

ORIGINAL RESEARCH

Encoding patterns for quantum algorithms

Manuela Weigold  | Johanna Barzen  | Frank Leymann  | Marie Salm 

University of Stuttgart, Stuttgart, Germany

Correspondence

Marie Salm, University of Stuttgart, Universitätsstr.
38, Stuttgart, Germany.
Email: marie.salm@iaas.uni-stuttgart.de

Funding information

Bundesministerium für Wirtschaft und Energie,
Grant/Award Number: 01MK20005N

Abstract

As quantum computers are based on the laws of quantum mechanics, they are capable of solving certain problems faster than their classical counterparts. However, quantum algorithms with a theoretical speed-up often assume that data can be loaded efficiently. In general, the runtime complexity of the loading routine depends on (i) the data encoding that defines how the data is represented by the state of the quantum computer and (ii) the data itself. In some cases, loading the data requires at least exponential time that destroys a potential speed-up. And especially for the first generation of devices that are currently available, the resources (qubits and operations) needed to encode the data are limited. In this work, we, therefore, present six patterns that describe how data is handled by quantum computers.

KEYWORDS

computational complexity, quantum computing techniques, quantum computing

1 | INTRODUCTION

Recent advantages in quantum technology have led to the first generation of commercial quantum computers [1, 2]. Compared to their classical counterparts, quantum computers have the potential to solve certain problems faster [3]. For example, factoring large prime numbers [4] or unstructured search [5] can, in principle, be done faster by a quantum computer. These speed-ups are possible because quantum computers are based on quantum bits (qubits) and, therefore, can exploit *superposition* or *entanglement*, which are unique characteristics of quantum mechanics. The quantum computers of this first generation have been coined *Noisy Intermediate Scale Quantum* (NISQ) devices [2] as they still have severe limitations: Their qubits are noisy and only stable for a limited amount of time until they decay. Measured by their number of qubits, the computers are of intermediate size; ranging from a few dozens to a few hundred qubits. Nevertheless, it is expected that hardware will further improve [1, 6, 7] and enable novel applications for quantum computers.

However, programming these quantum devices is challenging as their quantum nature as well as their hardware limitations must be taken into account. One key difference to

classical computing is the way data is handled by quantum computers. Current quantum computers do not have access to a database or a quantum version of random access memory (RAM) [8]. Thus, in order to use data in a quantum computer, this data has to be loaded by encoding it into the state of the qubits. However, there are various data encodings that define how the data can be represented by qubits. Note that the runtime of the loading routine depends on (i) the chosen data encoding, and (ii) on the data itself. It was proven that in the worst case, the loading routine is at least of exponential complexity, that is, an exponential number of parallel operations is needed [9]. However, a common assumption of algorithms with a theoretical linear or exponential speed-up is that the process of loading data requires only logarithmic or linear time [10]. Aaronson [10] refers to these assumptions as *fine-print* of the algorithm, indicating that these are often overlooked or at least not prominent to readers. Especially in the current NISQ era, understanding these runtime implications is crucial as only a certain amount of operations can be executed on noisy qubits. As developing software for quantum computers requires the interdisciplinary collaboration of, for example, physicists and computer scientists [11], a shared understanding of how input data is processed in particular is needed.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. *IET Quantum Communication* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

This paper is an extended version of our previous work [12] where we presented three pattern primitives and two data encoding patterns (ANGLE and QUANTUM RANDOM ACCESS MEMORY [QRAM] ENCODING) for quantum computing. Each pattern documents a proven solution for a re-occurring problem and may be built on pattern primitives, which sometimes correspond to fundamental concepts of the domain. Through the explicit documentation of similar solutions in quantum algorithms, patterns enable to make knowledge about quantum algorithms accessible to a broader audience. In this work, we contribute four new patterns that further describe how data is handled by quantum computers: SCHMIDT DECOMPOSITION describes a method for creating an arbitrary quantum state and thus can be used to load input data. For matrices, which are often used as an input for quantum algorithms, MATRIX ENCODING describes how such a type of input data can be represented as an operation on a quantum computer. QUANTUM PHASE ESTIMATION describes how an eigenvalue of such an operation can be encoded into a quantum state using the relative phase of quantum states in an intermediate step. POST-SELECTIVE MEASUREMENT can be used if a result is encoded in a specific part of the state. Additionally, we add a use case that demonstrates the usage of several of the patterns within a prominent algorithm to further illustrate the relations between them. For a better understanding of the new patterns, the contributions of our previous work [12] are presented in Sections 2.3.1, 3.2, 3.3 and 5. As the new patterns are also interconnected with the two original data encoding patterns, some minor adjustments have been made to the existing patterns and pattern primitives in order to describe their relations in more detail.

The rest of this paper is structured as follows: We introduce fundamentals and pattern primitives for quantum computing in Section 2 to establish a common understanding and terminology. In Section 3, we present the patterns and compare them to patterns of previous work [13]. A use case in Section 4 demonstrates how patterns of this work build up a broader algorithm. Section 5 describes related work and Section 6 concludes the paper and gives an outlook on future work.

2 | FUNDAMENTALS

In this section, we first introduce patterns and pattern primitives. This is followed by an overview of quantum computing and quantum algorithms. Finally, fundamental building blocks of quantum algorithms are introduced as pattern primitives.

2.1 | Patterns and pattern primitives

In this paper, we build on the concept of patterns of Alexander et al. [14]. They define patterns as structured, human-readable documents that document a proven solution to a reoccurring problem in a given context. Pattern primitives describe fundamental elements that re-occur in patterns and have first

been introduced in the context of software architecture to describe basic architectural units [15], for example, ports or components. Since then, the concept has been applied to various other domains [16–18]. Pattern primitives are more specific than the abstract solution of a pattern [15] and are, therefore, especially suited to establish a common ground and base terminology for patterns within one domain [19].

2.2 | Quantum computing

Since the first proposal of a quantum computer, various quantum computing models have been established, which describe computations in a different manner, for example, the adiabatic [20], gate-based [21] or one-way model [22]. In this work, we focus on the gate-based model that serves as the basis for many current devices [23], for example, by IBM¹, or Rigetti². However, in principle, our collection of quantum computing patterns can be extended with patterns or primitives for other models in the future.

For the first decades, quantum computing was mostly driven by research. Many algorithms for quantum computers were published before any commercial hardware existed [5, 24]. Thus, these first algorithms were only of theoretical relevance until the first commercial quantum computer was released in 2016 [25]. And even though until today no QRAM [26] exists, it is a common prerequisite of many algorithms [27–29].

Quantum algorithms are often hybrid and, therefore, consist of classical and quantum calculations. For example, Shor's algorithm [24] uses a quantum subroutine in an intermediate step. In the following, we describe a typical structure of quantum computation and then define the main components as primitives in the next section. Figure 1 illustrates the computation as a quantum circuit. First, the qubits of the register are initialised as $|0\rangle$. Then, an initial state is prepared. This step also includes loading data that (as a result) is encoded in the state of the register. Afterwards, quantum gates perform unitary transformations on the state of the register. Finally, one or multiple qubits are measured as the result of this computation indicated by measurement gates. The overall quantum circuit is characterised by its *depth*, which is the number of sequential executable gates and its *width* that equals the number of required qubits [30].

