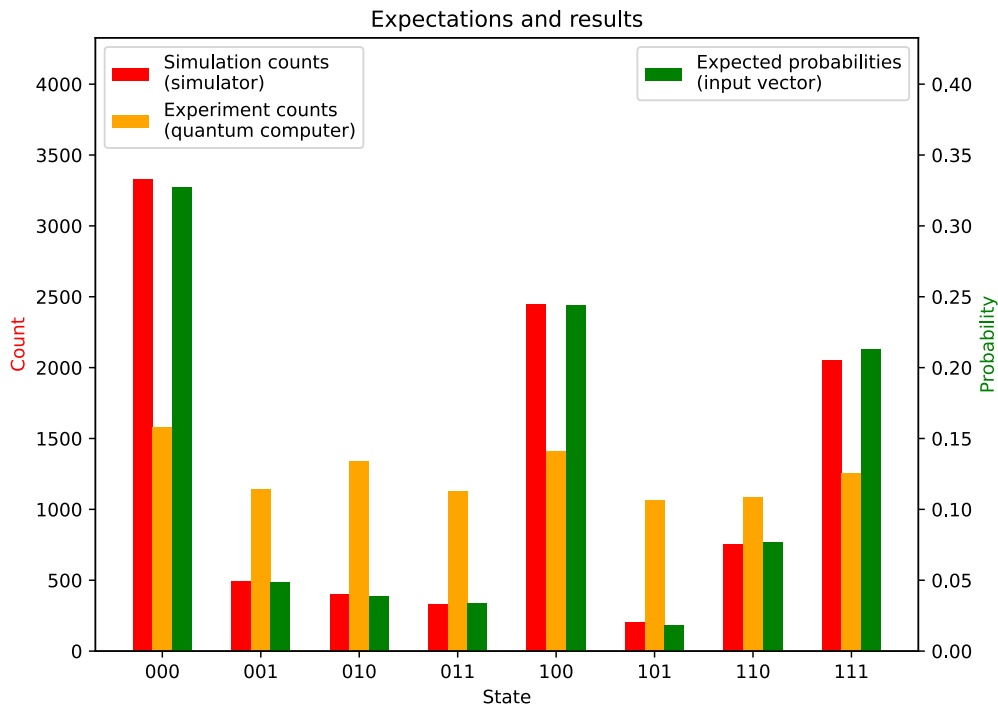


Figure 5.15 A histogram comparing the measurement outcomes of a simulation with the measurement results obtained by running a quantum circuit on the quantum computer *ibm_torino* and the expected distribution of probabilities calculated from an input array. The values were obtained from a circuit created by using the implemented amplitude encoding method to encode the following input array: $[0.57221017, -0.21990234, -0.19589899, -0.18398457, -0.4941013, 0.13438299, -0.27722379, -0.46145838]$.

