Sublinear Classical-to-Quantum Data Encoding using n-Toffoli Gates

Vittorio Pagni
Institute of Software Technology
German Aerospace Center (DLR)
Sankt Augustin, Germany
University of Cologne
Cologne, Germany
ORCID: 0009-0006-9753-3656

Gary Schmiedinghoff
Institute of Software Technology
German Aerospace Center (DLR)
Sankt Augustin, Germany
ORCID: 0000-0003-2259-7365

Kevin Lively
Institute of Software Technology
German Aerospace Center (DLR)
Sankt Augustin, Germany
ORCID: 0000-0003-2098-1494

Michael Epping
Institute of Software Technology
German Aerospace Center (DLR)
Sankt Augustin, Germany
ORCID: 0000-0003-0950-6801

Michael Felderer
Institute of Software Technology
German Aerospace Center (DLR)
Sankt Augustin, Germany
University of Cologne
Cologne, Germany
ORCID: 0000-0003-3818-4442

2025 IEEE International Conference on Quantum Computing and Engineering (QCE) | 979-8-3315-5736-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/QCE65121.2025.00034

***Abstract*—Quantum state preparation, also known as encoding or embedding, is a crucial initial step in many quantum algorithms and often constrains theoretical quantum speedup in fields such as quantum machine learning and linear equation solvers. One common strategy is amplitude encoding, which embeds a classical input vector of size N=2^n in the amplitudes of an n-qubit register. For arbitrary vectors, the circuit depth typically scales linearly with the input size N, rapidly becoming unfeasible on near-term hardware. We propose a general-purpose procedure with sublinear average depth in N, increasing the window of utility.**

Our amplitude encoding method encodes arbitrary complex vectors of size N=2^n at any desired binary precision using a register with n qubits plus 2 ancillas and a sublinear number of multi-controlled NOT (MCX) gates, at the cost of a probabilistic success rate proportional to the sparsity of the encoded data. The core idea of our procedure is to construct an isomorphism between target states and hypercube graphs, in which specific reflections correspond to MCX gates. This reformulates the state preparation problem in terms of permutations and *binary addition*. The use of MCX gates as fundamental operations makes this approach particularly suitable for quantum platforms such as *ion traps* and *neutral atom devices*. This geometrical perspective paves the way for more gate-efficient algorithms suitable for near-term hardware applications.

***Index Terms*—Quantum algorithm, state preparation, amplitude encoding, Toffoli, reversible computation**

I. Introduction

Quantum computers can prepare and manipulate states within an exponentially large Hilbert space of dimension $N = 2^n$ on an n-qubit register during computation. An exponential speedup is promised by algorithms such as the Harrow-Hassidim-Lloyd (HHL) algorithm [1], [2], used for solving systems of linear equations, and the quantum Fourier transform (QFT) [2], [3]. However, quantum-classical interfaces, i.e., loading classical data into the quantum register and reading out the quantum state, present a critical bottleneck for any algorithms aiming to operate on large input data. Quantum machine learning, in particular, requires massive amounts of data to be embedded into the Hilbert space [4]. In the literature, the process of loading classical data into a quantum computer is variously called *quantum state preparation (QSP)* [5], *encoding* [5], *quantum state loading* [6], or *quantum embedding* [7], [8]. Note that some of these terms have double meaning, such as encoding in the context of quantum data compression or embedding theory in quantum chemistry.

Just as Holevo's theorem constrains the maximum information gain by readout of an n-qubit register [9] likewise, an inefficient QSP can eliminate any theoretical speedup of a quantum algorithm. Particularly in the era of noisy intermediate-scale quantum (NISQ) devices requiring error mitigation, any reduction in circuit depth can mean the difference between useful and useless results [10]–[12]. Thus efficient embedding schemes are crucial on the quest for quantum utility.

One proposed solution are specialized quantum random-access memory (QRAM) [13], [14] units to access stored quantum mechanical states and their superpositions. This approach presents major scaling challenges, some of which are architecture-specific [15], [16], others that arise from more fundamental, physical limits [17], [18]. Even without dedicated QRAM hardware, one can optimize QSP with optimal

This study was funded by the QuantERA grant EQUIP via DFG project 491784278 and by the Federal Ministry for Economics and Climate Action (BMWK) via project ALQU and the Quantum Fellowship Program of DLR.

979-8-3315-5736-2/25/$31.00 ©2025 IEEE
DOI 10.1109/QCE65121.2025.00034

control [19] or efficient gate-based ansatzes, which itself is treated as a circuit-based QRAM in the literature. Different QSP schemes have been proposed in the past, with various strengths and weaknesses. Often, there is a tradeoff between circuit depth and width, i.e., the number of non-simultaneously executable gates and qubits, respectively.

For instance, basis encoding is used to embed classical data points v_i, $i \in [0, \ldots, N-1]$ into quantum states of either the form $\sum_{i=0}^{N-1} |v_i\rangle/\sqrt{N}$ or $\sum_{i=0}^{N-1} |i\rangle|v_i\rangle/\sqrt{N}$. While the corresponding encoding circuits have sublinear depth $\mathcal{O}(\log(N))$, they typically require lineary many qubits $\mathcal{O}(N)$ [6]. On the other extreme, amplitude encoding of the form $\sum_{i=1}^{N} \sqrt{v_i}|i\rangle/\sum_i |v_i|$ only requires $\mathcal{O}(\log_2(N))$ qubits, in principle, but general-purpose amplitude-encoding circuits with $\mathcal{O}(\log(N))$ depth typically require $\mathcal{O}(N)$ ancillary qubits [20]. Angle-based embedding schemes, which use the classical data as gate rotation angles, require at least $\mathcal{O}(N)$ rotation gates [21].

Many more proposals exist using at least $\mathcal{O}(N)$ CNOTs [22]–[25]. Multiple works focus on optimized state preparation schemes, for instance by using low-rank approximations [5], spectral decomposition methods [26] or genetic algorithms [7], [27]. Some proposal achieve high efficiency only for specific states, such as sparse states [25], [28]–[30], uniform states [31], or those with matrix product state representation [32], [33]. Some of these approaches allow for an adjustable tradeoff between circuit depth and width [34] or use probabilistic approaches [35], [36].

Yet other works rely on state preparation with variational algorithms [37]–[40], which introduces significant preprocessing overhead for general states without efficient representations such as matrix product states. In particular, the open problem of barren plateaus may introduce $\mathcal{O}(N)$ optimization overhead. A recent work encodes classical floating point values into quantum mechanical amplitudes to solve nonlinear partial differential equations [41], utilizing two ancilla qubits and $\mathcal{O}(\log(N))$ gates with a finite success probability. This is achieved through a binary expansion of the input vector and by performing a series of controlled rotations, which are tools that we also employ in this work.

Here we show how the L-bit binary expansion of a generic, input vector of length N can be loaded into the amplitudes of a non error-corrected quantum state of $n = \log_2(N)$ bits with 2 ancilla qubits by a sequence of multi-controlled NOT (MCX) gates, which are particularly suitable for NISQ platforms such as ion traps or neutral atom devices [42] [43] [44]. The logical flow of our approach is summarized in Figure 1.

We begin by introducing the pre-processing Algorithm 1 in Sec. II and describing the general procedure of the full Algorithm 2 in Sec. III. The concrete properties of the MCX circuit blocks are introduced in Sec. IV and reformulated in the language of Kronecker decompositions to exploit binary operations for numerical efficiency and hypercube graphs [45] for visualization and intuition. We explain Subroutines 1 and 2 that we use for finding efficient MCX decompositions in Sec. V and compare the performance for various input data against the Qiskit circuit in Sec. VI, where for each encoding circuit we then analyze a shallower version (core) with a success probability that depends on the input and a longer version (full) which incorporates amplitude amplification to boost the success probability, before discussing the conclusions of our research in Sec. VII.

II. Classical pre-processing

The first step consists of a classical pre-processing that, given a vector $\boldsymbol{v} \in \mathbb{R}^N$ and a bit precision $L \in \mathbb{N}$ as input, approximates the entries $\boldsymbol{v}$ with a L-bit binary expansion encoded in a binary matrix $\mathbf{B} \in \mathbb{F}_2^{N \times L}$. This pre-processing scheme has already been presented in our previous work [46]. We use an intermediate rescaled angle vector $\boldsymbol{\theta}$, defined by

$$\theta_i = \arcsin\left(\frac{v_i}{\|\boldsymbol{v}\|_\infty}\right) \Big/ \frac{\pi}{2} \in [-1, 1] \tag{1}$$

with $i \in \{0 \ldots N-1\}$. The signed L-bit binary representation (i.e. line 5 in Algorithm 1) of each θ_i is stored in the i^{th} row $\boldsymbol{B}_{i,:}$ of $\mathbf{B}$, where we use slice indexing notation (:) to indicate all elements in the row. Shortly, we will use the signed L-bit representation to encode an approximated vector $\boldsymbol{w}$, which converges to $\boldsymbol{v}$ as L increases. For complex input vectors, one encodes the modulus and phases separately [46].