2.3 | Pattern primitives for quantum algorithms

In this section, we present three pattern primitives for quantum computing. Each primitive has a *Name* and an *Icon* as a graphical representation. This is followed by a short *Description* and if applicable, a *Sketch*. As quantum computing is based on quantum mechanics, this is followed by a

¹<https://www.ibm.com>

²<https://www.rigetti.com>

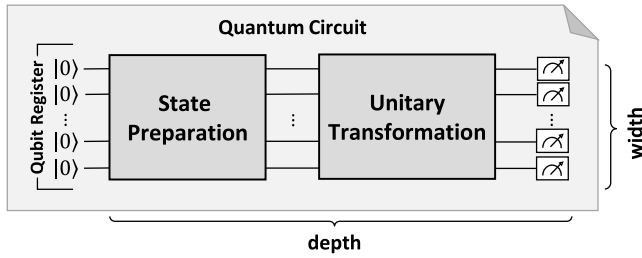


FIGURE 1 Typical structure of a quantum calculation that is represented as a quantum circuit that has a defined depth and width

Mathematical Definition and an *Example* that illustrates a concrete instance. Our primitives cover fundamental building blocks of quantum computing that serve as a basis for our data encoding patterns in Section 3. Note that we do not claim that this is an exhaustive list of pattern primitives for the domain. We first introduce the most basic primitive (QUBIT) and then present QUBIT REGISTER and QUANTUM GATE.



QUBIT

Description: A quantum bit (qubit) is the basic unit for information in quantum computing.

Mathematical Definition: The state of a qubit is represented by a two-dimensional vector $|\psi\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \text{ where } \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1 \quad (1)$$

Here, we used the Dirac notation for vectors where in particular

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

The set $\{|0\rangle, |1\rangle\}$ is a basis of the two-dimensional vector space of the qubit and is also referred to as the *computational basis*. The complex numbers α and β are the *amplitudes* of the quantum system and determine the possibility of a measurement outcome: with a probability of $|\alpha|^2$, a measurement in the computational basis results in $|0\rangle$ and with a probability of $|\beta|^2$, measuring results in the $|1\rangle$ state. As these are the only possible outcomes of a measurement, their possibilities ($|\alpha|^2$ and $|\beta|^2$) must sum up to 1. If $\alpha, \beta \neq 0$, the qubit is in a *superposition*—and therefore, in a linear combination—of $|0\rangle$ and $|1\rangle$. Since α and β are complex numbers, they can be written as $\alpha = r_1 e^{i\varphi_1}$ and $\beta = r_2 e^{i\varphi_2}$. The difference $\varphi_1 - \varphi_2$ defines the *relative phase* of the state. Multiplying both α and β by a complex number $|c| \neq 1$ changes the *global phase* of the state. In contrast to the relative phase, the global phase cannot be detected by measurement.

A common visualisation of the state of a qubit is the Bloch Sphere (see Figure 2) where each possible state is mapped to a point on the surface of the sphere.

Example: For example, the so-called $|+\rangle$ state that is depicted in Figure 2 is an equal superposition of $|0\rangle$ and $|1\rangle$:

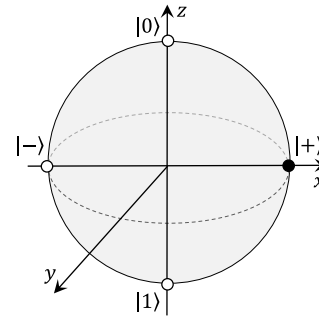


FIGURE 2 Bloch sphere representation of a qubit

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

The probability for both measurement outcomes ($|0\rangle$ and $|1\rangle$) is equal and can be calculated as follows: $|\frac{1}{\sqrt{2}}|^2 = 0.5$.



QUBIT REGISTER

Description: Multiple qubits can form a qubit register whose state is represented by a vector in a high-dimensional complex vector space.

Mathematical Definition: The state of an n -qubit register is defined as:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \text{ where } \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1. \quad (2)$$

In the equation above, $|i\rangle$ is used as a common notation for $|b_0 \dots b_n\rangle$ where $b_0 \dots b_n$ is the binary representation of i . If there are at least two amplitudes $\alpha_i, \alpha_j \neq 0, i \neq j$, the register is in superposition of all states with a non-zero amplitude. If each quantum system s is prepared in the state $|\psi_s\rangle$, the composite system can be described as a tensor product [21]:

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle \quad (3)$$

Often, is abbreviated to $|\psi_0 \psi_1 \dots \psi_n\rangle$. If the state is not separable, that is, it cannot be expressed as a tensor product of its components, it is in an *entangled* state [21].

Example: An example of an entangled state is $|\phi^+\rangle$:

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Note that measuring only one of the qubits determines the measurement outcome of the other (both are measured as either $|0\rangle$ or $|1\rangle$). A state that is not entangled is, for example,

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$$

as it can be rewritten as a product of the individual qubits (refer to [21] for an in-depth introduction on entanglement):

$$|\psi\rangle = |0\rangle \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right)$$



QUANTUM GATE

Description: Analogue to classical logic gates (e.g. AND, OR, and NOT) that act on bits, quantum computers manipulate qubits with quantum gates. Often, a calculation is described as a quantum circuit (Figure 3) that visualises the sequence of quantum gates that are applied to each qubit. Quantum gates can act on either one or multiple qubits.

Mathematical Definition: Except for measurement gates (i.e. the second gate on the first qubit in Figure 3), a quantum gate is defined by a unitary matrix U for which by definition the inverse matrix U^\dagger can be applied to undo the computation of U . By multiplying this matrix with a state vector, the resulting state vector can be obtained.

Example: An example of a one-qubit gate is R_y [9]:

$$R_y(2x) = \begin{pmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{pmatrix}$$

This defines a rotation by an angle $2x$ around the y axis of the Bloch Sphere (refer to Figure 2 for a visual representation). For example, rotating $|0\rangle$ around an angle of π in the Bloch Sphere can be expressed as follows:

$$R_y(\pi)|0\rangle = R_y\left(2\frac{\pi}{2}\right)|0\rangle = \begin{pmatrix} \cos \frac{\pi}{2} & -\sin \frac{\pi}{2} \\ \sin \frac{\pi}{2} & \cos \frac{\pi}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

which can be further simplified to:

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

and thus, results in a $|1\rangle$ state.

3 | PATTERNS

In this section, we first describe our method for collecting patterns (Section 3.1) and the pattern format (Section 3.2). This is followed by a presentation of patterns of which an overview is shown in Figure 4. In Section 3.3, three patterns for data encoding and state preparation are described: ANGLE ENCODING, QRAM ENCODING, and SCHMIDT DECOMPOSITION (note that the first two patterns are also contained in [12]). In Section 3.4, two patterns for unitary transformations are introduced: MATRIX ENCODING and QUANTUM PHASE ESTIMATION, followed by POST-SELECTIVE MEASUREMENT, a

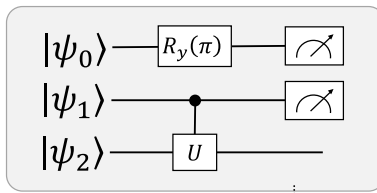


FIGURE 3 Exemplary quantum circuit. Quantum gates are applied in temporal order from left to right. Quantum gates are defined for either one (e.g. R_y) or multiple qubits, for example, the gate that is applied to the other two qubits