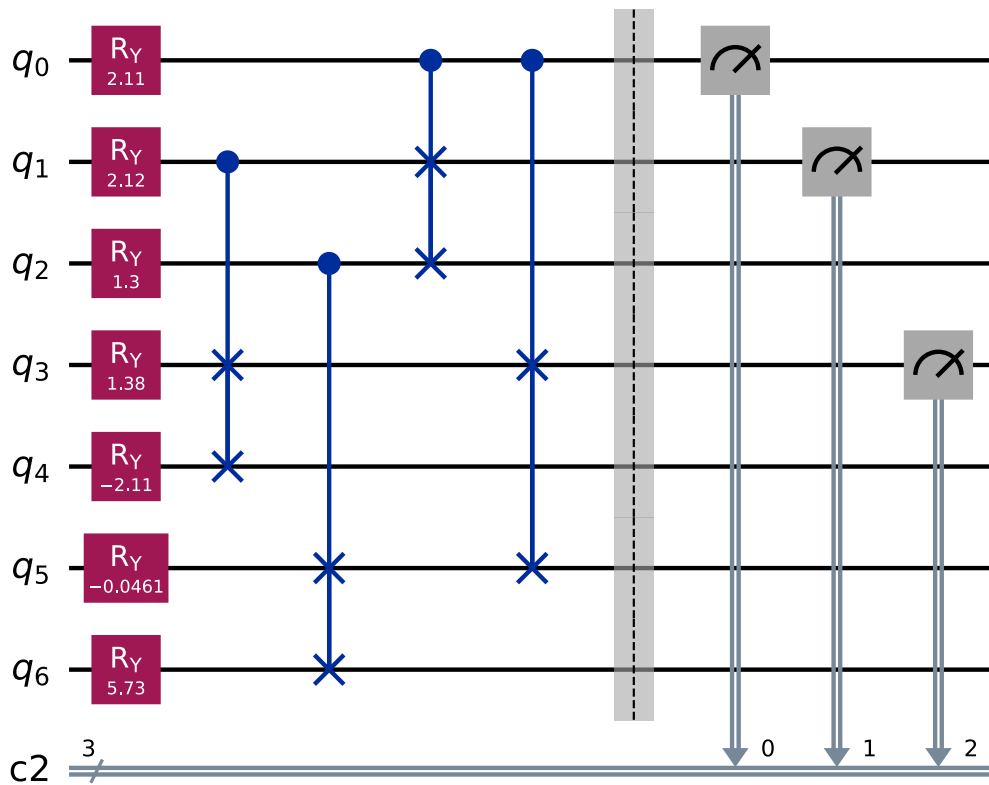
This time, it is clear that the measurement outcomes are more uniform, which indicates a state preparation of lower quality. This is due to the sheer number of gates employed by this encoding method. Each gate used in real quantum computers can introduce errors, and if there are many of them, the errors add up. The resulting fidelity between the states is $\simeq 0.84$, indicating substantial similarity but not perfect identity. This number may be viewed as a positive outcome considering the noise and errors in quantum computations. The trace distance is $\simeq 0.39$, showing moderate distinguishability between the states. Overall, the encoding method is fairly robust, though there is room for improvement to achieve higher precision and lower distinguishability.

5.5 Divide-and-conquer encoding method test

This example demonstrates the application of the divide-and-conquer encoding method to encode the input array described as:

$$[0.18607576, 0.15434713, 0.21177548, -0.37407953, \\ 0.69254782, -0.01595983, -0.50636497, 0.14312844].$$

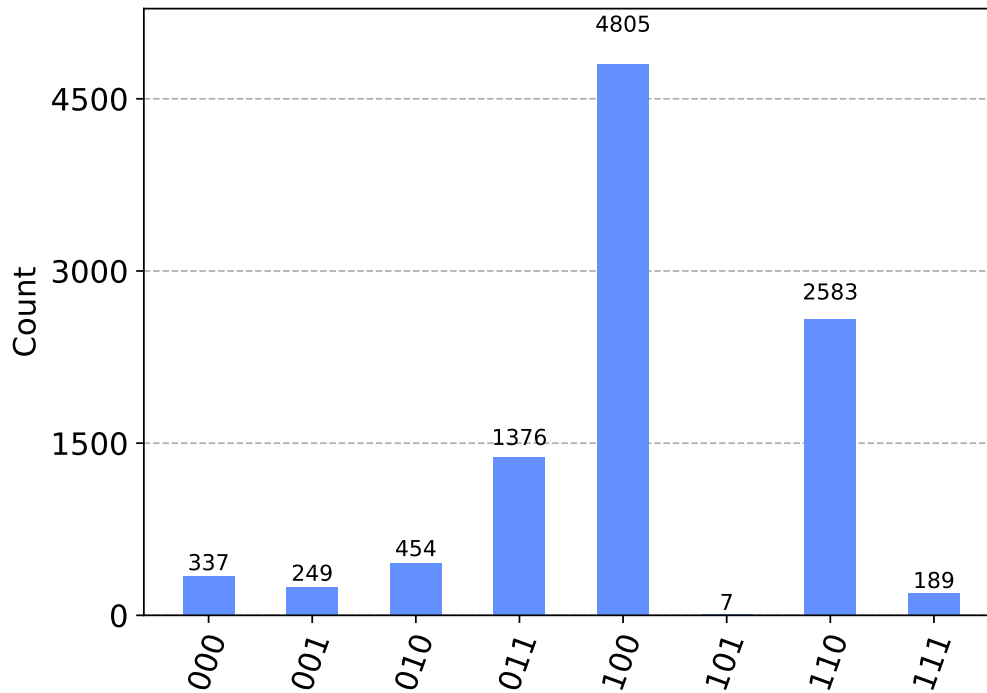
Using the implemented algorithm with this input produces the following quantum circuit:



■ **Figure 5.16** An illustration of a quantum circuit created by using the implemented divide-and-conquer encoding method to encode the following input array:
 $[0.18607576, 0.15434713, 0.21177548, -0.37407953, \\ 0.69254782, -0.01595983, -0.50636497, 0.14312844].$

This final circuit corresponds to the desired quantum circuit as well, demonstrating that the implementation successfully encodes this input array.

A simulation was conducted where all qubits were measured using the provided Qiskit simulator. The resulting data were as follows:



■ **Figure 5.17** A histogram containing the measurement results of a simulation, where the simulation was performed by measuring the appropriate qubits of a quantum circuit created by using the implemented divide-and-conquer encoding method to encode the following input array:
[0.18607576, 0.15434713, 0.21177548, -0.37407953,
0.69254782, -0.01595983, -0.50636497, 0.14312844].

The simulation outcomes are depicted in a histogram, accompanied by the probability distributions derived from the quantum state vector and the target probabilities computed based on the input array:

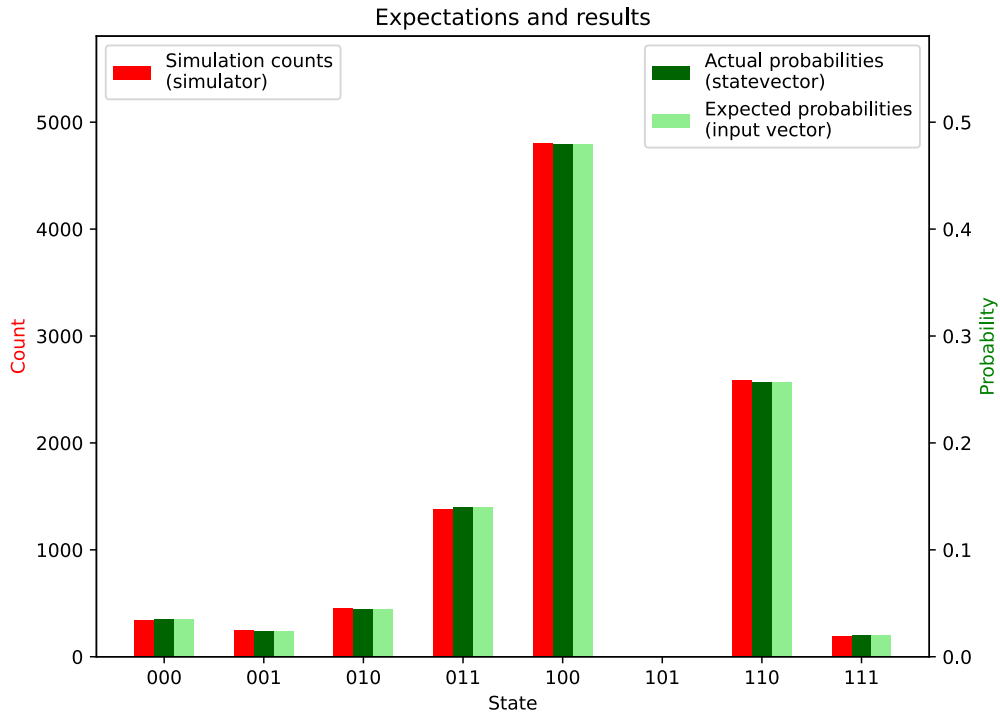


Figure 5.18 A histogram comparing the measurement results of a simulation with the distribution of probabilities of the created quantum state, as well as the expected probabilities calculated from an input array. The values were obtained from a quantum circuit created by using the implemented divide-and-conquer encoding method to encode the following input array:

[0.18607576, 0.15434713, 0.21177548, -0.37407953,
0.69254782, -0.01595983, -0.50636497, 0.14312844].

The simulation outcomes and the state vector probabilities almost perfectly copy the expected probabilities obtained from the generated input array.

The concluding figure illustrates the measurement outcomes obtained by running this quantum circuit on the most advanced device *ibm_torino* that was utilized in this experiment as well. Following is a histogram that displays the actual outcomes from the quantum computer alongside the data presented in the earlier histogram:

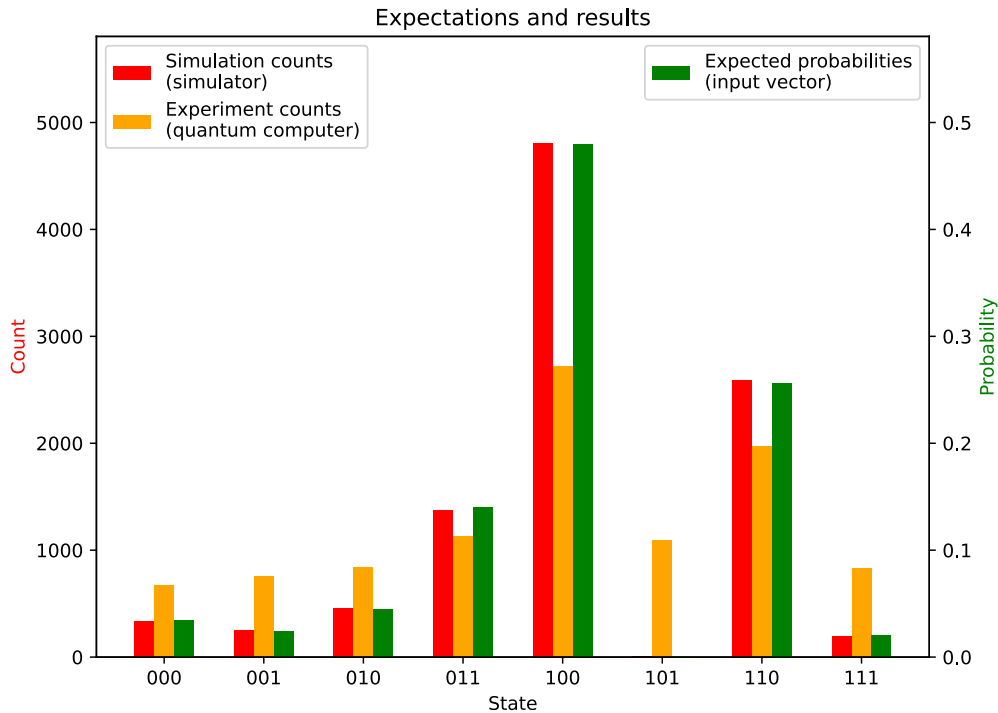


Figure 5.19 A histogram comparing the measurement outcomes of a simulation with the measurement results obtained by running a quantum circuit on the quantum computer *ibm_torino* and the expected distribution of probabilities calculated from an input array. The values were obtained from a circuit created by using the implemented divide-and-conquer encoding method to encode the following input array: $[0.18607576, 0.15434713, 0.21177548, -0.37407953, 0.69254782, -0.01595983, -0.50636497, 0.14312844]$.