Algorithm 1 Classical Pre-Processing

1: **Input:**
- real vector $\boldsymbol{v} \in \mathbb{R}^N$, $N = 2^n$
- encoding precision $L \in \mathbb{N}$

2: **Output:** binary matrix $\mathbf{B} \in \mathbb{F}_2^{N \times L}$.
3: **for** each $v_i \in \boldsymbol{v}$ **do**
4: $\quad \theta_i \coloneqq \arcsin\left(v_i/\|\boldsymbol{v}\|_\infty\right)/\frac{\pi}{2}$
5: $\quad$ set $B_{i,j}$ such that $\theta_i = (-1)^{B_{i,0}} \sum_{j=1}^{L-1} 2^{-j} B_{i,j}$
6: **end for**
7: **return** $\mathbf{B}$

A. Example

If we choose $n = 3$, $N = 2^n = 8$, $L = 5$, and

$$\boldsymbol{v} = \frac{1}{\sqrt{1265}}(15, 13, 10, -11, 12, -15, 5, 16)\ , \tag{2}$$

then the approximating vector is (here truncated to the second decimal digit)

$$\boldsymbol{w} = (0.43, 0.36, 0.26, -0.29, 0.33, -0.43, 0.13, 0.46)\ , \tag{3}$$

the angle vector $\boldsymbol{\theta}$ and the binary matrix $\mathbf{B}$ are

$$\boldsymbol{\theta} = (0.77,\ 0.60,\ 0.43,\ -0.48,\ 0.54,\ -0.77,\ 0.20,\ 1) \tag{4}$$

and

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}\ . \tag{5}$$

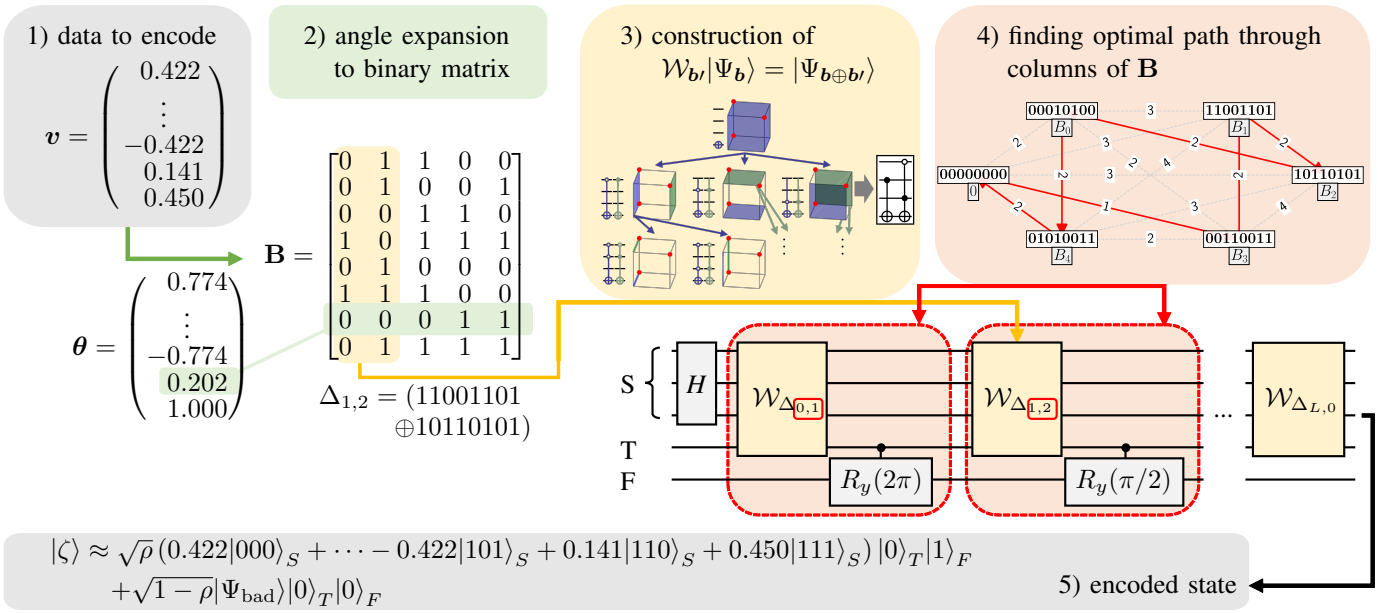

Fig. 1: Overview of the full algorithm. Input data (step 1) is transformed to renormalized angles whose binary expansion is stored in a matrix (step 2, see Sec. II). Each column can be encoded with a circuit containing fully controlled MCX (see Sec. III and Sec. IV), which we optimize using the tree algorithm (step 3, see Sec. V). We further reduce the cost by constructing an optimized order of applying the encoding layers (step 4, see Fig. 5) to produce the encoded state (step 5, see (14)).

III. General Procedure

The full algorithm is sketched in Fig. 1. In this Section, we will gradually build up the necessary concepts and conclude with the final Algorithm 2, which utilizes the subroutines that are discussed in Sec. V.

A. Encoding a single vector element

The binary matrix $\mathbf{B}$ associated with the input vector $\boldsymbol{v}$ has N rows, one for each entry in $\boldsymbol{v}$, and L columns, which correspond to the L bits in the binary representation of the $\boldsymbol{\theta}$ angle vector entries. A key aspect of this encoding is that we can reconstruct each real entry v_i by using L controlled rotations with decreasing angle. These rotation are either performed or skipped according to the L bits in the corresponding row $\boldsymbol{B}_{i,:}$ of $\mathbf{B}$.

Going back to the example, the first row is

$$\boldsymbol{B}_{0,:} = (0,1,1,0,0)\,. \tag{6}$$

If we apply L sequential R_y rotations conditioned on the entries of $\boldsymbol{B}_{0,:}$ with angles

$$\phi_l = \begin{cases} 2\pi & , l = 0 \\ \pi/2^l & , l \in \{1,\dots,L-1\} \end{cases} \tag{7}$$

to a qubit whose initial state is $|0\rangle$, we obtain

$$\begin{aligned} |\Psi_0\rangle &= \prod_{l=0}^{L-1} \Big(R_y(\phi_l)\Big)^{B_{0,l}} |0\rangle \\ &= \cos\left(\frac{1}{2}\sum_{l=0}^{L-1} B_{0,l}\cdot\phi_l\right)|0\rangle + \sin\left(\frac{1}{2}\sum_{l=0}^{L-1} B_{0,l}\cdot\phi_l\right)|1\rangle. \end{aligned}$$

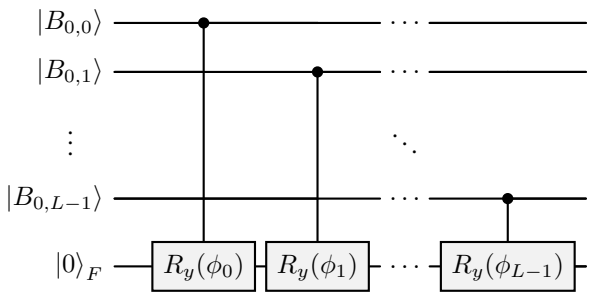

Fig. 2: An encoding layer for the first row of the matrix $\mathbf{B}$.

Exploiting the definition of ϕ_i and $\mathbf{B}$ we get

$$\begin{aligned} |\Psi_0\rangle = (-1)^{B_{0,0}} &\cos\left(\arcsin\left(\left|\frac{v_0}{v_\infty}\right|\right)\right)|0\rangle \\ &+ (-1)^{B_{0,0}} \sin\left(\arcsin\left(\left|\frac{v_0}{v_\infty}\right|\right)\right)|1\rangle \end{aligned}$$

and finally

$$|\Psi_0\rangle = \operatorname{sign}(v_0)\left(\sqrt{1-\left(\frac{v_0}{v_\infty}\right)^2}|0\rangle + \frac{v_0}{v_\infty}|1\rangle\right), \tag{8}$$

thereby encoding the normalized value of v_0 into the amplitude of the $|1\rangle$ state. Since all R_y commute, we can think of each $R_y(\phi_l)$ as encoding a specific precision level of the value v_i/v_∞: the sign level $(-1)^B_{i,0}$, and each binary decimal level $2^{-j}B_{i,j}$. The corresponding circuit is shown in Fig. 2.

B. Encoding an entire vector in parallel

Now that we know how to exploit R_y rotations in order to encode a single entry v_i of the input vector, corresponding to a single row of $\mathbf{B}$, into the amplitude of the $|1\rangle$ state, we can parallelize this procedure over the whole vector using quantum superposition. The central idea is to consider the columns $\boldsymbol{B}_{:,l}, l \in \{0, \ldots, L-1\}$ sequentially and at each step encode all entries of the column in parallel. The strategy we present here involves three registers:

- a SYSTEM (S) register with n qubits,
- a TARGET (T) register with one qubit,
- a FLAG (F) register with one qubit.

The algorithm begins with applying a layer of Hadamard gates to the SYSTEM register, creating the uniform superposition

$$|\Psi_{\mathbf{0}}\rangle_{S,T} := \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |0\rangle_T. \tag{9}$$

Looking at this state, we can consider the basis state $|i\rangle$ on the SYSTEM register, ranging from 0 to $N-1$, as an index for the state of the TARGET qubit. This allows us to see (9) as encoding a specific binary vector of length N, which contains only zeros. We now consider acting on this state with a unitary operator $\mathcal{W}_{\boldsymbol{b}}$ as a function of an arbitrary vector $\boldsymbol{b} \in \mathbb{F}_2^N$ such that it acts as

$$\mathcal{W}_{\boldsymbol{b}} |\Psi_{\mathbf{0}}\rangle = |\Psi_{\mathbf{0}\oplus\boldsymbol{b}}\rangle = |\Psi_{\boldsymbol{b}}\rangle\,, \tag{10}$$

where $|\Psi_{\boldsymbol{b}}\rangle$ is the uniform superposition

$$|\Psi_{\boldsymbol{b}}\rangle_{S,T} := \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |b_i\rangle_T, \tag{11}$$

in which the TARGET qubit is entangled with the SYSTEM register. More generally given a state as in (11) and a binary vector $\boldsymbol{b}' \in \mathbb{F}_2^N$ we want

$$\mathcal{W}_{\boldsymbol{b}'} |\Psi_{\boldsymbol{b}}\rangle = |\Psi_{\boldsymbol{b}\oplus\boldsymbol{b}'}\rangle. \tag{12}$$

For now we assume that such an operator $\mathcal{W}_{\boldsymbol{b}'}$ is given as a black box. We later describe how to implement the $\mathcal{W}$ operators in terms of MCX gates in Sec. IV. Since, as it turns out, this decomposition is not unique, we explain our algorithm for finding efficient implementations in Sec. V.