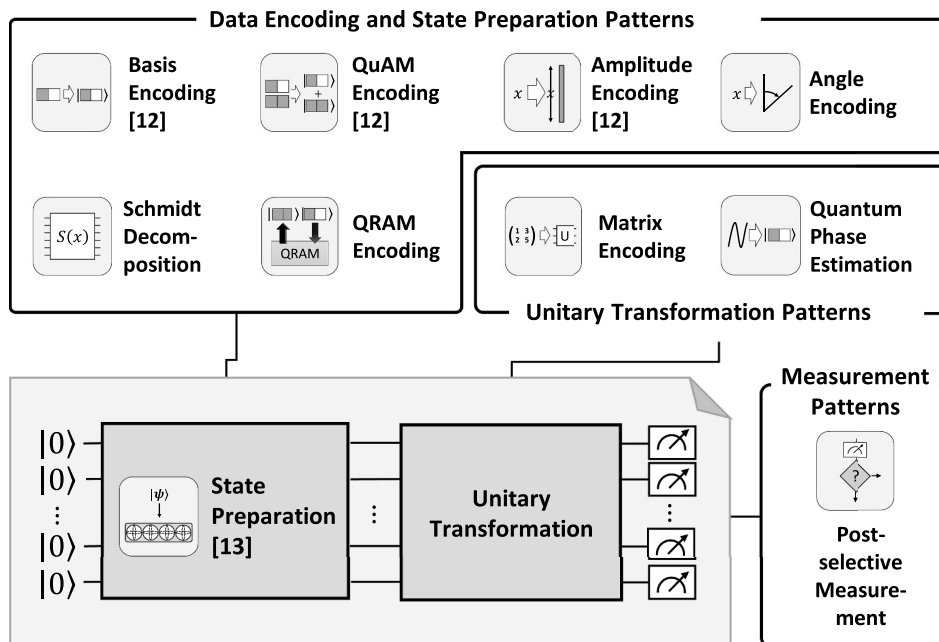


FIGURE 4 Overview of patterns included in this work

measurement pattern. An excerpt of all patterns can also be found online at *Quantum Computing Patterns*³.

3.1 | Method

Patterns are abstracted from existing solutions [14]. The patterns presented here and in previous publications [13, 31] were identified using the pattern authoring process of Fehling et al. [19], which we also describe in [13]. First, we analysed scientific publications, books, and technical documentation to collect re-occurring solutions. If we found at least three occurrences (Coplien's rule [32]) of a pattern candidate, we authored a pattern. For loading data, a proven solution is a specific data encoding used in various quantum algorithms. Note that given the current state of the art of quantum computing, we do not require a concrete implementation. Instead, we focus on authoring patterns for writing and understanding quantum algorithms. However, in future work, the patterns should be validated further in real applications.

3.2 | Pattern format

Depending on the domain, pattern authors use different formats for their patterns. Here, we use the pattern format of our previous work [13] that was based on the existing format of Fehling et al. [33]. Each pattern is introduced by a *Name* and an *Icon* that serves as a graphical representation of the pattern. Next to the icon, we denote the *Intent* that briefly summarises the purpose of the pattern. If the pattern is also known under different names, these are listed as an *Alias*. Then, the problem and the circumstances of the pattern are described in the *Context* section before the *Forces* are presented. The *forces* are trade-offs or considerations that must be taken into account for solving the problem. The *Solution* itself is described in an abstract manner and often visualised by a *Solution Sketch*. Consequences of the *solution* are described as the *Result* of the solutions. This is followed by an optional section for *Variants* of the pattern. As patterns are often applied in combination or solve similar problems, we describe the connections between them in the *Related Patterns* section. Finally, we list *Known Uses* of the pattern. For encoding patterns, algorithms that use the encoding and state preparation methods are listed here. Additionally, concrete implementations of these algorithms can also be referred to in this section.

3.3 | Data encoding and state preparation patterns

Data encodings for quantum computing define how data is represented by the state of a quantum system. Data encoding






patterns (of which an overview is given in Table 1) describe a particular encoding as a trade-off between three major forces:

- (i) *The number of qubits needed for the encoding should be minimal* because current devices are of intermediate size and thus only contain a limited number of qubits.
- (ii) *The number of parallel operations needed to realise the encoding should be minimal to minimise the width of the quantum circuit*—ideally, the loading routine is of constant or logarithmic complexity
- (iii) *The data must be represented in a suitable manner* for further calculations, for example, arithmetic operations.

Each of these patterns further refines STATE PREPARATION, a pattern of previous work [31] that describes the first phase at the beginning of an algorithm (see Figure 4). Each encoding pattern introduces or references a suitable state preparation method. As an alternative, a state preparation method for preparing an arbitrary quantum state can be used. The SCHMIDT DECOMPOSITION pattern describes one general approach for creating an arbitrary quantum state.

Table 1 gives an overview of the data encoding patterns presented in this work (marked in bold) as well as previous [13] encoding patterns. While the encodings of the first two patterns define how a single numerical data-point x_i is encoded, the three other patterns describe how a set X of n data-points can be represented. The representation of data in BASIS ENCODING is also part of two other encodings (QUAM and QRAM ENCODING). Therefore, we explain this pattern in more detail before we present the patterns ANGLE ENCODING, QRAM ENCODING, and SCHMIDT DECOMPOSITION in detail. Since we listed forces for data encoding patterns above, we omit them in the descriptions of ANGLE and QRAM ENCODING. For a BASIS ENCODING of a numerical data-point, its value is first approximated by its binary representation. The resulting bitstring $b_m \dots b_{-k}$ is then encoded by the $|b_m \dots b_{-k}\rangle$ state. Therefore, every bit of its bitstring is represented by a single qubit. Thus, BASIS ENCODING is not efficient in terms of the required number of qubits. In comparison, QUAM ENCODING

TABLE 1 Comparison of data encoding patterns. For the QUANTUM RANDOM ACCESS MEMORY (QRAM) ENCODING, we assume that all n data points are loaded

Encoding pattern	Encoding	Req. qubits
 BASIS ENCODING [13]	$x_i \approx \sum_{i=-k}^m b_i 2^i \mapsto b_m \dots b_{-k}\rangle$	$l = k + m$ per data-point
 ANGLE ENCODING	$x_i \mapsto \cos(x_i) 0\rangle + \sin(x_i) 1\rangle$	1 per data-point
 QUAM ENCODING [13]	$X \mapsto \sum_{i=0}^{n-1} \frac{1}{\sqrt{n}} x_i\rangle$	l
 QRAM ENCODING	$X \mapsto \sum_{i=0}^{n-1} \frac{1}{\sqrt{n}} i\rangle x_i\rangle$	$\lceil \log n \rceil + l$
 AMPLITUDE ENCODING [13]	$X \mapsto \sum_{i=0}^{n-1} x_i i\rangle$	$\lceil \log n \rceil$

³<https://quantumcomputingpatterns.org/>

uses superposition to encode a set of data points in a qubit register of the same length (assuming that the binary representation of all values is equally long or padded with zeros). QUANTUM RANDOM ACCESS MEMORY ENCODING needs $\lceil \log n \rceil$ additional qubits to represent the same data. Even more compact is the representation of data in AMPLITUDE ENCODING for which only $\lceil \log n \rceil$ qubits are needed. However, for an arbitrary data set, the last three encodings of Table 1 cannot be realised efficiently; that is in a constant or logarithmic number of parallel operations. While BASIS ENCODING and ANGLE ENCODING are not efficient in terms of required qubits, they can be realised in constant time (one single parallel operation). Further details and consequences of the encodings listed in Table 1 can be found in the corresponding patterns.