The histogram shows that the measurement outcomes copy the desired probabilities to a significant extent. Even though this encoding method uses fewer gates than the previous one, the worse resulting fidelity of $\simeq 0.84$ may indicate that this encoding method is not much better. However, this claim cannot be made after conducting just a single simple experiment. The trace distance is $\simeq 0.39$, showing moderate distinguishability between the states. However, overall, the encoding method is still fairly robust.

Conclusion

This thesis has conducted an extensive study into the problem of representing classical data in a form that quantum computers can process by thoroughly examining four data encoding approaches—the basis, angle, amplitude, and divide-and-conquer encoding methods. Two prominent achievements of this thesis are the detailed exploration of the theoretical underpinnings of these encoding methods and their practical implementation using the Qiskit library.

The thesis successfully provided a rigorous explanation of these different encoding methods, enabling informed decisions on which encoding method to use in various applications based on their elaborated characteristics. This was achieved through a series of detailed analyses and empirical tests. The thesis highlighted the traits and attributes of each encoding method, thereby demonstrating that the choice of encoding method constrains and impacts the subsequent data processing algorithms that can be used. While the complexities of the encoding strategies were described from a theoretical standpoint, empirical tests were conducted to explore qubit and gate complexities in practice. During this exploration, the methods were examined and compared with each other, revealing that the transpilation process in practical applications significantly increases the gate counts required to encode input vectors. This finding underscores the necessity of considering not only encoding method complexities from the theoretical perspective but also practical hardware limitations.

While the thesis did not introduce entirely new encoding strategies or improve the four discussed encoding methods in any significant manner, it successfully implemented these encoding techniques in a way that enables their use for encoding appropriate input vectors. The implementation was developed using the Qiskit library. The object-oriented nature of the implementation facilitates future expansion, allowing additional encoding methods to be incorporated by simply adding their respective algorithms without the need to work extensively with certain aspects of the Qiskit library. Simple tests were conducted using real quantum hardware to validate the implemented encoding methods and provide practical insights into their application.

The thesis achieved all the objectives outlined in the introduction by covering specific essential aspects of quantum computing, presenting and explaining the four encoding strategies, implementing them using the Qiskit library, and validating them via basic experiments on real quantum devices.

Several potential areas for future research have emerged from this study. Firstly, there is an opportunity to improve the divide-and-conquer encoding method by exploring the feasibility of disentangling ancillary qubits or leveraging the entanglement of qubits to manipulate already encoded data advantageously. Secondly, combining the beneficial traits of the amplitude and divide-and-conquer encoding methods presents another promising research direction. Exploring the integration of the strengths of both approaches might lead to the development of a hybrid method that could potentially combine efficient qubit usage with manageable circuit depth.

In conclusion, this thesis has provided a detailed exploration of the aforementioned quantum state preparation methods, offering valuable insights into their practical applications and implications. The findings underscore the significance of data encoding method selection in quantum computing and highlight several promising avenues for future research that could potentially advance the current state of these encoding methods.