When we act with a controlled $R_y(\phi)$ rotation that uses the qubit TARGET as control and the FLAG as target on the state in (11) tensored with the FLAG in the zero state, we get

$$\begin{aligned} &CR_y(\phi_j, T \to F)|\Psi_{\boldsymbol{b}}\rangle_{S,T}|0\rangle_F \\ &= \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |b_i\rangle_T (\cos(b_i\phi_j)|0\rangle_F + \sin(b_i\phi_j)|1\rangle_F) \\ &= \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |b_i\rangle_T \Big(R_y(b_i\phi_j)|0\rangle_F \Big). \end{aligned} \tag{13}$$

From this equation we see how the binary values b_i tell us which indices i and therefore which amplitudes are affected by the given rotation $R_y(\phi_j)$.

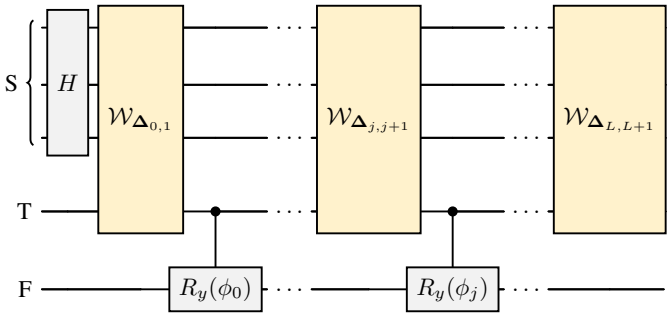

Fig. 3: Linear-path circuit that encodes the state $|\zeta\rangle$ starting from the initial state $|0\rangle$. It alternates the L shift gates $\mathcal{W}_{\boldsymbol{\Delta}_{j,j+1}}$ and the controlled rotations $CR_y(\phi_j, T \to F)$ for $j \in \{0, 1, \ldots, L-1\}$ followed by the final disentangling shift $\mathcal{W}_{\boldsymbol{\Delta}_{L,L+1}}$. The costs $|\mathcal{W}_{\boldsymbol{\Delta}}|$ can be reduced by using a better permutation σ of the $(\mathcal{W}, R_y)$ layers. The full protocol is described in Algorithm 2.

The final state we want to prepare is

$$\begin{aligned} |\zeta\rangle =& \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |0\rangle_T \left(R_y \left(\sum_{j=0}^{L-1} \phi_j B_{i,j} \right) |0\rangle_F \right) \\ =& \sqrt{\rho} |\Psi_{\text{good}}\rangle |0\rangle_T |1\rangle_F + \sqrt{1-\rho} |\Psi_{\text{bad}}\rangle |0\rangle_T |0\rangle_F \end{aligned} \tag{14}$$

where

$$|\Psi_{\text{good}}\rangle = \sum_{i=0}^{N-1} w_i |i\rangle_S. \tag{15}$$

$|\zeta\rangle$ is a weighted superposition of the desired state that encodes the approximating vector $\boldsymbol{w}$, $|\Psi_{\text{good}}\rangle$, entangled with the state $|1\rangle$ on the FLAG register and of an undesired state $|\Psi_{\text{bad}}\rangle$, associated to the state $|0\rangle$ on the FLAG. By measuring the FLAG qubit, we know if the procedure succeeded, which happens with a probability given by the data density parameter

$$\rho = \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{v_i}{\|\boldsymbol{v}\|_\infty} \right)^2 \in \left[\frac{1}{N}, 1 \right], \tag{16}$$

whose average value over uniform random inputs decreases as slowly as $\log(N)^{-1}$ [46]. If the FLAG qubit measures zero, the state must be discarded and the encoding repeated. While amplitude amplification can increase success probability, on near-term devices with limited circuit depth, using the shallow core encoding without amplification may be preferable.

In order to prepare the state $|\zeta\rangle$, we use the circuit in Fig. 3. We take the cost of preparing the superpositions of the SYSTEM and TARGET registers with the circuit $\mathcal{W}_{\boldsymbol{b}}$ to be the number of MCX gates in the decomposition found by Subroutine 1. We denote the cost by $|\mathcal{W}_{\boldsymbol{b}}|$, motivated by the cardinality of the set of gates in the decomposition. It turns out that the costs can be reduced by encoding the columns of $\mathbf{B}$ in a different order. To that end, we introduce the path matrix $\mathbf{P} \in \mathbb{F}_2^{N,L+2}$ containing a sequence of $L+2$ binary vectors

$$\mathbf{P} := (\mathbf{0}, \boldsymbol{B}_{:,0}, \ldots, \boldsymbol{B}_{:,L-1}, \mathbf{0}), \tag{17}$$

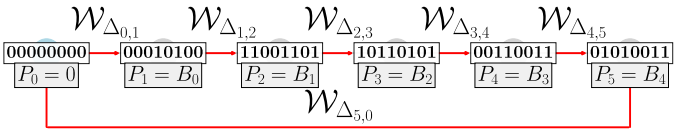

Fig. 4: Example of the path P for the input binary matrix in (5). The nodes are represented by the binary vectors, which are the columns of $\mathbf{B}$ plus the initial and final all-zero state, that need to be encoded into the superposition state of the three registers.

which is $\mathbf{B}$, padded with the all-zero column as the initial and final columns. We also use a padded $L+1$ angle vector

$$\mathbf{\Phi} := (0, \phi_0, \dots, \phi_{L-1})\,. \tag{18}$$

The XOR of every two consecutive columns in $\mathbf{P}$ is

$$\mathbf{\Delta}_{j,j+1} := \boldsymbol{P}_{:,j} \oplus \boldsymbol{P}_{:,j+1}, j \in \{0, \dots, L\}\,. \tag{19}$$

In the circuit, we alternate the L shift operators $\mathcal{W}_{\mathbf{\Delta}_{j,j+1}}$ acting on the SYSTEM and TARGET registers with the L controlled rotations $CR_y(\phi_j)$ targeting the FLAG qubit. Finally, we apply the shift operator $\mathcal{W}_{\mathbf{\Delta}_{L-1,L}}$ to disentangle the TARGET register from the others. With the first and last components of $\mathbf{P}$ being zero vectors, we trivially have

$$\mathcal{W}_{\Delta_{0,1}} = \mathcal{W}_{\mathbf{0}\oplus\boldsymbol{B}_{:,0}} = \mathcal{W}_{\boldsymbol{B}_{:,0}} \tag{20}$$

$$\mathcal{W}_{\Delta_{L,L+1}} = \mathcal{W}_{\boldsymbol{B}_{:,L}\oplus\mathbf{0}} = \mathcal{W}_{\boldsymbol{B}_{:,L}}\,. \tag{21}$$

This sequence of operators can be seen as a path along a graph whose nodes are the binary columns of $\mathbf{P}$ that can be encoded into the superposition in (11), as shown in Fig. 4.

At each application of $\mathcal{W}_{\mathbf{\Delta}_{j,j+1}}$, the quantum state transitions from one node to the next in Fig. 4, effectively shifting from a superposition state encoding column $\boldsymbol{P}_{:,j}$ to one encoding $\boldsymbol{P}_{:,j+1}$. Each controlled rotation $CR_y(\phi_j)$ adds the term $\boldsymbol{P}_{i,j+1}\phi_j$ for each state $|i\rangle$ of the SYSTEM in the superposition. We can write the intermediate state of the SYSTEM as

$$|\zeta_{j,m}\rangle := \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |P_{i,j}\rangle_T \left(R_y\left(\sum_{l=0}^{m} \phi_l P_{i,l} \right) \right) |0\rangle_F. \tag{22}$$

so that at the beginning of the j^{th} encoding step we have the state $|\zeta_{j,j}\rangle$. In this formalism, the initial state of the SYSTEM after the Hadamard layer described in (9) can be seen as encoding the all-zero column $\boldsymbol{P}_{:,0}$ and having only $0 \cdot P_{i,0} = 0$ in the sum of angles, which corresponds to

$$|\zeta_{0,0}\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |0\rangle_T |0\rangle_F = |\Psi_{\mathbf{0}}\rangle_{S,T} |0\rangle_F. \tag{23}$$

Moreover, we can write our final target state from (14) as $|\zeta\rangle = |\zeta_{L+1,L}\rangle$. During the j$^{\text{th}}$ encoding step we increment the first index by $\mathcal{W}_{\mathbf{\Delta}_{j,j+1}} |\zeta_{j,j}\rangle = |\zeta_{j+1,j}\rangle$ and then increment the second index by $CR_y(\Phi_{j+1}, T \to F)|\zeta_{j+1,j}\rangle = |\zeta_{j+1,j+1}\rangle$. As long as we begin and end with the all-zero columns $\boldsymbol{P}_{:,0}$ and $\boldsymbol{P}_{:,L+1}$, we can consider different permutations σ of the intermediate L columns of $\mathbf{P}$. These permutations correspond to the various paths in Fig. 5 and, due to the commutativity in the sum of angles, all yield the same final state $|\zeta\rangle$. Each edge of the path $(\boldsymbol{P}_{:,i}, \boldsymbol{P}_{:,i})$ is assigned with a cost $|\mathcal{W}_{\mathbf{\Delta}_{i,j}}|$, which we take to be the amount of MCX gates in $\mathcal{W}_{\mathbf{\Delta}_{i,j}}$. Finding the optimal ordering σ that minimizes the total cost