ANGLE ENCODING



Represent each data-point by a separate qubit

Alias: Qubit Encoding [34], (Tensor) Product Encoding [35].

Context: An algorithm requires an efficient encoding schema to be able to perform as many operations as possible within the decoherence time after the data has been loaded.

Solution: First, normalise all data-points that should be encoded to the interval $[0, \frac{\pi}{2}]$ [34]. Each value x_i is then represented by a single qubit as follows (Figure 5): a rotation around the y -axis of the Bloch Sphere (refer to Figure 2) is applied. Hereby, the angle for the rotation depends on the data value (see Section 2.3 for a more detailed description of the operation).

Result: The resulting quantum state for this encoding is separable, that is, the qubits are not entangled:

$$|\psi\rangle = \begin{pmatrix} \cos x_0 \\ \sin x_0 \end{pmatrix} \otimes \begin{pmatrix} \cos x_1 \\ \sin x_1 \end{pmatrix} \otimes \dots \otimes \begin{pmatrix} \cos x_n \\ \sin x_n \end{pmatrix} \quad (4)$$

The main advantage of this encoding is that it is very efficient in terms of operations: Only a constant number of parallel operations is needed regardless of how many data values need to be encoded [34]. However, the number of data values affects how many qubits are needed: One qubit is required to encode one component of the input vector. Thus, as only single-qubit rotations are required the state preparation is efficient while the number of qubits for this encoding is not optimal [36].

Related Patterns: This pattern further refines STATE PREPARATION.

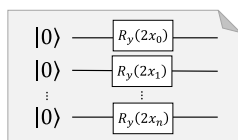


FIGURE 5 Quantum circuit for loading data in ANGLE ENCODING based on Leymann and Barzen [35]

Variants: The authors in [34] present *dense angle encoding* as an alternative encoding that exploits an additional property of qubits (relative phase) to use only $\frac{n}{2}$ qubits to encode n data points.

Known Uses: The authors in [34, 36] present a classifier based on this encoding. The encoding is also used in quantum image processing: In the so-called flexible representation of a quantum image, one qubit represents the colour information of a pixel, whereas another register represents the position [37]. In the context of quantum neural networks, a qubit using this encoding has been referred to as a quantum neuron (quron) [38]. PennyLane provides a state preparation method for angle encoding [39] for which the axis of the rotation can be specified (x , y , or z). If no loading routine is provided, this encoding can be prepared with standard qubit rotations in a straightforward manner [35].

QUANTUM RANDOM ACCESS MEMORY (QRAM) ENCODING



Use a quantum random access memory to access a superposition of data values at once

Context: An algorithm requires random access to the data values of the input.

Solution: A classical RAM that receives an address with a memory index, loads the data stored at this address into an output register. Quantum random access memory provides the same functionality, but the address and output register are quantum registers [8]. As a result, both the address and the output register can be in a superposition of multiple values. Figure 6 illustrates the basic functionality of a QRAM [26] that receives a superposition of the first two addresses $(\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle)$ as input and loads the corresponding data values into an empty output register. This leads to the following state of the overall quantum system:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle|000\rangle + \frac{1}{\sqrt{2}}|01\rangle|110\rangle \quad (5)$$

In general, loading m of n data values with a QRAM can, therefore, be described as the following operation [9]:

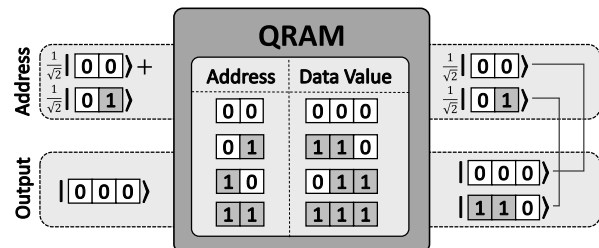


FIGURE 6 Basic functionality of a quantum random access memory (QRAM) is based on [8]. Given an address register that is in a superposition of addresses ($|00\rangle$ and $|01\rangle$), QRAM creates a superposition of addresses and their data values: $\frac{1}{\sqrt{2}}|00\rangle|000\rangle + \frac{1}{\sqrt{2}}|01\rangle|110\rangle$

$$\frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |a\rangle_i |0\rangle \xrightarrow{\text{QRAM}} \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |a\rangle_i |x_a\rangle \quad (6)$$

where the first register is the address register that is in a superposition of all m addresses to be loaded and the second register is the output register. Further, $|a\rangle_i$ specifies the address of the i th data value to be loaded and x_a is the data value associated with this address. The QRAM loads each data value $|x_a\rangle$ into the output register such that $|a\rangle_i |x_a\rangle$ is contained in the combined state of both registers. Depending on the data values, this state may be entangled.

Result: For this encoding, l qubits are needed to encode the data values using BASIS ENCODING. The address register requires $\lceil \log(n) \rceil$ additional qubits for a maximum of n addresses. The computational properties are similar to those of BASIS and QUAM ENCODING: As a superposition of the encoded data values is prepared, data can be processed in parallel (quantum parallelism) and arithmetic operations such as addition or multiplication can be used.

An algorithm that uses QRAM assumes that an efficient procedure exists that can be used to perform state preparation in logarithmic runtime [9]. As long as there are no hardware implementations of QRAM, a state preparation method must be used that mirrors the loading process of a QRAM. The main drawback of QRAM encoding is that, in general, no state preparation method for arbitrary input data exists that is as efficient as a QRAM. A promised speed-up of an algorithm that relies on QRAM can only be realised if data can be encoded efficiently by a known state preparation method.

Related Patterns: This pattern further refines STATE PREPARATION and uses BASIS ENCODING. CREATING ENTANGLEMENT [31] may be used, for example, loading the first and last address results in $\left(\frac{1}{\sqrt{2}} |00\rangle |000\rangle + \frac{1}{\sqrt{2}} |11\rangle |111\rangle \right)$ which is the entangled 5-qubit Greenberger-Horne-Zeilinger state. As a state preparation method, SCHMIDT DECOMPOSITION can be used.

Known Uses: An algorithm for state preparation is given by circuit family #3 of [40]. An alternative approach is presented in [41]. This encoding is used by algorithms for solving semi-definite programs [42]. Various other algorithms exist that require or are improved upon QRAM [26–29]. The Harrow-Hassidim-Lloyd (HHL) algorithm for solving linear equations [43] uses QRAM ENCODING to represent eigenvalues in an intermediate step [42].

SCHMIDT DECOMPOSITION



Prepare an arbitrary state

Context: A state $|s\rangle$ has to be prepared on an empty n -qubit register. If no state preparation method is known that exploits the structure of this state to prepare it efficiently, a method for creating an arbitrary state can be used instead.

Forces: For the creation of a circuit for an arbitrary state, the following forces apply:

- (i) The depth of the constructed state preparation circuit is preferably small.
- (ii) The runtime for constructing the state preparation circuit on a classical computer should not outweigh the potential benefit of using a quantum computer.