Bibliography

1. LI, Chunyuan; GAN, Zhe; YANG, Zhengyuan; YANG, Jianwei; LI, Linjie; WANG, Lijuan; GAO, Jianfeng. *Multimodal Foundation Models: From Specialists to General-Purpose Assistants*. 2023. Available from DOI: 10.48550/arXiv.2309.10020.
2. BOMMASANI, Rishi. *AI Spring? Four Takeaways from Major Releases in Foundation Models* [online]. Stanford HAI, 2023. [visited on 2024-04-27]. Available from: <https://hai.stanford.edu/news/ai-spring-four-takeaways-major-releases-foundation-models>.
3. ZHAI, Yongbiao; FENG, Zihao; ZHOU, Ye; HAN, Su-Ting. Energy-efficient transistors: suppressing the subthreshold swing below the physical limit. *Materials Horizons*. The Royal Society of Chemistry, 2021, vol. 8, pp. 1601–1617. Available from DOI: 10.1039/D0MH02029J.
4. PATEL, Pratyush; CHOUKSE, Esha; ZHANG, Chaojie; GOIRI, Íñigo; SHAH, Aashaka; MALEKI, Saeed; BIANCHINI, Ricardo. *Splitwise: Efficient generative LLM inference using phase splitting*. 2023. Available from DOI: 10.48550/arXiv.2311.18677.
5. LEE, Timothy B. The Real Research Behind the Wild Rumors About OpenAI’s Q Project. *Ars Technica* [online]. 2023 [visited on 2024-04-28]. Available from: <https://arstechnica.com/ai/2023/12/the-real-research-behind-the-wild-rumors-about-openais-q-project/>.
6. LYNCH, Shana. *AI Index: State of AI in 13 Charts* [online]. Stanford HAI, 2024. [visited on 2024-04-27]. Available from: <https://hai.stanford.edu/news/ai-index-state-ai-13-charts>.
7. METZ, Cade; WEISE, Karen; ISAAC, Mike. AI Chips: Why Big Tech Companies Are Racing to Develop Their Own. *The New York Times* [online]. 2024 [visited on 2024-04-27]. Available from: <https://www.nytimes.com/2024/01/29/technology/ai-chips-nvidia-amazon-google-microsoft-meta.html>.

8. KHAN, Hassan N.; HOUNSHELL, David A.; FUCHS, Erica R. H. Science and research policy at the end of Moore's law. *Nature Electronics*. 2018, vol. 1. ISSN 2520-1131. Available from DOI: 10.1038/s41928-017-0005-9.
9. CBRE RESEARCH. *Intelligent Investment: Global Data Center Trends 2023* [online]. CBRE, 2023-07. [visited on 2024-05-03]. Tech. rep. Available from: <https://www.cbre.com/insights/reports/global-data-center-trends-2023>.
10. BAJWA, Arsheeya; SIMAO, Paul; GREGORIO, David. Microsoft, OpenAI plan \$100 billion data-center project, media report says. *Reuters* [online]. 2024 [visited on 2024-05-03]. Available from: <https://www.reuters.com/technology/microsoft-openai-planning-100-billion-data-center-project-information-reports-2024-03-29/>.
11. WEISE, Karen. In Race to Build A.I., Tech Plans a Big Plumbing Upgrade. *The New York Times* [online]. 2024 [visited on 2024-05-03]. Available from: <https://www.nytimes.com/2024/04/27/technology/ai-big-tech-spending.html>.
12. BERREBY, David. As Use of A.I. Soars, So Does the Energy and Water It Requires. *Yale Environment 360* [online]. Yale School of the Environment, 2024 [visited on 2024-04-28]. Available from: <https://e360.yale.edu/features/artificial-intelligence-climate-energy-emissions>.
13. KOOT, Martijn; WIJNHOVEN, Fons. Usage impact on data center electricity needs: A system dynamic forecasting model. *Applied Energy*. 2021, vol. 291. ISSN 0306-2619. Available from DOI: <https://doi.org/10.1016/j.apenergy.2021.116798>.
14. MÖLLER, Matthias; VUIK, Cornelis. On the impact of quantum computing technology on future developments in high-performance scientific computing. *Ethics and Information Technology*. 2017, vol. 19, pp. 253–269. Available from DOI: 10.1007/s10676-017-9438-0.
15. ARUTE, Frank; ARYA, Kunal; BABBUSH, Ryan; BACON, Dave; BARDIN, Joseph C.; BARENDT, Rami, et al. Quantum supremacy using a programmable superconducting processor. *Nature*. 2019, vol. 574, pp. 505–510. ISSN 1476-4687. Available from DOI: 10.1038/s41586-019-1666-5.
16. MELANSON, Denis; KHATER, Mohammad Abu; AIFER, Maxwell; DONATELLA, Kaelan; GORDON, Max Hunter; AHLE, Thomas; CROOKS, Gavin; MARTINEZ, Antonio J.; SBAHI, Faris; COLES, Patrick J. Thermodynamic Computing System for AI Applications. *ArXiv*. 2023. Available from DOI: 10.48550/arXiv.2312.04836.
17. MUCCI, Tim; STRYKER, Cole. *What Is Artificial Superintelligence?* [Online]. IBM, 2023-12. [visited on 2024-05-03]. Available from: <https://www.ibm.com/topics/artificial-superintelligence>.