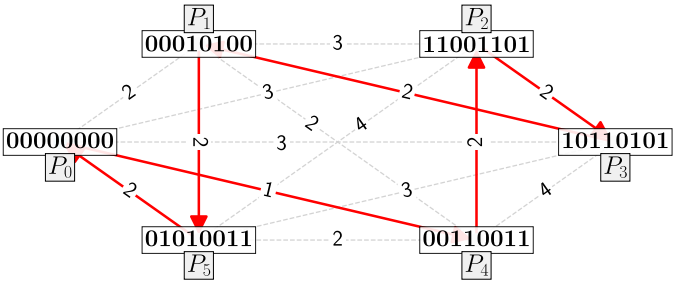

Fig. 5: Complete graph representing all the possible encoding paths along the columns of $\mathbf{P}$, starting and ending with the all-zero columns $\boldsymbol{P}_{:,0} = \boldsymbol{P}_{:,L+1}$, Each edge $(\boldsymbol{P}_{:,i}, \boldsymbol{P}_{:,j})$ is associated with a vector $\mathbf{\Delta}_{i,j} = \boldsymbol{P}_{:,i} \oplus \boldsymbol{P}_{:,j}$ and with a cost $|\mathcal{W}_{\mathbf{\Delta}_{i,j}}|$, which is the amount of MCX in the decomposition of the shift operator. We see in red the optimal path given by the permutation $\sigma = (4)(2)(3)(1)(5)$ for the example in (5).

$$D(\sigma) := \sum_{j=0}^{L} |\mathcal{W}_{\mathbf{\Delta}_{\sigma(j),\sigma(j+1)}}| \tag{24}$$

is equivalent to solving a travelling salesman problem (TSP). The cost function $|\mathcal{W}_{\mathbf{\Delta}_{i,j}}|$ can be generalized, for example by assigning weights to MCX gates depending on the number of involved qubits. Algorithm 2 describes the whole procedure.

Algorithm 2 MCX Amplitude Encoder

1: **Input:**
- binary encoding matrix $\mathbf{B} \in \mathbb{F}_2^{N \times L}$
- $|0\rangle$-initialized quantum registers SYSTEM, TARGET, FLAG

2: **Output:**

$$|\zeta\rangle = \sqrt{\rho} \sum_{i=0}^{N-1} w_i |i\rangle_S |0\rangle_T |1\rangle_F + \sqrt{1-\rho} |\Psi_{\text{bad}}\rangle_S |0\rangle_T |0\rangle_F$$

3: $|\zeta_{0,0}\rangle = H^{\otimes n} \otimes I \otimes I |0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_S |0\rangle_T |0\rangle_F$
4: $\mathbf{P} = (\mathbf{0}, \boldsymbol{B}_{:,0}, \dots, \boldsymbol{B}_{:,L-1}, \mathbf{0})$
5: call Subroutine 1 from Sec. V to get costs $|\mathcal{W}_{\mathbf{\Delta}_{j,l}}|$
6: find best permutation $\sigma = \text{solveTSP}(\mathbf{P}, \{|\mathcal{W}_{\mathbf{\Delta}_{j,l}}|\})$
where $\sigma(0) = 0, \sigma(L+1) = L+1$
7: $\mathbf{\Delta}_{l,l+1} = \boldsymbol{P}_{:,\sigma(l)} \oplus \boldsymbol{P}_{:,\sigma(l+1)}, l \in \{0, \dots, L\}$
8: **for** $l \in \{0, \dots, L-1\}$ **do**
9: $\quad |\zeta_{l+1,l}\rangle = \mathcal{W}_{\Delta_{l,l+1}} |\zeta_{l,l}\rangle$
10: $\quad |\zeta_{l+1,l+1}\rangle = CR_y(\Phi_{\sigma(l+1)} = \phi_l, T \to F) |\zeta_{l+1,l}\rangle$
11: **end for**
12: disentangle TARGET register $\mathcal{W}_{\Delta_{L,L+1}} |\zeta_{L,L}\rangle = |\zeta_{L+1,L}\rangle$
13: **return** $|\zeta\rangle = |\zeta\rangle_{L+1,L}$

IV. Implementation of $\mathcal{W}$ with MCX gates

In this section, we explain how each $\mathcal{W}_{\boldsymbol{b}}$ can be implemented with MCX gates. The basic decomposition and important properties of the optimization problem are described in Sec. IV-A. We then introduce two equivalent, but useful ways to view the problem: a formulation as finding the most efficient way to address vertices on a hypercube graph in Sec. IV-B, which provides a useful visual interpretation, and a formulation as finding a minimal Kronecker decomposition in Sec. IV-C, which is particularly suited for numerical implementations with bit-wise operations.

We aim to construct a $\mathcal{W}_{\llcorner}$ which has the properties in (10) and (12). Looking at the equations (9) and (11), we see that we can address every b_i, $i \in \{0, 1, \ldots, 2^n - 1\}$ in the superposition individually by means of one of the 2^n possible fully controlled MCX. As we will show in the following sections, using less than fully controlled MCX allows to simultaneously target specific subsets of the computational basis states, making the basis overcomplete and therefore allowing for shorter decompositions. These pictures will aid us in formulating the optimization Subroutine 1 in Sec. V, which finds a shallow MCX circuit that implements $\mathcal{W}_{\boldsymbol{b}}$.

A. Properties of $\mathcal{W}_{\boldsymbol{b}}$

It turns out that each of the encoding circuits $\mathcal{W}_{\boldsymbol{b}}$ for a binary vector $\boldsymbol{b}$, which must fulfil (12), can be implemented by a product of MCX gates

$$\mathcal{W}_{\boldsymbol{b}} = \prod_{j=0}^{M-1} \mathrm{MCX}(\boldsymbol{c}_j) \tag{25}$$

which take the form

$$\mathrm{MCX}(\boldsymbol{c}) = \mathbb{P}_{\boldsymbol{c},S} \otimes X_T + (\mathbb{1}_S - \mathbb{P}_{\boldsymbol{c},S}) \otimes \mathbb{1}_T \,, \tag{26}$$

where $\boldsymbol{c} \in \{0, 1, \mathrm{I}\}^n$ is the control string of the MCX, where we use the symbol I to indicate the uncontrolled qubit indices, $\mathbb{P}_{\boldsymbol{c},S}$ is the projector to the corresponding subspace. Every I symbol in $\boldsymbol{c}$ indicates that $\mathbb{P}_{\boldsymbol{c}}$ acts as the identity on the correspondent qubit. For example, for $n = 3$

$$\mathbb{P}_{00\mathrm{I},S} = |00\rangle\langle 00| \otimes \mathbb{1} = |000\rangle\langle 000| + |001\rangle\langle 001| \tag{27}$$

The most naive implementation of (25) uses fully controlled MCX and can be constructed by using the binary positions $\{\boldsymbol{\nu} \,|\, b_{\boldsymbol{\nu}} = 1\}$ of each 1 in $\boldsymbol{b}$ as binary control strings $\{0, 1\}^n$, see Fig. 6. To reduce the number M of MCX gates, one must utilize those that only use a subset of the SYSTEM qubits as control, which means trying to use $\boldsymbol{c}_j$ strings with the largest possible number of I entries, denoted $\#_{\mathrm{I}}$.

The decomposition (25) is therefore not unique and finding the one that minimizes the amount M of MCX gates is in general an NP-hard problem, see Sec. IV-C. In fact, there are

$$2^{(n-m)} \binom{n}{m} \tag{28}$$

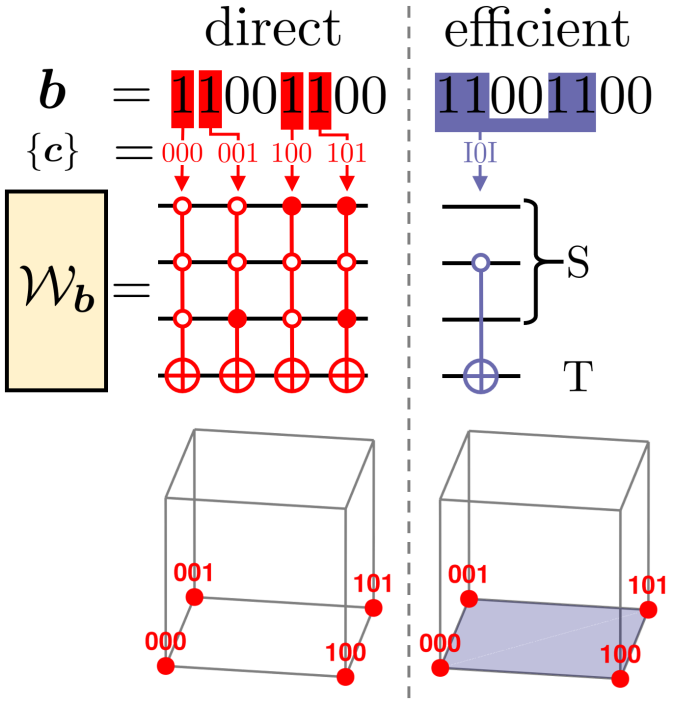

Fig. 6: Two implementations of an exemplary $\mathcal{W}_{\boldsymbol{b}}$ circuit for the binary vector $\boldsymbol{b} = (1, 1, 0, 0, 1, 1, 0, 0)$, which is shown as a bit string for brevity. The direct implementation (left, red) uses four fully controlled $\mathrm{MCX}(\boldsymbol{c}_j, 3)$ with control strings $\{\boldsymbol{c}_j\} = \{000, 001, 100, 101\}$ and the efficient implementation (right, blue) requires only one single-controlled $\mathrm{MCX}(\mathrm{I0I}, 3)$ with $\boldsymbol{c} = \mathrm{I0I}$. At the bottom we show the corresponding graph representations (see Sec. IV-B), where the control strings containing no (two) I symbols are visualized as nodes (faces) of the cube.