Solution: To generate a circuit for the creation of $|s\rangle$, it first needs to be expressed in terms of two subspaces V and W that span $H^{\otimes n}$. First, orthogonal basis $\{f_1, \dots, f_k\} \in V$ and $\{g_1, \dots, g_k\} \in W$ are chosen and $|s\rangle$ is represented as a linear combination of these basis vectors:

$$|s\rangle = \sum_{ij} b_{ij} \cdot f_i \otimes g_j \quad (7)$$

Then, the singular value decomposition (SVD) of the matrix $M = \{b_{ij}\}$ is computed (see [44] for detailed instructions):

$$M = (U_1 U_2) \begin{pmatrix} A \\ 0 \end{pmatrix} V^* \quad (8)$$

where the matrix U obtained by the SVD is rewritten by U_1 and U_2 . The entries of the diagonal matrix A build the set $\{\alpha_1, \dots, \alpha_m\}$, which defines the *Schmidt decomposition* of $|s\rangle$:

$$|s\rangle = \sum_{i=1}^m \alpha_i \cdot u_i \otimes v_i, \alpha_i \in \mathbb{R} \geq 0, \text{ where } \sum_{i=1}^m \alpha_i = 1 \quad (9)$$

where $\alpha_1, \dots, \alpha_m$ are the *Schmidt coefficients* for the *Schmidt basis* $\{u_i\}, \{v_i\}$. The circuit in Figure 7 can be used to prepare $|s\rangle$ on an empty register [45]: First, B transforms the amplitude of the first register to the Schmidt coefficients. Then, a series of CNOT operations copies this state to the second register. Finally, U_1 and V transform the computational basis states $\{e_i\}$ into the Schmidt basis states:

$$(U_1 \otimes V) \sum_{i=1}^m \alpha_i \cdot e_i \otimes e_i = \sum_{i=1}^m \alpha_i \cdot u_i \otimes v_i \quad (10)$$

For the execution on a quantum computer, the unitary matrices must be further decomposed into one and two qubit gates.

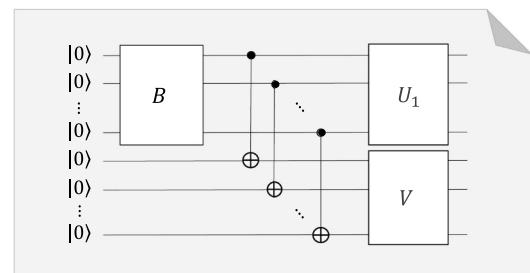


FIGURE 7 State preparation circuit for $|s\rangle$, adapted from [35]

Result: The state $|s\rangle$ is created in the register. For this state, the Schmidt coefficients α_i are known, which can be used to quantify entanglement [21]. The state $|s\rangle$ is separable if and only if exactly one of the Schmidt coefficients is non-zero. In the worst case, the depth of the circuit is exponential (more precisely: $\frac{23}{48}2^n$) [46]. Note that arbitrary state preparation was shown to be of exponential complexity, that is, a circuit of exponential depth will always be needed in the worst case.

Related Patterns: This pattern refines STATE PREPARATION and can be used as a state preparation method for AMPLITUDE or QRAM ENCODING.

Known Uses: This pattern can be used to create random states with a controlled amount of entanglement [47]. An implementation in Mathematica was provided in [48].

3.4 | Unitary transformation and measurement patterns

Unitary transformation patterns describe best practices for operations after an initial state has been created (see Figure 4). The following unitary transformation patterns complement patterns of this category of previous work [31]. First, MATRIX ENCODING is presented, which describes how a matrix can be represented as an operation. Then, QUANTUM PHASE ESTIMATION is introduced, which is a routine for estimating eigenvalues of an operation and uses two of the previously presented data encoding patterns. Finally, we present a measurement pattern, POST-SELECTIVE MEASUREMENT.

MATRIX ENCODING



Represent a matrix as an operation on a quantum computer

Alias: Dynamic Encoding [9].

Context: A matrix A has to be represented on a quantum computer as an operation U . Therefore, it is not sufficient to represent the entries of the matrix by a data encoding pattern as this does not define an operation.

Forces: A representation of an operation balances the following forces:

- (i) The depth of the circuit that represents the operation is preferably small.
- (ii) The eigenvalues and eigenvectors of the quantum operation should correspond to those of A .

Solution: If A is not Hermitian, encode

$$H = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} \quad (11)$$

instead of A . Since H is Hermitian, $U = e^{-iHt}$ is unitary and can be used to encode H where t specifies the time for which the operation is applied (usually t is chosen to be small).

Result: Since the eigenvectors $\{|v_i\rangle\}$ of the $n \times n$ matrix H form a basis, each state vector $|\psi\rangle$ can be written as a linear combination of these eigenvectors:

$$|\psi\rangle = \gamma_1|v_1\rangle + \dots + \gamma_n|v_n\rangle \quad (12)$$

Applying U to this state results in the following state [9]:

$$e^{-iHt}|\psi\rangle = e^{-i\lambda_1 t}\gamma_1|v_1\rangle + \dots + e^{-i\lambda_n t}\gamma_n|v_n\rangle \quad (13)$$

If $|\psi\rangle$ is an eigenvector of H and, thus, only one $\gamma_i \neq 0$, the state acquires a global phase shift: $U\gamma_i|v_i\rangle = e^{-i\lambda_i t}\gamma_i|v_i\rangle$ (see Section 2.3 for a definition of phase shift). This nicely reflects that applying the matrix A to an eigenstate leads to a scaling factor. The main drawback of this encoding is that the depth of the circuit that implements U can be exponential.

Related Patterns: This pattern can be used to define a unitary operation as input for QUANTUM PHASE ESTIMATION.

Known Uses: This encoding is often used for the simulation of a quantum system, see [21] for various examples. The HHL algorithm for solving linear equations [43] is an example of an algorithm that uses MATRIX ENCODING in conjunction with QUANTUM PHASE ESTIMATION (see Section 4). PennyLane provides an implementation for this encoding [49].

QUANTUM PHASE ESTIMATION (QPE)



Approximate the eigenvalue of a unitary matrix

Alias: Phase estimation algorithm (PEA).

Context: Given a unitary matrix U and one of its eigenstates, the corresponding eigenvalue should be determined. The eigenstate $|v\rangle$ is given on a register in BASIS ENCODING. Applying U to the eigenstate $|v\rangle$ results in a global phase:

$$U|v\rangle = e^{2\pi i\varphi}|v\rangle$$

where the eigenvalue $\lambda = e^{2\pi i\varphi}$ is uniquely determined by $\varphi \in [0, 1]$. Therefore, it is sufficient to estimate φ .

Forces: This pattern balances the following forces:

- (i) It is not possible to measure a global phase.
- (ii) φ should be estimated with high precision.
- (iii) In order to use current NISQ computers, the executed quantum circuits should have a low depth and width.
- (iv) Preferably, only a few executions on the quantum computer are needed.

Solution: Use the circuit shown in Figure 8 to estimate the approximation of θ . First, a register of m ancillae is brought into a UNIFORM SUPERPOSITION. Next, controlled versions of powers of U are applied on the register of the eigenstate following the scheme depicted in Figure 8. Each application of a controlled- U operation results in a *phase kickback* of the control qubit, that is, this qubit acquires a relative phase of φ . This results in the overall state:

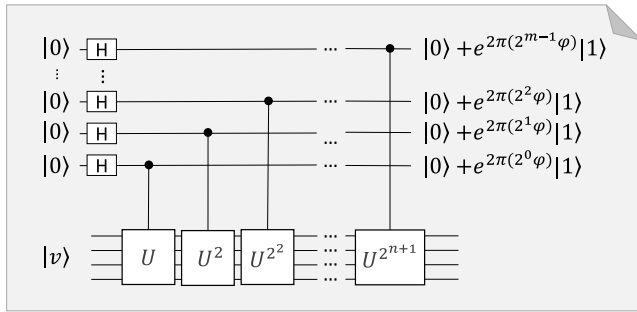


FIGURE 8 QUANTUM PHASE ESTIMATION routine

$$|\psi\rangle = \sum_{y=0}^{2^m-1} e^{2\pi i\varphi y} |y\rangle \quad (14)$$

where φ is encoded in the relative phase. To extract this information, the inverse of the quantum Fourier transformation is applied on the ancillae register.

Result: If φ is a rational number, the ancillae register contains the eigenvalue in BASIS ENCODING (assuming a proper number of ancillae). Otherwise, an approximation is produced with a probability of at least $\frac{4}{\pi}$. Increasing the precision of the approximation by adding more ancillae is costly because this also increases the number of required controlled- U operations. Because of these demanding hardware requirements, this algorithm is often regarded as non-suitable for NISQ devices.

Variants: The second register can also be initialised as an arbitrary quantum state, which is always a linear combination of eigenvectors. In this case, the algorithm approximates a superposition of eigenvalues for these eigenvectors in the output register. Other variants of this algorithm further improve the depth of the circuit or require fewer measurements.

Related Patterns: This pattern uses AMPLITUDE ENCODING and MATRIX ENCODING as input and produces an output in BASIS ENCODING.

Known Uses: QUANTUM PHASE ESTIMATION is at the heart of many algorithms [50]. One prominent example that we review in Section 4 in more detail is the HHL algorithm [43] for solving a set of linear equations. Other quantum machine learning algorithms follow a similar scheme [9, 51], for example, quantum support vector machine (QSVM) [27] or quantum principal component analysis [29]. Qiskit [52] provides an implementation for this pattern.

POST-SELECTIVE MEASUREMENT



Select one branch of a superposition to proceed with

Context: As quantum operations are unitary, they perform linear transformations. However, sometimes a non-linear transformation is desirable. For example, if the result of a computation is stored in one branch of the superposition, for example,

$$|\psi\rangle = \frac{1}{\sqrt{2}}|1\rangle|f(x)\rangle + \frac{1}{\sqrt{2}}|0\rangle|g(x)\rangle$$

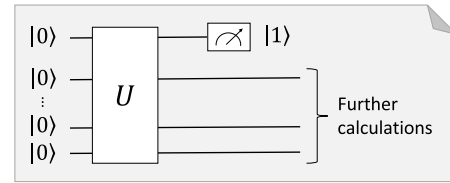


FIGURE 9 Circuit representation of a post-selective measurement based on [51]

one would like to proceed with $|1\rangle|f(x)\rangle$ and discard the rest of the superposition.

Solution: Use measurement to force the quantum state to collapse into one of the two branches. For the example above, measuring the first qubit in the computational basis results in either $|0\rangle$ or $|1\rangle$. If the measurement indicates that the preferred branch was selected (i.e. $|1\rangle$ for our example), one can proceed with further calculations (shown in Figure 9). Otherwise, the current computation is discarded and restarted from the beginning, that is, by first resetting each qubit to $|0\rangle$ followed by the operations of the rest of the circuit.

Result: The resulting approach is probabilistic; thus, the average number of iterations needed depends on the amplitude of the selected branch. In the example above, the algorithm proceeds with a probability of 50%.

Related Patterns: AMPLITUDE AMPLIFICATION [53] can be used to increase the probability to select a particular branch.

Known Uses: A non-linear transformation can be favourable for designing quantum neural networks [54]. The HHL algorithm [43] and various other algorithms that build on it, for example, the QSVM [51] uses this technique. POST-SELECTIVE MEASUREMENTS are also used in repeat-until-success circuits [55] that restore the state before the measurement via a recovery operation instead of restarting the whole computation. OpenQASM 3.0 introduces the functionality to explicitly model such an approach through the usage of measurements within a *while* loop [56].

4 | USE CASE: HHL ALGORITHM

This section presents a use case that demonstrates the usage of our data encoding patterns within a quantum algorithm. Therefore, we rephrase the well-known HHL algorithm [43] and, in particular, highlight the encoding patterns and data conversions used throughout the algorithm. For simplicity, we will omit normalisation constants in the following, assume a sufficient amount of ancillae and neglect further possible improvements of the algorithm. Finally, we draw general conclusions on the effect of the used data encodings.

The HHL algorithm solves a system of linear equations: given a matrix $A \in \mathbb{C}^{n \times n}$, which fulfils certain prerequisites⁴ and a vector $|b\rangle \in \mathbb{C}^n$, finds $|x\rangle \in \mathbb{C}^n$ such that $A|x\rangle = |b\rangle$. An equivalent formulation of this equation is: $|x\rangle = A^{-1}|b\rangle$.

⁴The condition number of A is low and A is a sparse matrix.

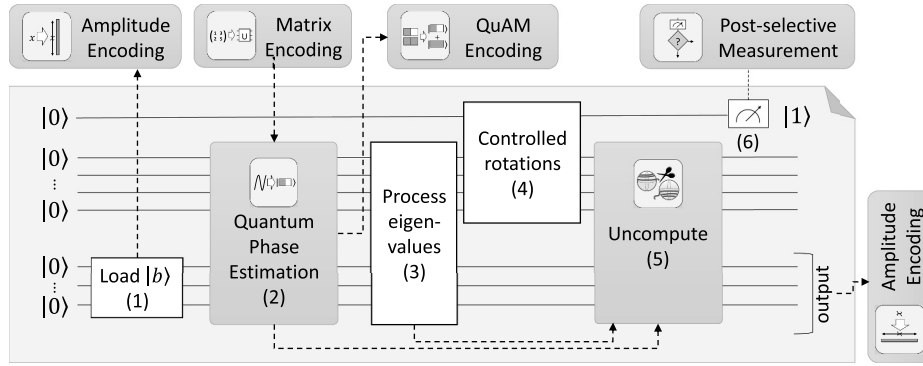


FIGURE 10 Circuit of the HHL Algorithm [43], and applied patterns

Due to the spectral theorem, $|b\rangle$ can be written in terms of the eigenvectors $\{u_i\}$ of A , that is, $|b\rangle = \sum_{j=0}^{n-1} \beta_j |u_j\rangle$, and thus, the equation can be reformulated as follows:

$$|x\rangle = \sum_{j=0}^{n-1} \lambda_j^{-1} \beta_j |u_j\rangle \quad (15)$$

which is, indeed, the output of the HHL algorithm.

To create the state of Equation (15), the circuit depicted in Figure 10 is used. First, the vector $|b\rangle$ is loaded as AMPLITUDE ENCODING into the first register (step 1):

$$|b\rangle = \sum_{i=0}^{n-1} b_i |i\rangle \quad (16)$$

Then, the QUANTUM PHASE ESTIMATION (with a MATRIX ENCODING of A as an input in step 2) results in a QUAM ENCODING of the approximated eigenvalues in the second register that is entangled with $|b\rangle$:

$$\sum_{j=0}^{n-1} \beta_j |\lambda_j\rangle |u_j\rangle \quad (17)$$

In the remaining steps (3–6), the algorithm inverts the eigenvalues and converts them from QUAM ENCODING (a digital encoding) to an analogous AMPLITUDE ENCODING, see [42] for a detailed description of the conversion. To disentangle the registers, an UNCOMPUTE operation is used. Since the inversion of the eigenvalues is implemented as a probabilistic operation, a POST-SELECTIVE MEASUREMENT, therefore, reveals if the overall algorithm succeeded (i.e. collapsed into the desired state of Equation (15)) or otherwise needs to be repeated.

The encodings used throughout the algorithm also reveal the main caveats of the algorithm that were pointed out in [10]:

- Loading $|b\rangle$ in AMPLITUDE ENCODING may require an exponential number of operations and, therefore, destroy the exponential speed-up—this has also been referred to as the ‘input problem’ [57]. In theory, it is of course possible to apply the HHL algorithm on the output of a previous

algorithm—however, the hardware requirements of HHL alone exceed the current state of NISQ devices [57].

- The MATRIX ENCODING in the QPE poses several restrictions on the input matrix A —otherwise, this part of the algorithm cannot be performed efficiently and the exponential speed-up vanishes.
- The algorithm produces the solution vector $|x\rangle$ in AMPLITUDE ENCODING—which prevents a simple readout of this value. It is possible to estimate the amplitudes by multiple repetitions of the algorithms—unfortunately, this also destroys the speed-up and has been referred to as the ‘output problem’ [57].

5 | RELATED WORK

The encoding patterns presented in this work complement encoding patterns [13] and patterns for quantum algorithms [31] of our previous works. While various other publications [58–60] introduce ‘quantum patterns’, none of these confirm patterns in the sense of Alexander et al. [14]. Therefore, to our best knowledge, no other pattern for quantum computing have been published so far.

Besides the encodings mentioned in this paper, various other encodings have been used in quantum algorithms. An overview of data encodings with a focus on quantum machine learning can be found in [9, 34, 36] while the authors in [37] describe encodings for quantum image processing. Duan et al. [51] review the HHL algorithm and data encodings used within this algorithm. However, they especially focus on the HHL while our work gives a general overview of existing data encodings and their usage in several different quantum algorithms. Thereby, we choose HHL as a demonstration of their applicability. In [34], the authors show that using ANGLE ENCODING as input data for a quantum classifier restricts the decision boundaries that the model can learn to a simple sine-function. Ref. [61] further generalises these findings and shows that the expressive power of a model can be increased by repeating the encoding. Ref. [35] draws general conclusions for data loading and various encodings in the NISQ era. In contrast to our approach, none of the previously mentioned overviews uses patterns to provide easy access to knowledge about data encodings.

In the context of quantum machine learning, the process of encoding data is also referred to as applying a quantum feature map ϕ [62, 63]. The authors point out that quantum feature maps and kernels in machine learning are related: Each feature map implicitly defines a quantum kernel. After applying the quantum feature map (and thus, loading the data), the data can be analysed in high-dimensional space. Therefore, each encoding that we presented in this paper gives rise to a quantum feature map ϕ and defines a quantum kernel.

Quantum random access memory is an essential component in larger quantum computers [26]. Several architectures have been proposed and demonstrated on a small scale (see, e.g. [64]). However, building a larger QRAM remains an open technical challenge for hardware providers [57, 65], which we also emphasise in our QRAM ENCODING pattern.

As current NISQ devices are limited by their hardware, various tools estimate or determine the resources required by a quantum algorithm (in particular, qubits and operations). Microsoft introduced the *Quantum Resources Estimator*⁵ for implementations in their quantum programming language Q#. The *NISQ Analyzer* of Salm et al. [66] considers concrete input data of quantum algorithms and suggests suitable implementations and quantum computers for execution. The knowledge about data encodings captured in our patterns can be used to further improve these estimations.

6 | CONCLUSION AND OUTLOOK

Quantum computing is an emerging topic that requires interdisciplinary collaboration and the combination of expert knowledge across various domains. In our original work [12], we introduced three pattern primitives and two data encoding patterns (ANGLE ENCODING and QRAM ENCODING) to foster a common understanding of this complex topic. We also compared them to data encoding patterns of previous work [13]. While QRAM ENCODING is used among many algorithms, one common assumption—the existence of a QRAM for state preparation—is currently not fulfilled by current devices. In this work, we contribute four new patterns: SCHMIDT DECOMPOSITION can be used to prepare an arbitrary state while a hardware implementation for a QRAM is still missing. MATRIX ENCODING and QUANTUM PHASE ESTIMATION are patterns for algorithms that require a matrix as an input. POST-SELECTIVE MEASUREMENT can be used to proceed with the branch where the solution is encoded. Finally, we demonstrated the usage of several of the patterns within an algorithm and conclude that encodings are a key ingredient for understanding quantum algorithms—and why certain input data can prevent an exponential speed-up.

In future work, we want to collect further patterns for quantum computing, for example, for hybrid algorithms. Furthermore, as the hardware progresses, the patterns collected so far should be further validated in real applications.

ACKNOWLEDGMENT

This work is partially funded by the BMWi project PlanQK (01MK20005N).

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Manuela Weigold  <https://orcid.org/0000-0002-4554-260X>

Johanna Barzen  <https://orcid.org/0000-0001-8397-7973>

Frank Leymann  <https://orcid.org/0000-0002-9123-259X>

Marie Salm  <https://orcid.org/0000-0002-2180-250X>

REFERENCES

1. National Academies of Sciences, Engineering and Medicine.: Quantum computing: progress and prospects. The National Academies Press, Washington (2019)
2. Preskill, J.: Quantum computing in the NISQ era and beyond. *Quantum*. 2, 79 (2018)
3. Horodecki, R., et al.: Quantum entanglement. *Rev. Mod. Phys.* 81(2), 865–942 (2009)
4. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* 41(2), 303–332 (1999)
5. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 96* (1996)
6. IBM's roadmap for scaling quantum technology. (2020). <https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/>
7. Scaling ionq's quantum computers: the roadmap. (2020). <https://ionq.com/posts/december-09-2020-scaling-quantum-computer-roadmap>
8. Johnston, E.R., Harrigan, N., Gimeno-Segovia, M.: *Programming quantum computers: essential algorithms and code samples*. O'Reilly Media, Incorporated (2019)
9. Schuld, M., Petruccione, F.: *Supervised learning with quantum computers*, ser. Quantum Science and Technology. Springer International Publishing (2018)
10. Aaronson, S.: Read the fine print. *Nat. Phys.* 11, 291–293 (2015)
11. Weder, B., et al.: The quantum software lifecycle. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, Nov. 2020, Workshop, pp. 2–9. [Online]. <https://dl.acm.org/doi/10.1145/3412451.3428497>
12. Weigold, M., et al.: Expanding data encoding patterns for quantum algorithms. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pp. 95–101 (2021)
13. Weigold, M., et al.: Data encoding patterns for quantum computing. In: *Proceedings of the 27th Conference on Pattern Languages of Programs*. The Hillside Group (2021)
14. Alexander, C., Ishikawa, S., Silverstein, M.: *A pattern language: towns, buildings, construction*. Oxford University Press, (1977)
15. Zdun, U., Avgeriou, P.: Modeling architectural patterns using architectural primitives. In: *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*, pp. 133–146. ACM (2005)
16. Zdun, U., Hentrich, C., Dustdar, S.: Modeling process-driven and service-oriented architectures using patterns and pattern primitives. *ACM Trans. Web.* 1(3) p. 14-es (2007)
17. Endres, C., et al.: Declarative vs. imperative: two modeling patterns for the automated deployment of applications. In: *Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS 2017)*, pp. 22–27. Xpert Publishing Services (2017)
18. Schumm, D., et al.: A pattern language for costumes in films. In: Kohls, C., Fiesser, A. (eds.) *Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP 2012)*. ACM, New York (2012)

⁵<https://docs.microsoft.com/th-th/quantum/user-guide/machines/resources-estimator>

19. Fehling, C., et al.: A process for pattern identification, authoring, and application. In: Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014). ACM (2014)
20. Aharonov, D., et al.: Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Rev.* 50(4), 755–787 (2008)
21. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge University Press, Cambridge (2010)
22. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. *Phys. Rev. Lett.* 86(22), 5188–5191 (2001)
23. LaRose, R.: Overview and comparison of gate level quantum software platforms. *Quantum.* 3, 130 (2019)
24. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science, pp. 124–134. Ieee (1994)
25. IBM makes quantum computing available on ibm cloud to accelerate innovation (2016). <https://www-03.ibm.com/press/us/en/pressrelease/49661.wss>
26. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. *Phys. Rev. Lett.* 100(16), 15 (2008)
27. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. *Phys. Rev. Lett.* 113(13), 130503 (2014)
28. Wiebe, N., Kapoor, A., Svore, K.M.: Quantum deep learning, arXiv:1412.3489, (2015)
29. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. *Nat. Phys.* 10(9), 631–633 (2014)
30. Salm, M., et al.: About a criterion of successfully executing a circuit in the NISQ era: what $wd \ll 1/e_{\text{eff}}$ really means. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020), pp. 10–13. ACM (2020). Workshop
31. Leymann, F.: Towards a pattern language for quantum algorithms. In: Quantum Technology and Optimization Problems, ser. Lecture Notes in Computer Science (LNCS), vol. 11413, pp. 218–230. Springer International Publishing, Cham (2019)
32. Coplien, J.O.: Software patterns. SIGS Books & Multimedia (1996)
33. Fehling, C., et al.: Cloud computing patterns: fundamentals to design, build, and manage cloud applications. Springer (2014)
34. LaRose, R., Coyle, B., Robust data encodings for quantum classifiers, arXiv:2003.01695, (2020)
35. Leymann, F., Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. *Quant. Sci. Tech.* 5, 1–28 (2020). <https://doi.org/10.1088/2058-9565/abae7d>
36. Grant, E., et al.: Hierarchical quantum classifiers. *NPJ Quant. Inf.* 4(1), 1–8 (2018)
37. Yan, F., Ilyasu, A.M., Venegas-Andraca, S.E.: A survey of quantum image representations. *Quant. Inf. Process.* 15(1), 1–35 (2016)
38. Schuld, M., Sinayskiy, I., Petruccione, F.: The quest for a quantum neural network. *Quant. Inf. Process.* 13(11), 2567–2586 (2014)
39. Templates (2020). <https://pennylane.readthedocs.io/en/stable/introduction/templates.html>
40. Cortese, J.A., Braje, T.M.: Loading classical data into a quantum computer, arXiv:1803.01958, (2018)
41. Prakash, A.: Quantum algorithms for linear algebra and machine learning. Ph.D. dissertation. EECS Department, University of California, Berkeley (2014)
42. Mitarai, K., Kitagawa, M., Fujii, K.: Quantum analog-digital conversion. *Phys. Rev.* 99, 01 (2019)
43. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* 103(15), 150502 (2009)
44. Olver, P.J., Shakiban, C., Shakiban, C.: Applied linear algebra, vol. 1. Springer (2006)
45. Abhijith, J., et al.: Quantum algorithm implementations for beginners. arXiv e-prints, pp. arXiv-1804 (2018)
46. Plesch, M., Brukner, Č.: Quantum-state preparation with universal gate decompositions. *Phys. Rev.* 83(3) (2011)
47. DASKIN, A., GRAMA, A., KAIS, S.: Quantum random state generation with predefined entanglement constraint, arXiv preprint arXiv:1403.0270, (2014)
48. Iten, R., et al.: Introduction to universalqcompiler, ArXiv, vol. abs/1904.01072, 2019
49. ApproxTimeEvolution (2021). <https://pennylane.readthedocs.io/en/stable/code/api/pennylane.templates.subroutines.ApproxTimeEvolution.html>
50. Cleve, R., et al.: Quantum algorithms revisited. *Proc. Math. Phys. Eng. Sci.* 454(1969), 339–354 (1998)
51. Duan, B., et al.: A survey on hhl algorithm: from theory to application in quantum machine learning. *Phys. Lett.* 384(24), 126595 (2020)
52. PhaseEstimation (2021). <https://qiskit.org/documentation/stubs/qiskit.circuit.library.PhaseEstimation.html>
53. Leymann, F.: Towards a pattern language for quantum algorithms. In: First International Workshop, QTOP 2019. Springer, Munich (2019). March 18, 2019, Proceedings
54. Cao, Y., Guerreschi, G.G., Aspuru-Guzik, A.: Quantum neuron: an elementary building block for machine learning on quantum computers, (2017)
55. Paetznick, A., Svore, K.M.: Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries, (2014)
56. Cross, A.W., et al.: Openqasm 3: A broader and deeper quantum assembly language, (2021)
57. Biamonte, J., et al.: Quantum machine learning. *Nature.* 549(7671), 195–202 (2017)
58. Gilliam, A., et al.: Foundational patterns for efficient quantum computing, arXiv:1907.11513, (2019)
59. Huang, Y., Martonosi, M.: Statistical assertions for validating patterns and finding bugs in quantum programs. In: Proceedings of the 46th International Symposium on Computer Architecture, ser. ISCA '19, pp. 541–553. Association for Computing Machinery, New York (2019)
60. Perdrix, S.: Quantum patterns and types for entanglement and separability. *Electron. Notes Theor. Comput. Sci.* 170, 125–138 (2007)
61. Schuld, M., Sweke, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models, arXiv:2008.08605, (2020)
62. Schuld, M., Killoran, N.: Quantum machine learning in feature hilbert spaces. *Phys. Rev. Lett.* 122(4), 040504 (2019)
63. Havlíček, V., et al.: Supervised learning with quantum-enhanced feature spaces. *Nature.* 567(7747), 209–212 (2019)
64. Blencowe, M.: Quantum ram. *Nature.* 468(7320), 44–45 (2010)
65. Giliberto, C., et al.: Quantum machine learning: a classical perspective. *Proc. Math. Phys. Eng. Sci.* 474(2209), 20170551 (2018)
66. Salm, M., et al.: The NISQ analyzer: automating the selection of quantum computers for quantum algorithms. In: Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020), pp. 66–85. Springer International Publishing (2020) [Online]. https://link.springer.com/chapter/10.1007/978-3-030-64846-6_5

How to cite this article: Weigold, M., et al.: Encoding patterns for quantum algorithms. *IET Quant. Comm.* 2(4), 141–152 (2021). <https://doi.org/10.1049/qtc2.12032>