18. CHOI, Charles Q. IBM's Quantum Leap: The Company Will Take Quantum Tech Past the 1,000-Qubit Mark in 2023. *IEEE Spectrum*. 2023, vol. 60, no. 1, pp. 46–47. ISSN 1939-9340. Available from DOI: 10.1109/MSPEC.2023.10006669.
19. GAMBETTA, Jay. The hardware and software for the era of quantum utility is here. *IBM Quantum Research Blog* [online]. 2023 [visited on 2024-05-03]. Available from: <https://www.ibm.com/quantum/blog/quantum-roadmap-2033>.
20. GYONGYOSI, Laszlo; IMRE, Sandor. A Survey on quantum computing technology. *Computer Science Review*. 2019, vol. 31, pp. 51–71. ISSN 1574-0137. Available from DOI: <https://doi.org/10.1016/j.cosrev.2018.11.002>.
21. BIAMONTE, Jacob; WITTEK, Peter; PANCOTTI, Nicola; REBENTROST, Patrick; WIEBE, Nathan; LLOYD, Seth. Quantum machine learning. *Nature*. Springer Science and Business Media LLC, 2017, vol. 549, no. 7671, pp. 195–202. ISSN 1476-4687. Available from DOI: 10.1038/nature23474.
22. SCHULD, Maria; SWEKE, Ryan; MEYER, Johannes Jakob. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*. American Physical Society, 2021, vol. 103. Available from DOI: 10.1103/PhysRevA.103.032430.
23. LI, Guangxi; YE, Ruilin; ZHAO, Xuanqiang; WANG, Xin. Concentration of Data Encoding in Parameterized Quantum Circuits. *ArXiv*. 2022, vol. abs/2206.08273. Available from DOI: 10.48550/arXiv.2206.08273.
24. ALVAREZ-RODRIGUEZ, Unai; LAMATA, Lucas; ESCANDELL-MONTERO, Pablo; MARTÍN-GUERRERO, José D.; SOLANO, Enrique. Supervised Quantum Learning without Measurements. *Scientific Reports*. 2017, vol. 7, no. 1. ISSN 2045-2322. Available from DOI: 10.1038/s41598-017-13378-0.
25. SCHULD, Maria; PETRUCCIONE, Francesco. *Supervised Learning with Quantum Computers*. Springer Cham, 2018. ISBN 978-3-319-96423-2. Available from DOI: 10.1007/978-3-319-96424-9.
26. WEIGOLD, Manuela; BARZEN, Johanna; LEYMANN, Frank; SALM, Marie. Encoding patterns for quantum algorithms. *IET Quantum Communication*. 2020, vol. 2, no. 4, pp. 141–152. Available from DOI: 10.1049/qt2.12032.
27. ARAUJO, Israel F.; PARK, Daniel K.; PETRUCCIONE, Francesco; SILVA, Adenilton J. da. A divide-and-conquer algorithm for quantum state preparation. *Scientific Reports*. 2021, vol. 11, no. 1. ISSN 2045-2322. Available from DOI: 10.1038/s41598-021-85474-1.