ways to choose m control bits with either zero-control or one-control on n bits. The total amount of potential MCX is equivalent to the number of possible strings $\boldsymbol{c} \in \{0, 1, \mathrm{I}\}^n$,

$$\sum_{m=0}^{n} 2^{(n-m)} \binom{n}{m} = 3^n. \tag{29}$$

Finding an efficient decomposition of the operator $\mathcal{W}$ as in (25), with minimal M, is equivalent to identifying a minimal set of control strings $\{\boldsymbol{c}_k\}$ that fully specify the decomposition in terms of MCX gates. More generally, M may represent the total cost of implementing $\mathcal{W}$, where each MCX gate in the decomposition is assigned a customizable cost, allowing for weighted optimization. The order of strings $\boldsymbol{c}_k$ is irrelevant because all MCX gates with the same target commute, and all control strings in $\{\boldsymbol{c}_k\}$ must be unique for an optimal decomposition, because $\mathrm{MCX}(\boldsymbol{c})^2 = \mathrm{I}$. Any two MCX whose control strings differ only in a single position can be joined

$$\mathrm{MCX}(\ldots 0 \ldots)\mathrm{MCX}(\ldots 1 \ldots) = \mathrm{MCX}(\ldots \mathrm{I} \ldots) \,, \tag{30a}$$
$$\mathrm{MCX}(\ldots 0 \ldots)\mathrm{MCX}(\ldots \mathrm{I} \ldots) = \mathrm{MCX}(\ldots 1 \ldots) \,, \tag{30b}$$
$$\mathrm{MCX}(\ldots 1 \ldots)\mathrm{MCX}(\ldots \mathrm{I} \ldots) = \mathrm{MCX}(\ldots 0 \ldots) \,. \tag{30c}$$

In the same way, any MCX can be split into two MCXs. Using the relations (30), one can in principle manipulate the set of control strings $\{\boldsymbol{c}_k\}$ to find a more efficient decomposition (25). The properties of the MCX gates directly transfer to the $\mathcal{W}$ operator

$$\mathcal{W}_{\boldsymbol{b}}^{\dagger} = \mathcal{W}_{\boldsymbol{b}} = \mathcal{W}_{\boldsymbol{b}}^{-1} \tag{31a}$$

$$[\mathcal{W}_{\boldsymbol{b}}, \mathcal{W}_{\boldsymbol{b}'}] = 0 \ , \tag{31b}$$

$$\mathcal{W}_{\boldsymbol{b}}\mathcal{W}_{\boldsymbol{b}'} = \mathcal{W}_{\boldsymbol{b}\oplus\boldsymbol{b}'} \ . \tag{31c}$$

B. Hypercube graph representation

We now introduce an equivalent geometric description of the MCX decomposition (25) in terms of a hypercube graph $G(\{\boldsymbol{c}_j \,|\, j \in \{0,\dots,M-1\}\})$, which is useful for visualization. Generally, for any n-dimensional binary $\boldsymbol{b}$ we start by drawing the n-dimensional hypercube graph containing all nodes $\{0,1\}^n$. In this graph, two nodes have an edge between them if their Hamming distance is 1. Next, we collect the binary positions $\{\boldsymbol{\nu} \,|\, b_{\boldsymbol{\nu}} = 1\}$ of every 1 appearing in $\boldsymbol{b}$ and mark the corresponding nodes in the graph. A simple example for $n = 3$ can be seen on the left side of Fig. 6.

In this graph picture, each node corresponds to a control string $\boldsymbol{c}$ containing no identity symbols I. A control string with a single I can be visualized as an edge and strings with two Is correspond to faces (see right side of Fig. 6). More generally, any control string $\boldsymbol{c}$ with Is appearing at $\#_{\mathrm{I}}$ positions can be visualized as sub-hypercubes of dimension $n - \#_{\mathrm{I}}$. The transformation (30) on the MCX-level correspond to joining subgraphs, e.g., two nodes into an edge, two edges into a face, or more generally, joining two $(\#_{\mathrm{I}}-1)$-dimensional sub-hypercubes into a $\#_{\mathrm{I}}$-dimensional sub-hypercube. An example of this can be seen in Fig. 6, where four nodes are joined into a face, which corresponds to replacing four fully-controlled MCX by a single MCX with two I symbols.

This visualization can be useful to get an intuitive understanding of efficient decompositions. For instance, Fig. 7 displays the graph of an efficient MCX decomposition $\{c_j\}_{\text{efficient}} = \{0\mathrm{III}, \mathrm{I1I1}\}$ for $\boldsymbol{b} = 1111101000000101$ in $n = 4$ dimensions that exploits the involution $\mathrm{MCX}(\boldsymbol{c})^2 = \mathrm{I}$ by including the same SYSTEM states in multiple control strings. This solution may not be obvious when starting from $\{\boldsymbol{c}_j\}_{\text{direct}} = \{\boldsymbol{\nu} \,|\, b_{\boldsymbol{\nu}} = 1\}$, but is straightforward to see when drawing the graph for $\boldsymbol{b}$. This example shows that sometimes, optimal solutions cannot be found by simply joining control strings to fewer control as in (30a), i.e., only searching for sub-hypercubes embedded in the subgraph $G(\{c_j\}_{\text{direct}})$.

C. Kronecker decomposition representation

As a third option, we can reformulate the problem of finding an MCX-decomposition (25) with minimal number M of MCX into the problem of finding a minimal Kronecker decomposition, which allows for an efficient numerical implementation with bit-wise operations. First, we associate each

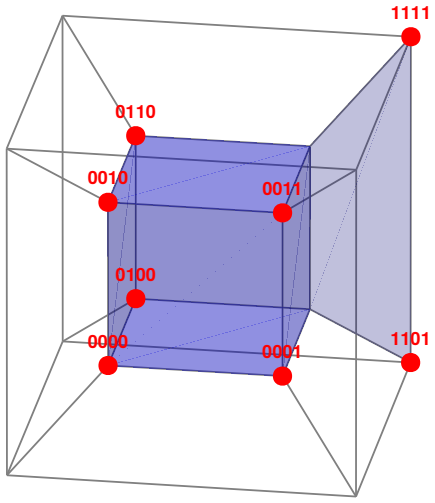

Fig. 7: Visualization in $n = 4$ dimensions of an efficient $\mathcal{W}_{\boldsymbol{b}}$ implementation for $\boldsymbol{b} = 1111101000000101$. The direct implementation uses eight fully-controlled MCX with control strings $\{c_j\}_{\text{direct}} = \{0000, 0001, 0010, 0011, 0100, 0110, 1101, 1111\}$. An efficient implementation uses only two MCXs $\{c_j\}_{\text{efficient}} = \{0\mathrm{III}, \mathrm{I1I1}\}$, i.e., the central cube 0III and the right-rear face I1I1. This way, both gates include control on the SYSTEM states 0101 and 0111 which cancels out.

control string $\boldsymbol{c} \in \{0,1,\mathrm{I}\}^n$ with a binary vector $\boldsymbol{b}_{\boldsymbol{c}} \in \mathbb{F}_2^{2^n}$ via the following mapping

$$\boldsymbol{b}_{\boldsymbol{c}} := \bigotimes_{i=0}^{n-1} \boldsymbol{t}(c_i), \quad \boldsymbol{t}(c_i) = \begin{cases} (1,0) & \text{if } c_i = 0 \\ (0,1) & \text{if } c_i = 1 \\ (1,1) & \text{if } c_i = \mathrm{I} \end{cases} \tag{32}$$

expresses the binary vector $\boldsymbol{b}_{\boldsymbol{c}}$ as a tensor product of 2-dimensional binary vectors in $\{(1,0),(0,1),(1,1)\}$. Every tensor product corresponds to a single MCX as

$$\mathcal{W}_{\boldsymbol{b}_{\boldsymbol{c}}} = \mathrm{MCX}(\boldsymbol{c}) \ . \tag{33}$$

A general binary vector $\boldsymbol{b} \in \mathbb{F}_2^N$ can be decomposed into Kronecker products

$$\boldsymbol{b} = \bigoplus_{j=0}^{M-1} \boldsymbol{b}_{\boldsymbol{c}_j} \ , \tag{34}$$

which is equivalent to the MCX decomposition (25). Thus, finding a minimal MCX decomposition of $\mathcal{W}_{\boldsymbol{b}}$ is equivalent to finding the shortest Kronecker decomposition, which is known to be NP-hard [47], [48]. The advantage of this formulation is that the joining and splitting operations (30) can be expressed as binary additions (XOR operations) on the tensor products $\boldsymbol{b}_{\boldsymbol{c}}$. Thus, in Sec. V we will formulate our algorithm in terms of Kronecker decompositions.

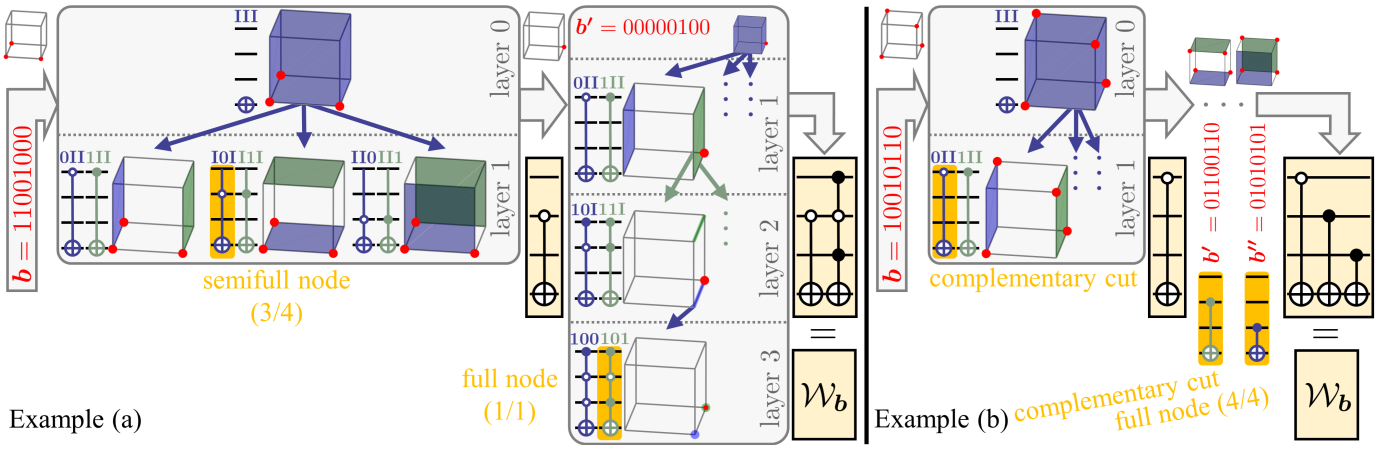

Fig. 8: Visualization of Subroutine 1, which produces a MCX-circuit $\mathcal{W}_{\boldsymbol{b}}$, on two exemplary inputs (a) $\boldsymbol{b} = 11001000$ and (b) $\boldsymbol{b} = 10010110$. In both cases, the tree is traversed from the top layer 0 (no control) downwards (layer l contains MCXs with l control qubits) until tree nodes are encountered that are full or semi-full or that have a complementary bi-partition. The tree algorithm is repeated until all necessary MCXs are found. Some branches and trees are omitted for brevity.

V. Algorithm for efficient construction of $\mathcal{W}_{\boldsymbol{b}}$

Subroutine 1 construct_$\mathcal{W}_{\boldsymbol{b}}$ (Kronecker Decomposition)

Input: binary vector $\boldsymbol{b} \in \mathbb{F}_2^N, N = 2^n$
Output: control strings $C_{\mathcal{W}_{\boldsymbol{b}}} \subset \{0,1,\mathrm{I}\}^n$ for an efficient decomposition $\boldsymbol{b} = \bigoplus_{j=0}^{M-1} \boldsymbol{b}_{\boldsymbol{c}_j}$, see (34)
$C_{\mathcal{W}_{\boldsymbol{b}}} = \emptyset$
while $\boldsymbol{b}$!= **0** **do**
  $C_{\text{candidates}}$ = findNextControlStrings($\boldsymbol{b}$)
  C_{selected} = findMaxIndependentSet($C_{\text{candidates}}$)
  $\boldsymbol{b} = \boldsymbol{b} \bigoplus \{\boldsymbol{b}_{\boldsymbol{c}} \,|\, \boldsymbol{c} \in C_{\text{selected}}\}$
  $C_{\mathcal{W}_{\boldsymbol{b}}} = C_{\mathcal{W}_{\boldsymbol{b}}} \cup C_{\text{selected}}$
end while
return $C_{\mathcal{W}_{\boldsymbol{b}}}$

This Subroutine is used by Algorithm 2 and returns control strings $C_{\mathcal{W}_{\boldsymbol{b}}} \subset \{0,1,\mathrm{I}\}^n$ for an efficient $\mathcal{W}_{\boldsymbol{b}}$ circuit, as illustrated in Fig. 8. For visualization, we sort all control strings $\{0,1,\mathrm{I}\}^n$ into a tree, where the layer index l corresponds to how many control qubits (how many symbols 0 and 1) are used. The root of the tree in layer 0 contains the string $\mathrm{I}^{\otimes n}$. For each control string in a node, we create a new branch for every I symbol, which leads to a node containing the bi-partition $(\boldsymbol{c}_0, \boldsymbol{c}_1)$, where the selected I is replaced by either 0 or 1. The best candidates for controls strings are found by calling Subroutine 2, from which Subroutine 1 selects a maximum independent set $C_{\text{selected}} = \{\boldsymbol{c} \,|\, \boldsymbol{b}_{\boldsymbol{c}} \,\&\, \boldsymbol{b}_{\boldsymbol{c}'} = \boldsymbol{0}\}$ with a greedy algorithm. Each selected $\boldsymbol{c}$ is added to the solution $\mathcal{W}_{\boldsymbol{b}}$ and the binary vector is transformed to $\boldsymbol{b} \to \boldsymbol{b} \oplus \boldsymbol{b}_{\boldsymbol{c}}$. This is repeated until the binary vector is transformed to $\boldsymbol{0}$.

Subroutine 2 finds the lowest tree layer with valid nodes and returns that layer's most promising control strings $C_{\text{candidates}}$. Valid nodes can appear in three types: full nodes, semi-full nodes and complementary cuts. Full nodes are prioritized over semi-full nodes, which are preferred over complementary cuts.

Subroutine 2 findNextControlStrings

Input: binary vector $\boldsymbol{b} \in \mathbb{F}_2^N, N = 2^n$
Output: set $C_{\text{candidates}} \subset \{0,1,\mathrm{I}\}^n$ of control strings that can be applied next in Subroutine 1
currentLayer = {root} = $\{\mathrm{I}^{\otimes n}\}$
while true **do**
  currentLayer = doCutsToGetNextLayer(currentLayer)
  if hasFullNodes(currentLayer) **then**
    return $\{\boldsymbol{c} \in \text{currentLayer} \,|\, F_{\boldsymbol{b},\boldsymbol{c}} = 1\}$
  else if hasSemiFullNodes(currentLayer) **then**
    return $\{\boldsymbol{c} \in \text{currentLayer} \,|\, F_{\boldsymbol{b},\boldsymbol{c}} \geq 3/4\}$
  else if hasComplNodes(currentLayer) **then**
    return $\{\boldsymbol{c}_0 \,|\, (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \text{complNodes(currentLayer)}\}$
  end if
end while

Valid nodes are defined by their fullness

$$F_{\boldsymbol{b},\boldsymbol{c}} = \frac{\#_1(\boldsymbol{b}_{\boldsymbol{c}} \,\&\, \boldsymbol{b})}{\#_1(\boldsymbol{b}_{\boldsymbol{c}})} \in [0,1], \tag{35}$$

with the bitwise AND operator &. It is desirable to select a $\boldsymbol{c}$ with many I's (low layer in tree) and high $F_{\boldsymbol{b},\boldsymbol{c}}$, because this removes many 1's from $\boldsymbol{b}$ with a single MCX. We define a *full* node to have $F_{\boldsymbol{b},\boldsymbol{c}} = 1$ and a *semi-full* node to have $F_{\boldsymbol{b},\boldsymbol{c}} \geq 3/4$, see example (a) in Fig. 8. A *complementary cut* is a bi-partition $(\boldsymbol{c}_0, \boldsymbol{c}_1)$ in whose selected subspaces the 1-positions $\{\boldsymbol{\nu} \,|\, b_{\boldsymbol{\nu}} = 1\}$ of $\boldsymbol{b}$ are complementary. This is most-easily understood graphically, see example (b) in Fig. 8. Formally, a cut on layer l is complementary if $\boldsymbol{b}_{\boldsymbol{c}_0} \,\&\, \bar{\boldsymbol{b}}$ is identical to $(\boldsymbol{b}_{\boldsymbol{c}_1} \,\&\, \boldsymbol{b}) \ll l$ with the bit-shift operator $\ll$ and the complementary binary $\bar{\boldsymbol{b}}$ of $\boldsymbol{b}$. When no (semi-)full node is found in a layer, but it has a complementary cut $(\boldsymbol{c}_0, \boldsymbol{c}_1)$, it is still beneficial to add either $\boldsymbol{c}_0$ or $\boldsymbol{c}_1$ to the solution, because this creates additional full nodes for the next steps.

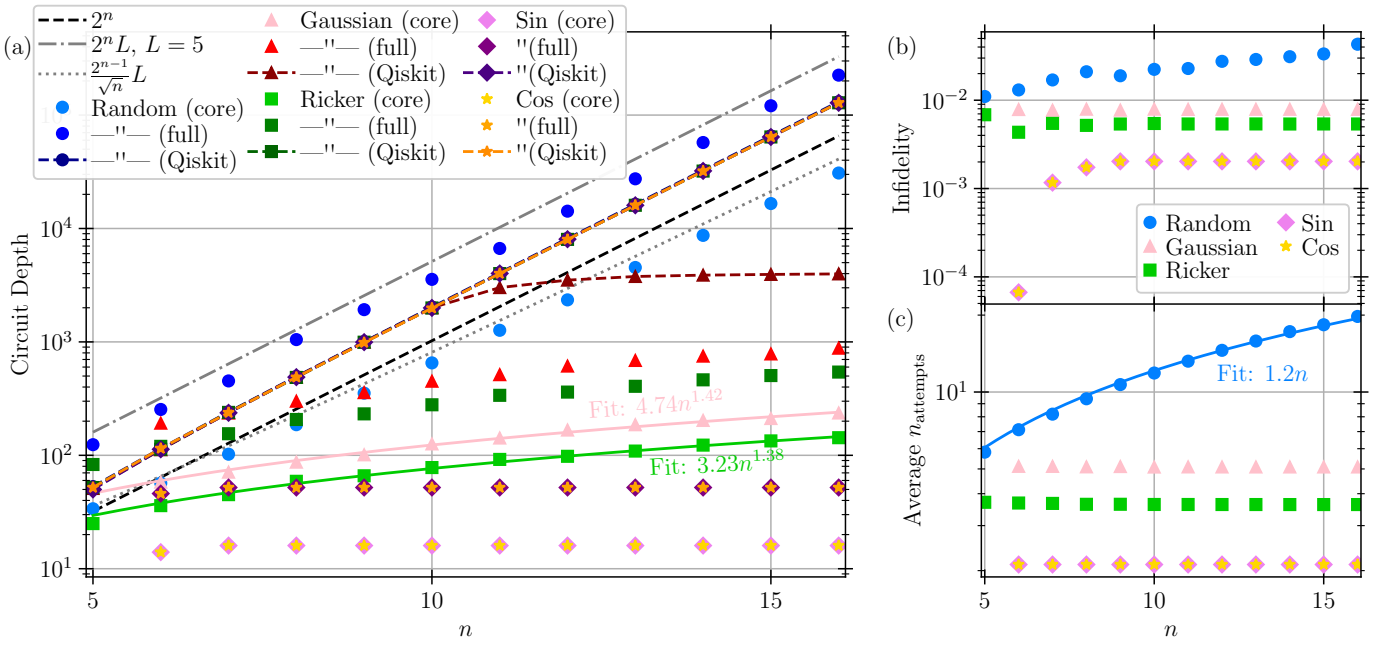

Fig. 9: Scaling of the MCX encoder algorithm circuit depth (left) with (full) and without (core) subsequent amplitude amplification, infidelity (top right) and average number of necessary attempts for a successful encoding without amplitude amplification (bottom right), for different input classes with $n \in [5, 16]$ and $L = 5$. The data spans common test functions, including: (i) a random vector with entries drawn i.i.d. from a standard normal distribution (before normalization), (ii) a Ricker wavelet and a Gaussian function, both with $\sigma = 1$, $\mu = 0$, sampled in the interval $[-3, 3]$, and (iii) the functions $\sin(x)$ and $\cos(x)$ sampled in the interval $[0, 2\pi]$. The —"— character in the legend refers to the previously mentioned input type. The depths are compared to equivalent Qiskit encoding circuits transpiled to the gate basis `Rx`, `Ry`, `Rz`, `CNOT` using the highest optimization level (level 3). All results from Qiskit overlap, showing exponential scaling greater than 2^n, except for the circuit encoding the Gaussian, whose depth saturates at around 1300 at $n = 11$. Solid lines indicate fits to the numerical data, while the dashed lines are included to aid visualization.

VI. RESULTS

To evaluate the performance and scalability of our MCX encoding algorithm, we conduct a benchmarking analysis across a range of input vector classes and sizes. In particular, we analyze how the circuit depth, infidelity, and success probability of the core circuits scale with the number of qubits. The considered inputs span both structured and unstructured data, allowing us to highlight the advantages of our approach in practical settings. Our test set includes:

- a vector randomly sampled with i.i.d. standard normal entries and then normalized to one, which is equivalent to uniformly sampling from the hypersphere $\mathcal{S}_{N-1}$ [46].
- a Ricker wavelet and a Gaussian function, both sampled in the interval $[-3, 3]$ with $\sigma = 1$, $\mu = 0$, and
- the functions $\sin(x)$ and $\cos(x)$ sampled uniformly in the interval $[0, 2\pi]$.

For each input type we compare the circuit depth to the correspondent Qiskit circuit, after transpiling the latter into the basis set {`Rx`, `Ry`, `Rz`, `CNOT`} using the highest available optimization level (level 3), with an all-to-all connected geometry [49], [50].

Figure 9 shows the results of this analysis. We choose the precision L to be 5. This comparison sheds light on the distinct scaling regimes estimated numerically and highlights the efficiency gains achievable with the MCX encoder, particularly for data exhibiting regularity, where our approach provides an exponential improvement over Qiskit for both the full and the core versions, showing a poly-log scaling in N for both the Gaussian and Ricker wavelet data while both saturating to the same relatively small constants for $\cos(x)$ and $\sin(x)$, namely 16 (core) and 52 (full), with a success probability of roughly 50% for the former. While the full MCX circuit has depth $\approx 2^n L$ for unstructured data—worse than Qiskit in this case—the *core* MCX circuit demonstrates a better scaling of $\approx \frac{2^{n-1}L}{\sqrt{n}}$, which significantly outperforms Qiskit for any fixed L and extends the window of utility. The infidelity, which can be arbitrarily exponentially decreased by choosing greater values of L, remains constant and below 1% for all examples of structured data, while it slowly increases for unstructured data, reaching up to 4% for $n = 16$. The average number of attempts, n_{attempts}, needed to successfully prepare the state using the core MCX circuit remains constant and below 5 for all the structured inputs we considered, and scales linearly with the number of qubits for unstructured data, in accordance with

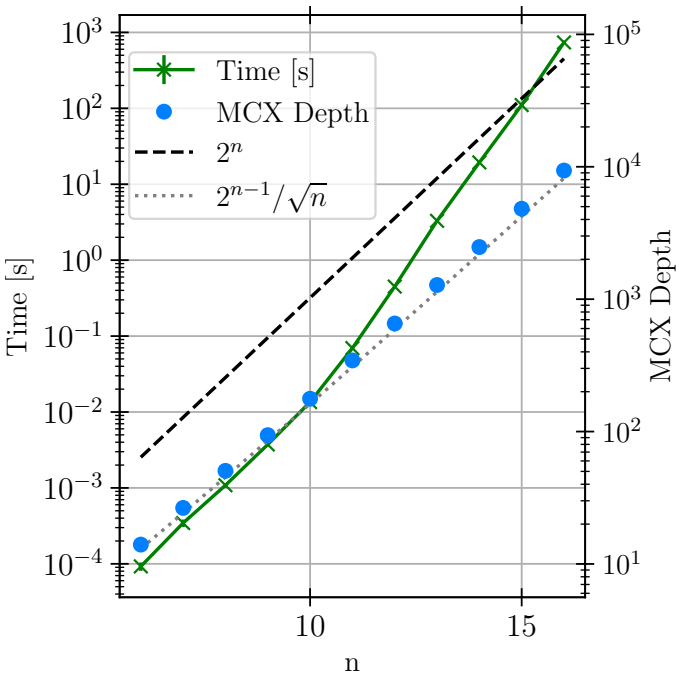

Fig. 10: Single core performance of the encoding subroutine for a single binary vector. The left y axis shows the walltime to decompose an arbitrary vector of size 2^n on a single Intel Xeon Platinum 8280 CPU core for n number of qubits on the x axis. Results are averaged over 10 random binary vectors at each n. Standard deviation of the time results are not visible at this scale. The right y axis shows the resultant depth of the MCX gate decomposition. The 2^n and $2^n/\sqrt{n}$ lines are plotted on the same scale as the MCX depth. This code can be straightforwardly parallelized across the tree search.

the asymptotic behavior of $\frac{1}{\rho}$ (see (16)) described in [46].

Figure 10 shows the efficiency of our implementation of the central computational bottleneck of Algorithm 2. This consists of Subroutines 1 and 2, which find the Kronecker decomposition of arbitrary binary vectors of size 2^n. By working directly with 64-bit unsigned integer data types we were able to optimize the implementation to where the bottleneck operations are bitwise operators. Figure 10 shows how the walltime for the decomposition of single random bitvectors on a single CPU core scales as roughly 2^n, until reaching bitvectors of size 1024 or 16 UInt64 entries, where the scaling rules change, likely due to the efficiency of CPU operations over the UInt64 vectors. This code can be straightforwardly parallelized to further improve the performance.

VII. Conclusion & Outlook

In this work, we introduce an algorithm to embed an L-bit binary approximation of N classical real values into a quantum register of $n+2$ qubits, where $n = \log_2(N)$, using amplitude encoding via a sequence of MCX operations interleaved with L many R_y rotations.

The algorithm begins by constructing a binary matrix $\mathbf{B} \in \mathbb{F}_2^{N\times L}$ that stores the binary expansions of the input values. Each column of $\mathbf{B}$ corresponds to a power of two in the expansion and to a particular intermediate quantum state, with the final column representing the target state.

The columns of $\mathbf{B}$ (along with an additional all-zero column) are then associated with nodes in a fully connected undirected weighted graph, where each Hamiltonian cycle corresponds to a distinct encoding sequence. The weight of each edge reflects the number of MCX gates required to transition between configurations, making the shortest-depth circuit equivalent to the optimal TSP tour. Since L determines the binary precision, its value does not render the TSP intractable for most practical cases [51].

To efficiently implement each transition, we apply a tree-based algorithm that finds a minimal Kronecker decomposition of a binary vector, optimizing the MCX decomposition.

Additionally, we establish an isomorphism between this graph and a hypercube, allowing for intuitive visualization and aiding the discovery of efficient encoding paths. The resulting core encoding circuit succeeds with probability ρ, the data density parameter defined in (16), whose average over uniformly random inputs decreases slowly as $\log(N)^{-1}$ [46].

This probability can be boosted using amplitude amplification with a trivial oracle, at the cost of increasing circuit depth by a factor of $\sim 1/\sqrt{\rho}$ (full circuit).

We analyzed the performance of our algorithm on several examples of structured input data from the literature, and compared the circuit depth with that of equivalent Qiskit encoding circuits transpiled to the gate set {`Rx`, `Ry`, `Rz`, `CNOT`} at the highest available optimization level (level 3). For each encoding circuit, we analyzed both the probabilistic (core) and pseudo-deterministic (full) variants. For both versions, we present numerical evidence for a scaling of the circuit depth that is $\mathcal{O}(n^\alpha L)$, $\alpha \approx 1.4$ for common structured inputs such as Gaussian or Ricker wavelet distributions, and $\mathcal{O}(L)$ for functions like $\cos(x)$ and $\sin(x)$, providing an exponential improvement over Qiskit. The success probability is constant in N and exceeds 20% in all cases.

For unstructured data, the full MCX circuit has exponential depth $\approx 2^n L$, worse than Qiskit. In contrast, the *core* MCX circuit scales more favorably as $\approx \frac{2^{n-1}L}{\sqrt{n}}$, outperforming Qiskit for fixed L, with success probability $\sim \frac{1}{n}$. Though not fully deterministic, the core version suits NISQ devices, where shorter circuits reduce noise at the cost of occasional retries. For structured data, the success probability tends to remain constant.

Future improvements are possible beyond the current low-level, binary-optimized implementation. For example, Subroutines 1 and 2 could be enhanced to better discriminate between semi-full nodes by leveraging precise fullness values when selecting nodes from maximum independent sets. Additionally, on a more fundamental level, one can choose a different basis than the binary one given by $\mathbf{B}$, such as the Hadamard–Walsh basis, which we explore in ongoing work.

REFERENCES

[1] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, Oct. 2009.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 1st ed. Cambridge University Press, 2012.

[3] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," *IBM Research Report*, 2002.

[4] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*, ser. Quantum Science and Technology. Cham: Springer International Publishing, 2018.

[5] I. F. Araujo, C. Blank, I. C. S. Araújo, and A. J. Da Silva, "Low-Rank Quantum State Preparation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 1, pp. 161–170, Jan. 2024.

[6] B. Hu, X. Huang, R. Zhou, Y. Wei, Q. Wan, and C. Pang, "A theoretical framework for quantum image representation and data loading scheme," *Science China Information Sciences*, vol. 57, no. 3, pp. 1–11, 2014. [Online]. Available: https://doi.org/10.1007/s11432-013-4866-x

[7] K. Phalak, A. Ghosh, and S. Ghosh, "Optimizing Quantum Embedding using Genetic Algorithm for QML Applications," 2024.

[8] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, "Quantum embeddings for machine learning," 2020.

[9] A. S. Holevo, "Bounds for the Quantity of Information Transmitted by a Quantum Communication Channel," *Probl. Peredachi Inf.*, vol. 9, no. 3, pp. 3–11, 1973.

[10] Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean, and T. E. O'Brien, "Quantum error mitigation," *Rev. Mod. Phys.*, vol. 95, p. 045005, Dec 2023. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.95.045005

[11] K. Lively, T. Bode, J. Szangolies, J.-X. Zhu, and B. Fauseweh, "Noise robust detection of quantum phase transitions," *Phys. Rev. Res.*, vol. 6, p. 043254, Dec 2024. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.6.043254

[12] S. Mangini, M. Cattaneo, D. Cavalcanti, S. Filippov, M. A. C. Rossi, and G. García-Pérez, "Tensor network noise characterization for near-term quantum computers," *Phys. Rev. Res.*, vol. 6, p. 033217, Aug 2024. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.6.033217

[13] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum Random Access Memory," *Physical Review Letters*, vol. 100, no. 16, p. 160501, Apr. 2008.

[14] O. D. Matteo, V. Gheorghiu, and M. Mosca, "Fault-Tolerant Resource Estimation of Quantum Random-Access Memories," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–13, 2020.

[15] K. Phalak, A. Chatterjee, and S. Ghosh, "Quantum Random Access Memory For Dummies," 2023.

[16] S. Jaques and A. G. Rattew, "QRAM: A Survey and Critique," 2023.

[17] Y. Wang, Y. Alexeev, L. Jiang, F. T. Chong, and J. Liu, "Fundamental causal bounds of quantum random access memories," *npj Quantum Information*, vol. 10, no. 1, p. 71, Jul. 2024.

[18] G. Camacho and B. Fauseweh, "Prolonging a discrete time crystal by quantum-classical feedback," *Phys. Rev. Res.*, vol. 6, p. 033092, Jul 2024. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.6.033092

[19] Y. Peng and F. Gaitan, "High-fidelity quantum state preparation using neighboring optimal control," *Quantum Information Processing*, vol. 16, no. 10, p. 261, Oct. 2017.

[20] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. Da Silva, "A divide-and-conquer algorithm for quantum state preparation," *Scientific Reports*, vol. 11, no. 1, p. 6329, Mar. 2021.

[21] E. M. Stoudenmire and D. J. Schwab, "Supervised learning with quantum-inspired tensor networks," 2017. [Online]. Available: https://arxiv.org/abs/1605.05775

[22] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Physical Review A*, vol. 83, no. 3, p. 032302, Mar. 2011.

[23] V. Shende, S. Bullock, and I. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, Jun. 2006.

[24] V. Bergholm, J. J. Vartiainen, M. Möttönen, and M. M. Salomaa, "Quantum circuits with uniformly controlled one-qubit gates," *Physical Review A*, vol. 71, no. 5, p. 052330, May 2005.

[25] E. Malvetti, R. Iten, and R. Colbeck, "Quantum Circuits for Sparse Isometries," *Quantum*, vol. 5, p. 412, Mar. 2021.

[26] M. Rosenkranz, E. Brunner, G. Marin-Sanchez, N. Fitzpatrick, S. Dilkes, Y. Tang, Y. Kikuchi, and M. Benedetti, "Quantum state preparation for multivariate functions," 2024. [Online]. Available: https://arxiv.org/abs/2405.21058

[27] A. Wright, M. Lewis, P. Zuliani, and S. Soudjani, "T-Count Optimizing Genetic Algorithm for Quantum State Preparation," in *2024 IEEE International Conference on Quantum Software (QSW)*. Shenzhen, China: IEEE, Jul. 2024, pp. 58–68.

[28] F. Mozafari, G. De Micheli, and Y. Yang, "Efficient deterministic preparation of quantum states using decision diagrams," *Physical Review A*, vol. 106, no. 2, p. 022617, Aug. 2022.

[29] T. M. L. De Veras, L. D. Da Silva, and A. J. Da Silva, "Double sparse quantum state preparation," *Quantum Information Processing*, vol. 21, no. 6, p. 204, Jun. 2022.

[30] N. Gleinig and T. Hoefler, "An Efficient Algorithm for Sparse Quantum State Preparation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 433–438.

[31] F. Mozafari, H. Riener, M. Soeken, and G. De Micheli, "Efficient Boolean Methods for Preparing Uniform Quantum States," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–12, 2021.

[32] J. Gonzalez-Conde, T. W. Watts, P. Rodriguez-Grasa, and M. Sanz, "Efficient quantum amplitude encoding of polynomial functions," *Quantum*, vol. 8, p. 1297, Mar. 2024. [Online]. Available: https://doi.org/10.22331/q-2024-03-21-1297

[33] Y. Sato, H. Tezuka, R. Kondo, and N. Yamamoto, "Quantum algorithm for partial differential equations of nonconservative systems with spatially varying parameters," *Phys. Rev. Appl.*, vol. 23, p. 014063, Jan 2025. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.23.014063

[34] I. F. Araujo, D. K. Park, T. B. Ludermir, W. R. Oliveira, F. Petruccione, and A. J. Da Silva, "Configurable sublinear circuits for quantum state preparation," *Quantum Information Processing*, vol. 22, no. 2, p. 123, Feb. 2023.

[35] X.-M. Zhang, M.-H. Yung, and X. Yuan, "Low-depth quantum state preparation," *Physical Review Research*, vol. 3, no. 4, p. 043200, Dec. 2021.

[36] D. K. Park, F. Petruccione, and J.-K. K. Rhee, "Circuit-Based Quantum Random Access Memory for Classical Data," *Scientific Reports*, vol. 9, no. 1, p. 3949, Mar. 2019.

[37] M. Ben-Dov, D. Shnaiderov, A. Makmal, and E. G. Dalla Torre, "Approximate encoding of quantum states using shallow circuits," *npj Quantum Information*, vol. 10, no. 1, p. 65, Jul. 2024.

[38] G. Marin-Sanchez, J. Gonzalez-Conde, and M. Sanz, "Quantum algorithms for approximate function loading," *Physical Review Research*, vol. 5, no. 3, p. 033114, Aug. 2023.

[39] K. Nakaji, S. Uno, Y. Suzuki, R. Raymond, T. Onodera, T. Tanaka, H. Tezuka, N. Mitsuda, and N. Yamamoto, "Approximate amplitude encoding in shallow parameterized quantum circuits and its application to financial market indicators," *Physical Review Research*, vol. 4, no. 2, p. 023136, May 2022.

[40] C. Zoufal, A. Lucchi, and S. Woerner, "Quantum Generative Adversarial Networks for learning and loading random distributions," *npj Quantum Information*, vol. 5, no. 1, p. 103, Nov. 2019.

[41] F. Gaitan, "Circuit implementation of oracles used in a quantum algorithm for solving nonlinear partial differential equations," *Physical Review A*, vol. 109, no. 3, p. 032604, Mar. 2024.

[42] C.-P. Yang, Q.-P. Su, Y. Zhang, and F. Nori, "Implementing a multi-target-qubit controlled-not gate with logical qubits outside a decoherence-free subspace and its application in creating quantum entangled states," *Phys. Rev. A*, vol. 101, p. 032329, Mar 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.101.032329

[43] H.-D. Yin, X.-X. Li, G.-C. Wang, and X.-Q. Shao, "One-step implementation of toffoli gate for neutral atoms based on unconventional rydberg pumping," *Optics Express*, vol. 28, no. 24, p. 35576, Nov. 2020. [Online]. Available: http://dx.doi.org/10.1364/OE.410158

[44] N. Goel and J. K. Freericks, "Native multiqubit toffoli gates on ion trap quantum computers," 2021. [Online]. Available: https://arxiv.org/abs/2103.00593

[45] W. H. Mills, "Some complete cycles on the n-cube," *Proceedings of the American Mathematical Society*, vol. 14, no. 4, pp. 640–643, 1963, accessed 2 Oct. 2024. [Online]. Available: https://doi.org/10.2307/2034292
[46] V. Pagni, S. Huber, M. Epping, and M. Felderer, "Fast quantum amplitude encoding of typical classical data," 2025. [Online]. Available: https://arxiv.org/abs/2503.17113
[47] S. Dalleiger and J. Vreeken, "Efficiently factorizing boolean matrices using proximal gradient descent," 2023. [Online]. Available: https://arxiv.org/abs/2307.07615
[48] C. Wan, W. Chang, T. Zhao, M. Li, S. Cao, and C. Zhang, "Fast and efficient boolean matrix factorization by geometric segmentation," 2020. [Online]. Available: https://arxiv.org/abs/1909.03991
[49] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, "Quantum circuits for isometries," *Phys. Rev. A*, vol. 93, p. 032318, Mar 2016. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.93.032318
[50] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.
[51] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.