28. NIELSEN, Michael A.; CHUANG, Isaac L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 978-1-107-00217-3. Available from DOI: 10.1017/CB09780511976667.
29. WEIGOLD, Manuela; BARZEN, Johanna; LEYMAN, Frank; SALM, Marie. Expanding Data Encoding Patterns For Quantum Algorithms. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. 2021, pp. 95–101. Available from DOI: 10.1109/ICSA-C52384.2021.00025.
30. MOTTONEN, Mikko; VARTIAINEN, Juha J.; BERGHOLM, Ville; SALOMAA, Martti M. Transformation of quantum states using uniformly controlled rotations. *ArXiv*. 2004. Available from DOI: 10.48550/arXiv.quant-ph/0407010.
31. PRESKILL, John. Quantum Computing in the NISQ era and beyond. *Quantum*. 2018, vol. 2. ISSN 2521-327X. Available from DOI: 10.22331/q-2018-08-06-79.
32. SHOR, Peter W. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*. American Physical Society, 1995, vol. 52. Available from DOI: 10.1103/PhysRevA.52.R2493.
33. LADD, T. D.; JELEZKO, F.; LAFLAMME, R.; NAKAMURA, Y.; MONROE, C.; O'BRIEN, J. L. Quantum computers. *Nature*. 2010, vol. 464, pp. 45–53. ISSN 1476-4687. Available from DOI: 10.1038/nature08812.
34. GAMBETTA, Jay M.; CHOW, Jerry M.; STEFFEN, Matthias. Building logical qubits in a superconducting quantum computing system. *npj Quantum Information*. 2017, vol. 3. ISSN 2056-6387. Available from DOI: 10.1038/s41534-016-0004-0.
35. GONZALEZ, Danielle; ZIMMERMANN, Thomas; NAGAPPAN, Nachiappan. The State of the ML-universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub. In: Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 431–442. MSR '20. ISBN 9781450375177. Available from DOI: 10.1145/3379597.3387473.
36. IBM. *Qiskit* [online]. [N.d.]. [visited on 2024-05-04]. Available from: <https://www.ibm.com/quantum/qiskit>.
37. REPOSITORY CONTRIBUTORS. *Qiskit: An Open-Source SDK for Working with Quantum Computers* [online]. GitHub, 2017 [visited on 2024-05-04]. Available from: <https://github.com/Qiskit>.
38. JOHNSON, Blake; TREINISH, Matthew; BELLO, Luciano; LISHMAN, Jake; KRSULICH, Kevin. Qiskit 1.0 coming in February, 2024. *IBM Quantum Research Blog* [online]. 2023 [visited on 2024-05-10]. Available from: <https://www.ibm.com/quantum/blog/qiskit-1-launch>.

39. IBM QUANTUM DOCUMENTATION. *Bit-ordering in the Qiskit SDK* [online]. [visited on 2024-05-11]. Available from: <https://docs.quantum.ibm.com/build/bit-ordering>.
40. IBM QUANTUM DOCUMENTATION. *Documentation* [online]. [visited on 2024-05-14]. Available from: <https://docs.quantum.ibm.com/>.
41. IBM QUANTUM DOCUMENTATION. *CSwapGate* [online]. [visited on 2024-05-14]. Available from: <https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.CSwapGate>.
42. IBM QUANTUM DOCUMENTATION. *Native gates and operations* [online]. [visited on 2024-05-14]. Available from: <https://docs.quantum.ibm.com/run/native-gates>.
43. IBM QUANTUM DOCUMENTATION. *Transpiler* [online]. [visited on 2024-05-14]. Available from: <https://docs.quantum.ibm.com/api/qiskit/transpiler#optimization-stage>.



Appendix

Repository

All the source code files created for this thesis are hosted in the following publicly accessible *GitHub* repository:

<https://github.com/PavelSlaninka/QuantumEncodingThesis>

The repository contains the implementation of the four encoding methods introduced in the chapter Data encoding methods. The repository also contains multiple *Jupyter notebooks* that are mainly used to demonstrate the implemented encoding methods. The notebooks also contain several figures displayed throughout the thesis, along with the source code used to generate them. The main implementation aspects are discussed in the chapter Implementation, where an overview of this repository is presented. The following page contains a summary of the repository contents.

The structure of the repository is outlined below, along with a brief description of each file included:

