

# Quantum encoder for fixed-Hamming-weight subspaces

Renato M.S. Farias<sup>1,2,\*</sup>, Thiago O. Maciel<sup>1</sup>, Giancarlo Camilo<sup>1</sup>, Ruge Lin<sup>1,3</sup>,  
Sergi Ramos-Calderer<sup>1,3</sup> and Leandro Aolita<sup>1</sup>

<sup>1</sup>Quantum Research Center, *Technology Innovation Institute*, Abu Dhabi, United Arab Emirates

<sup>2</sup>Instituto de Física, *Universidade Federal do Rio de Janeiro*, P.O. Box 68528, Rio de Janeiro,  
Rio de Janeiro 21941-972, Brazil

<sup>3</sup>Departament de Física Quàntica i Astrofísica and Institut de Ciències del Cosmos (ICCUB),  
*Universitat de Barcelona*, Barcelona, Spain



(Received 6 September 2024; revised 5 March 2025; accepted 6 March 2025; published 4 April 2025)

We present an exact  $n$ -qubit computational-basis amplitude encoder of real- or complex-valued data vectors of  $d = \binom{n}{k}$  components into a subspace of fixed Hamming weight  $k$ . This represents a polynomial space compression of degree  $k$ . The circuit is optimal in that it expresses an arbitrary data vector using only  $d - 1$  (controlled) reconfigurable beam splitter (RBS) gates. It is constructed by an efficient classical algorithm that sequentially generates all bitstrings of weight  $k$  and identifies the gates that superpose the corresponding states with the correct amplitudes. An explicit compilation into CNOTs and single-qubit gates is presented, with the total CNOT-gate count of  $\mathcal{O}(kd)$  provided in analytical form. In addition, we show how to load data in the binary basis by sequentially stacking encoders of different Hamming weights using  $\mathcal{O}(d \log(d))$  CNOT gates. Moreover, using generalized RBS gates that mix states of different Hamming weights, we extend the construction to efficiently encode arbitrary sparse vectors. We perform a proof-of-principle demonstration of our scheme on a commercial trapped-ion quantum computer. We successfully upload a  $q$ -Gaussian probability distribution in the non-log-concave regime with  $n = 6$  and  $k = 2$ . We also illustrate how the effect of hardware noise can be alleviated by quantum error mitigation. Numerically, we show how our encoder can improve the performance of variational quantum algorithms for problems that include particle-preserving symmetries. Our results constitute a versatile framework for quantum data compression with various potential applications in fields such as quantum chemistry, quantum machine learning, and constrained combinatorial optimizations.

DOI: [10.1103/PhysRevApplied.23.044014](https://doi.org/10.1103/PhysRevApplied.23.044014)

## I. INTRODUCTION

Amplitude encoding schemes in the basis of Hamming-weight- $k$  (HW- $k$ ) states of  $n$  qubits correspond to an interesting regime of polynomial space compression, with  $n \in \mathcal{O}(kd^{1/k})$  for data vectors of size  $d$  [1–4]. They represent a middle ground between two distinct encoding scenarios. On one side, there is an amplitude encoding scheme of zero compression that efficiently loads data in the unary basis [5–9]. On the other, there are techniques to load amplitudes in the binary basis, which provides exponential compression [10–18] but requires a number of two-qubit gates that grows exponentially with  $n$  [19–22]. Moreover, HW- $k$  encoders have the additional benefit of being the natural subspace for applications in fields such as quantum chemistry, due to the particle-preserving nature of typical Hamiltonians [1,2,23–25]. They have also been used in the context of quantum machine learning [3,4,6–8,26] and quantum finance [5,9,27,28]. When used as variational

circuits, they take advantage of the reduced subspace to mitigate the barren plateau problem [26,28,29].

Current proposals for HW- $k$  encoders present some drawbacks in practice. For instance, Refs. [3,8] use the aforementioned unary-basis encoders as subroutines. While these approaches allow for the generalization to HW  $k > 1$  using only two-qubit Givens rotations, they require orthogonalization of the input data [7] or solving nonlinear systems of  $\binom{n}{k} - 1$  equations to compute the rotation angles [3]. In this work, we address these classical overheads while still performing an exact encoding.

We present an exact, auxiliary-qubit-free amplitude encoder for real and complex data in HW- $k$  subspaces. We describe a classical algorithm that, given an initial bit-string of Hamming weight  $k$  and a data vector of size  $d$ , outputs instructions for a sequence of (controlled) Givens rotations and their respective angles in the hyperspherical coordinates of the data vector, making it efficient to compute. An explicit compilation into CNOTs and single-qubit gates is presented, with the total CNOT-gate count of  $\mathcal{O}(kd)$  provided in analytical form. Moreover, our HW- $k$  encoder

\*Contact author: [renato.msf@gmail.com](mailto:renato.msf@gmail.com)

can be used as a subroutine to power more complex algorithms. For instance, by combining all Hamming weights  $k \leq n$ , we obtain a binary encoder using  $\mathcal{O}(d \log(d))$  CNOT gates. Using a generalized reconfigurable beam splitter (RBS) gate designed to superpose states with different Hamming weights, we extend our methods and derive a procedure to load data in the paradigm of the classical sparse-access model. We deployed a proof-of-concept instance on the Aria-1 ion-trap quantum processor from IonQ [30], uploading a  $q$ -Gaussian probability distribution. The results of the demonstration were further enhanced using a well-known error-mitigation technique called Clifford data regression [31,32]. Numerically, we performed two demonstrations: first, we analyzed the effects of circuit noise in the fidelity of the encoder under a simple noise model; then, we showed how the state encoder can also be used as a variational quantum ansatz, making it a promising candidate for problems with particle-preserving symmetry.

The remainder of this paper is structured as follows. In Sec. II, we introduce the gates used in our algorithm. Then, we present our framework for HW- $k$  encoders in Sec. III. Sections IV and V expand on the sparse and binary encoders, respectively, using the fixed-HW- $k$  encoder as a subroutine. In Sec. VI A, we show the deployment of our HW- $k$  encoder on quantum hardware, and we conclude in Sec. VII.

## II. PRELIMINARIES

### A. Notation

Let  $b \in \{0, 1\}^{\otimes n}$  be a bitstring of length  $n$  and  $|b\rangle$  its Hamming weight (HW); given two bitstrings  $b$  and  $b'$ , their Hamming distance is  $|b \oplus b'|$ , where  $\oplus$  is addition mod 2. For convenience, we refer to the HW of a computational basis state  $|b\rangle$  as being the HW of the bitstring  $b$ , and both are used interchangeably. The value of the  $j$ th bit of a bitstring  $b$  is denoted by  $b(j)$ . Given a set  $\mathcal{I}$ , we denote its complement by  $\mathcal{I}_c$ . We also use the shorthand notation  $[d] := \{1, 2, \dots, d\}$ .

Here, we introduce gates that superpose two computational basis states with tunable amplitudes. These will be the building blocks for our encoders. We define the single-qubit Pauli-Y rotation as  $R_y(\theta) := e^{-i\theta Y}$ . This gate superposes the two computational basis states  $|0\rangle$  and  $|1\rangle$  of a single-qubit subspace by acting as

$$\begin{aligned} R_y(\theta) |0\rangle &= \cos(\theta) |0\rangle + \sin(\theta) |1\rangle, \\ R_y(\theta) |1\rangle &= \cos(\theta) |1\rangle - \sin(\theta) |0\rangle. \end{aligned} \quad (1)$$

When acting on a multiqubit state, (controlled)  $R_y(\theta)$  rotations can be used to superpose any two computational basis states of Hamming distance 1.

Another gate of interest is one creating a superposition between two computational basis states of Hamming

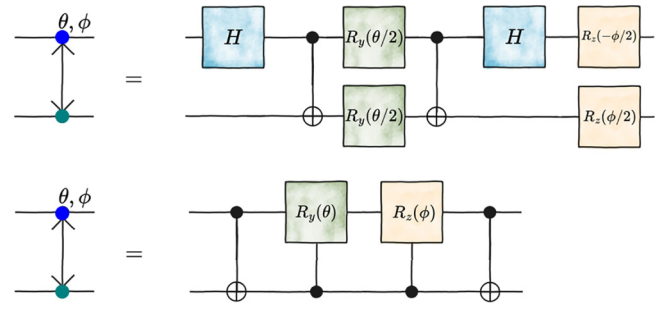


FIG. 1. Two possible compilations of the complex RBS gate  $R_{\text{out}}^{\text{in}}(\theta, \phi)$ . Colors indicate the input (blue) and output (green) qubits. Top: compilation using two CNOTs and single-qubit gates. Bottom: compilation using two CNOTs and two controlled Pauli rotations; these rotations can be merged into a single controlled SU(2) gate (see Appendix A 1).

distance 2 by acting only on a two-qubit subspace. Denoting by **in** (**out**) the bit that has value 1 (0) in the first state and 0 (1) in the second, we define the two-qubit gate  $R_{\text{out}}^{\text{in}}(\theta, \phi)$ , which acts on qubits **in** and **out**, as

$$\begin{aligned} R_{\text{out}}^{\text{in}}(\theta, \phi) |10\rangle &= e^{i\phi} \cos(\theta) |10\rangle + e^{-i\phi} \sin(\theta) |01\rangle, \\ R_{\text{out}}^{\text{in}}(\theta, \phi) |01\rangle &= e^{-i\phi} \cos(\theta) |01\rangle - e^{i\phi} \sin(\theta) |10\rangle. \end{aligned} \quad (2)$$

and as the identity elsewhere. We call  $R_{\text{out}}^{\text{in}}(\theta, \phi)$  a “complex RBS gate,” reducing to the usual RBS gate for  $\phi = 0$  [33]. We also denote  $R_{\text{out}}^{\text{in}}(\theta, 0)$  as  $R_{\text{out}}^{\text{in}}(\theta)$ . In Fig. 1, we present two possible compilations of the complex RBS using CNOTs,  $R_y(\theta)$ , and  $R_z(\phi) := e^{-i\phi Z}$  gates. In Appendix A 1, we show how to reduce the number of controlled operations in the second compilation by merging the two controlled Pauli rotations into a single controlled SU(2) rotation. Definition 1 generalizes Eq. (2) for two arbitrary computational basis states of possibly different HWs.

**Definition 1 (Generalized complex RBS gate; gRBS).** Let  $m, m' \geq 1$  be integers, **in**  $:= \{\text{in}_1, \dots, \text{in}_m\} \subset [m + m']$ , and **out**  $:= \{\text{out}_1, \dots, \text{out}_{m'}\} := [m + m'] \setminus \text{in}$ . Let  $|1_{\text{in}} 0_{\text{out}}\rangle$  ( $|0_{\text{in}} 1_{\text{out}}\rangle$ ) stand for the computational basis state having  $m$  ones (zeros) at bit positions **in** and  $m'$  zeros (ones) at bit positions **out**. We define the generalized complex RBS gate  $R_{\text{out}}^{\text{in}}(\theta, \phi)$  relative to the input and output sets **in** and **out** as the  $(m + m')$ -qubit gate acting as

$$\begin{aligned} R_{\text{out}}^{\text{in}}(\theta, \phi) |1_{\text{in}} 0_{\text{out}}\rangle &= e^{i\phi} \cos(\theta) |1_{\text{in}} 0_{\text{out}}\rangle + e^{-i\phi} \sin(\theta) |0_{\text{in}} 1_{\text{out}}\rangle, \\ R_{\text{out}}^{\text{in}}(\theta, \phi) |0_{\text{in}} 1_{\text{out}}\rangle &= e^{-i\phi} \cos(\theta) |0_{\text{in}} 1_{\text{out}}\rangle - e^{i\phi} \sin(\theta) |1_{\text{in}} 0_{\text{out}}\rangle \end{aligned} \quad (3)$$

on the two-dimensional subspace spanned by  $\{|1_{\text{in}} 0_{\text{out}}\rangle, |0_{\text{in}} 1_{\text{out}}\rangle\}$ , and as the identity elsewhere.

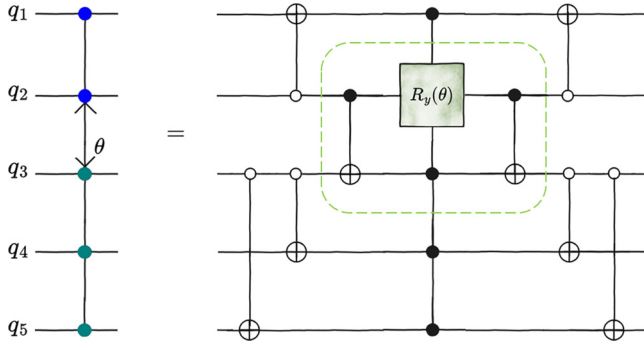


FIG. 2. Possible compilation of the gRBS gate. An illustration of the  $(m + m')$ -qubit gRBS gate for  $m = 2$  and  $m' = 3$ , namely  $R_{3,4,5}^{1,2}(\theta, \phi = 0)$ . This gate maps  $|11000\rangle$  onto a superposition of  $|11000\rangle$  and  $|00111\rangle$  with real amplitudes, according to Eq. (3). Colors indicate input (blue) and output (green) qubits. We chose the compilation in Fig. 1 (bottom), shown by the green dashed box. Controlled-gRBS gates can be built by adding extra controls to the controlled  $R_y$  rotation. This compilation can be generalized to  $R_{3,4,5}^{1,2}(\theta, \phi)$  by replacing the  $R_y$  gate with a generic  $SU(2)$  gate (see Appendix A 1).

Whenever needed, we will write down the gate addresses explicitly, i.e.,  $R_{\text{out}}^{\text{in}}(\theta, \phi) \equiv R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}(\theta, \phi)$ . For  $m' = m$ , the gRBS gate in Eq. (3) is HW-preserving, reducing to Eq. (2) if  $m' = m = 1$ . The latter will be the basic building block of our fixed-Hamming-weight encoder for dense data in Sec. III. Additionally, gRBS gates with generic  $m, m'$  superpose computational basis states of possibly different HWs and will be used to handle sparse data in Sec. IV and to build a binary basis encoder in Sec. V.

In Fig. 2, we present a possible compilation of the gRBS in Eq. (3) when  $\phi = 0$  (see Appendix A 1 for the generalization to  $\phi \neq 0$ ). The compilation proceeds as follows: (i) choose any two qubits, one from the **in** set and one from the **out** set, on which to apply an RBS gate—in this example,  $R_3^2(\theta)$  (green dashed box); (ii) add a network of  $m + m' - 2$  anti-CNOT gates with the anticontrols on each qubit selected in step (i) and the targets on all the other qubits from the respective set (**in** or **out**); (iii) add the RBS gate chosen in step (i), controlled by all the remaining qubits from **in** and **out**—in this example,  $c_{1,4,5}R_3^2(\theta)$ ; (iv) undo the anti-CNOT network of step (ii).

As explained in the following sections, our encoders will repeatedly add a new component  $|b'\rangle$  to a superposition state  $|\psi\rangle$  by acting nontrivially only on the two-dimensional subspace spanned by  $|b'\rangle$  and one of the existing components  $|b\rangle$  of  $|\psi\rangle$ . This requires, in general, controlled gRBS gates, as detailed in Lemma 1.

**Lemma 1 (Adding a new amplitude to a state  $|\psi\rangle$ ).** Let  $n \geq 2$  be an integer,  $b, b' \in \{0, 1\}^{\otimes n}$  bitstrings of Hamming weights  $|b|$  and  $|b'|$ , respectively, and  $|\psi\rangle$  be an arbitrary  $n$ -qubit state having a nonzero amplitude along the

computational basis state  $|b\rangle$  but not along  $|b'\rangle$ . Let  $\mathcal{I}, \mathcal{I}'$  be the sets of bits where  $b$  and  $b'$  have a 1, respectively, and define the sets of input, output, control, and anticontrol qubits respectively by **in**  $:= \mathcal{I} \setminus \mathcal{I}'$ , **out**  $:= \mathcal{I}' \setminus \mathcal{I}$ , **ctrl**  $:= \mathcal{I} \cap \mathcal{I}'$ , and **ctrl**  $:= \mathcal{I}_c \cap \mathcal{I}'_c$ . Their respective cardinalities are denoted  $m, m', \ell$ , and  $\ell'$ , with  $m + m' + \ell + \ell' = n$ . Then, the  $n$ -qubit multicontrolled gate  $\bar{c}_{\text{ctrl}} c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta, \phi)$  acts on  $|\psi\rangle$  by superposing its  $|b\rangle$  component with  $|b'\rangle$  as in Eq. (3) while leaving all the other components untouched. In particular,

- (1) if  $m' \geq m$  and  $|b\rangle$  is the computational basis state of maximum HW in  $|\psi\rangle$ , the anticontrols can be removed and the  $(n - \ell')$ -qubit gate  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta, \phi)$  superposes  $|b\rangle$  with  $|b'\rangle$  of HW  $|b'| = |b| + (m' - m)$ ;
- (2) if  $m' \leq m$  and  $|b\rangle$  is the computational basis state of minimum HW in  $|\psi\rangle$ , the controls can be removed, and the  $(n - \ell)$ -qubit gate  $\bar{c}_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta, \phi)$  superposes  $|b\rangle$  with  $|b'\rangle$  of HW  $|b'| = |b| - (m - m')$ .

Our HW- $k$  encoders in Sec. III fall in the scope of the first subcase of the above lemma and, therefore, will be based on multicontrolled gRBS gates. In Appendix A 1, we show that a  $c_{\text{ctrl}_1, \dots, \text{ctrl}_\ell} R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}(\theta, \phi)$  gate can be compiled using at most  $22(m + m') + 20(\ell - 1)$  CNOT gates, with exact compilations provided for  $\ell \leq 3$  and  $m = m' = 1$ , and upper bounds otherwise (see Lemma 2 and Table I for a summary).

### III. HAMMING-WEIGHT- $k$ ENCODERS

We start this section by defining generic amplitude encoders and parameter-optimal encoders. Next, we present our Hamming-weight- $k$  encoder.

#### A. Amplitude encoders

Let us define amplitude encoders for an arbitrary set of computational basis states.

**Definition 2 (Amplitude encoder).** Let  $\mathbf{x}$  be any  $d$ -dimensional data vector,  $\|\mathbf{x}\| := \sqrt{\sum_{j=1}^d |x_j|^2}$  be its  $\ell_2$ -norm, and  $B := \{|b_j\rangle : b_j \in \{0, 1\}^{\otimes n}\}_{j \in [d]}$  be a set of  $d$  computational basis states of  $n$  qubits, with  $n \geq \log_2(d)$ . An amplitude encoder in the basis  $B$  is an  $n$ -qubit parameterized quantum circuit  $\text{Load}_B(\mathbf{x})$ , with gate parameters depending on  $\mathbf{x}$ , such that

$$\text{Load}_B(\mathbf{x}) |0\rangle^{\otimes n} := \frac{1}{\|\mathbf{x}\|} \sum_{j=1}^d x_j |b_j\rangle. \quad (4)$$

We explicitly distinguish between real-valued data vectors,  $\mathbf{x} \in \mathbb{R}^d$ , and complex-valued ones,  $\mathbf{x} \in \mathbb{C}^d$ , since the basic quantum gates will be distinct in each case. Gate parameters in Definition 2 refer to rotation angles such as  $\theta$  and  $\phi$  introduced in Sec. II. The total number of

parameterized gates in a given circuit may vary depending on the specific circuit synthesis. In particular, in the case of variational encoders, usual schemes [3,4] resort to over-parameterization to reach the necessary expressivity to encode an arbitrary vector  $\mathbf{x}$ . We will present a deterministic protocol to construct circuits using the minimum number of parameters to express any  $\mathbf{x}$  exactly. Our construction is based on the observation that, due to the  $\ell_2$ -normalization in Eq. (4), we effectively encode the vector  $\mathbf{x}/\|\mathbf{x}\|$ , which lies in the unit  $(d-1)$ -sphere for  $\mathbf{x} \in \mathbb{R}^d$  or  $(2d-1)$ -sphere for  $\mathbf{x} \in \mathbb{C}^d$ . Therefore, expressing an arbitrary data vector requires exactly  $d-1$  and  $2d-1$  real parameters, respectively. This motivates the following definition of a parameter-optimal encoder.

**Definition 3 (Parameter-optimal amplitude encoder).** An amplitude encoder  $\text{Load}_B(\mathbf{x})$  is parameter-optimal if it uses exactly  $d-1$  gate parameters for  $\mathbf{x} \in \mathbb{R}^d$  or  $2d-1$  gate parameters for  $\mathbf{x} \in \mathbb{C}^d$ .

Efficient quantum data encoders using  $\mathcal{O}(d)$  two-qubit gates are known in the unary basis  $B_1 = \{|b_j\rangle : b_j \in \{0,1\}^{\otimes n} \text{ and } |b_j| = 1\}_{j \in [d]}$  [5–9]; however, they require as many qubits as data entries ( $n = d$ ) and hence do not offer any space compression. The binary basis  $B_{\text{binary}} = \{|b_j\rangle : b_j \in \{0,1\}^{\otimes n}\}$ , with  $n = \lceil \log_2(d) \rceil$ , gives rise to exponential space compression, but the corresponding encoders have exponential gate complexity [22]. In the next section, we study amplitude encoding on a Hamming-weight- $k$  subspace, i.e.,

$$B_k := \{|b_j\rangle : b_j \in \{0,1\}^{\otimes n} \text{ and } |b_j| = k\}_{j \in [d]}, \quad (5)$$

with  $k \in [n]$  and  $d = \binom{n}{k}$ . The amplitude encoder  $\text{Load}_{B_k}(\mathbf{x})$  operates in an intermediate regime of polynomial space compression,  $n \in \mathcal{O}(kd^{1/k})$ . In Sec. V, we construct a binary basis encoder combining HW- $k$  encoders for all  $k$ , since  $B_{\text{binary}} \equiv \bigcup_{k=0}^n B_k$ .

#### SUBROUTINE 1. Finding the next HW- $k$ bitstring

---

```

1 Input: bitstring  $b$  and indices marked of marked bits of  $b$ 
2 Output: new bitstring  $b'$  satisfying  $|b'| = |b|$  and
    $|b \oplus b'| = 2$ , and updated list of marked bits
3
4 NEXTBS( $b$ , marked)
5  $p \leftarrow \max(\text{marked})$  ▷ index of the pivot bit
6 if  $b(p) = 0$  then
7    $\ell \leftarrow$  index of nearest 1 to the right of  $p$ 
8 else
9    $\ell \leftarrow$  index of farthest 0 to the right of  $p$ , without
     passing another 1
10  $b' \leftarrow b$  with  $p$ -th and  $\ell$ -th bits exchanged
11  $j \leftarrow$  starting index of the last sequence of equal bits in  $b'$ 
12  $\text{marked}' \leftarrow \text{marked} \setminus \{p\} \cup \{p+1, \dots, j-1\}$ 
13   ▷ unmark bit  $p$  and mark all bits between  $p$  and  $j$ 
14 return  $\{b', \text{marked}'\}$ 

```

---

## B. HW- $k$ amplitude encoder

In this section, we introduce an amplitude encoder  $\text{Load}_{B_k}(\mathbf{x})$  on a Hamming-weight- $k$  subspace  $B_k$  given by Eq. (5). Hereafter, without loss of generality, the vector  $\mathbf{x}$  is assumed to have dimension  $d = \binom{n}{k}$ . The resulting quantum circuit uses the HW-preserving gates  $R_{\text{out}}^{\text{in}}(\theta, \phi)$  introduced in Sec. II as basic ingredients. The circuit architecture is decided by a classical algorithm that generates a sequence  $[b_j]_{j \in [d]}$  of all HW- $k$  bitstrings ordered such that the Hamming distance  $|b_j \oplus b_{j+1}|$  between subsequent bitstrings is 2. To generate these sequences, we use a *Gray code* called Ehrlich's algorithm [34]. This code receives an initial bitstring  $b$  with  $|b| = k$  as input and outputs the sequence of all bitstrings of HW  $k$  by flipping only one pair of bits at each step of an iterative procedure. The total number of iterations is exactly  $\binom{n}{k} - 1$ . For completeness, the procedure for one such iterative step is presented in Algorithm 1. The fact that  $|b_j \oplus b_{j+1}| = 2$  for every  $j$  allows one to superpose the corresponding states  $|b_j\rangle$  and  $|b_{j+1}\rangle$  using (controlled) RBS gates, as explained in Sec. II [see Eq. (2)]. In Algorithm 2, we present a routine that, given  $b_j$  and  $b_{j+1}$ , computes the parameters {in, out, ctrl} of the gate needed to superpose  $|b_j\rangle$  and  $|b_{j+1}\rangle$  as follows: the 1s that remain unchanged between bitstrings  $b_j$  and  $b_{j+1}$  correspond to control qubits, while the bits that were 1 (0) in  $b_j$  and became 0 (1) in  $b_{j+1}$  are associated with the input (output) of the RBS gate. The angles  $(\theta_j, \phi_j)$  are coordinates of the vector  $\mathbf{x}/\|\mathbf{x}\|$  on the hypersphere and depend on whether  $\mathbf{x} \in \mathbb{R}^d$  or  $\mathbf{x} \in \mathbb{C}^d$ . They will be presented in Secs. III B 1 and III B 2 below.

The general procedures to construct circuits for  $\mathbf{x} \in \mathbb{R}^d$  and  $\mathbf{x} \in \mathbb{C}^d$  are given by Algorithms 1 and 2, respectively. These algorithms are classical and have four independent routines that can be summarized as follows:

#### Summary of Algorithms 1 and 2

1. Use the Ehrlich algorithm (Subroutine 1) to generate a sequence  $[b_j]_{j \in [d]}$  of HW- $k$  bitstrings with Hamming distances  $|b_j \oplus b_{j+1}| = 2$  for all  $j$ ;
2. Given  $b_j$  and  $b_{j+1}$ , use Subroutine 2 to compute gate addresses ctrl, in, out needed to superpose states  $|b_j\rangle$  and  $|b_{j+1}\rangle$ ;
3. If  $\mathbf{x} \in \mathbb{R}^d$ , compute gate parameter  $\theta_j$  using Eq. (7); if  $\mathbf{x} \in \mathbb{C}^d$ , calculate  $(\theta_j, \phi_j)$  using Eqs. (9) and (10);
4. Add gate  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta_j, \phi_j)$  (or  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta_j)$ ) to the circuit;
5. Repeat steps 2–4,  $\forall j$ .

SUBROUTINE 2. Getting gate addresses from  $b, b'$ 


---

```

1 Input: two bitstrings  $b$  and  $b'$ , and set untouched of
  indices of bits with value 1 in the initial bitstring
2 Output: Gate parameters  $\{in, out, ctrl\}$  needed to
  superpose  $|b\rangle, |b'\rangle$ , and updated list of untouched bits
3
4  $GATEADDRESSES(b, b', untouched)$ 
5  $\mathcal{I} \leftarrow$  list of indices where  $b$  has an 1
6  $\mathcal{I}' \leftarrow$  list of indices where  $b'$  has an 1
7  $ctrl \leftarrow \mathcal{I} \cap \mathcal{I}'$ 
8  $in \leftarrow \mathcal{I} \setminus ctrl$ 
9  $out \leftarrow \mathcal{I}' \setminus ctrl$ 
10  $untouched' \leftarrow untouched \setminus \{in, out\}$ 
11  $\triangleright$  remove  $in, out$  since now they have been touched
12  $ctrl \leftarrow ctrl \setminus untouched'$   $\triangleright$  remove unnecessary controls
13 return  $\{in, out, ctrl, untouched'\}$ 

```

---

We refer to the particular bitstring sequence generated in step 1 of this summary as the *Ehrlich ordering* (see Table II in Appendix B for an example with  $n = 6$  and  $k = 2$ ). It is worth noting that if one is required to encode amplitudes into specific pre-established states  $|b_j\rangle$  (e.g., preserving standard binary ordering), then one must sort the input data vector  $\mathbf{x}$  beforehand to match the Ehrlich ordering. Hereafter, we assume that either  $\mathbf{x}$  is sorted in Ehrlich ordering or that the data do not require a specific ordering. When that is not the case, more complicated gates may be needed to superpose  $|b_j\rangle$  and  $|b_{j+1}\rangle$ —this is the scope of the sparse-access model encoder discussed in Sec. IV. Additionally, for  $k = 1$ , the circuit recovers the unary encoding architecture [35].

*A priori*, the procedure above adds a controlled RBS gate with exactly  $k - 1$  controls to the circuit at each iteration. This would result in the naive count of  $\binom{n}{k} - 1$  controlled RBS gates in total with  $k - 1$  controls each; however, several controls can be removed at the initial iterations since the corresponding control qubits are in the state  $|1\rangle$  and have not yet been entangled with any other qubit. The variable “*untouched*” in Algorithm 2 keeps track of these disentangled qubits and is used to remove unnecessary controls from the list of gates returned by the naive algorithm. As a result of this optimization, the circuit resulting from Algorithm 1 consists of  $\binom{n-(k-\ell)}{\ell+1}$  controlled RBS gates having  $\ell$  controls each, with  $\ell + 1 \in [k]$  ( $\ell = 0$  corresponds to removing all the controls and  $\ell = k - 1$  to no removal). This leads to a significant reduction in the total CNOT-gate count of the circuit. The total number of parameterized gates,  $\sum_{\ell=0}^{k-1} \binom{n-(k-\ell)}{\ell+1} = \binom{n}{k} - 1$ , remains the same, while the total CNOT-gate count of the optimized HW- $k$  encoder  $\text{Load}_{B_k}(\mathbf{x})$  generated by Algorithm 1 is

$$\begin{aligned}
\# \text{CNOTS} &= \sum_{\ell=0}^{k-1} \binom{n-(k-\ell)}{\ell+1} C_\ell \\
&\leq \left[ \binom{n}{k} - 1 \right] C_{k-1},
\end{aligned} \tag{6}$$

ALGORITHM 1. HW- $k$  encoder for dense  $\mathbf{x} \in \mathbb{R}^d$ .

---

```

1 Input: number of qubits  $n$ , Hamming weight  $k$ , and
  data vector  $\mathbf{x}$  of dimension  $d \leq \binom{n}{k}$ 
2 Output: Quantum circuit  $\mathcal{C} = \text{Load}_{B_k}(\mathbf{x})$ 
3
4  $\text{HWENCODER}(n, k, \mathbf{x})$ 
5  $\mathcal{C} \leftarrow \text{Circuit}(n)$ 
6 for  $i \leftarrow n - k + 1$  to  $n$  do
7    $\lfloor$  Add gate  $X$  on qubit  $i$  to  $\mathcal{C}$ 
8    $b \leftarrow 1^k 0^{n-k}$   $\triangleright$  initial bitstring
9    $marked \leftarrow \{1, \dots, k\}$   $\triangleright$  indices of marked bits
10   $untouched \leftarrow \{n - k + 1, \dots, n\}$   $\triangleright$  indices of bits with
    value 1 that have never been touched by an RBS gate
11  for  $j \leftarrow 1$  to  $d - 1$  do
12     $\{b', marked'\} \leftarrow \text{NEXTBS}(b, marked)$ 
13     $\{in, out, ctrl, untouched'\} \leftarrow$ 
       $GATEADDRESSES(b, b', untouched)$ 
14    Compute  $\theta_j$  from Eq. (7)
15    Add gate  $c_{ctrl} R_{out}^{in}(\theta_j)$  to  $\mathcal{C}$ 
16     $\{b, marked, untouched\} \leftarrow \{b', marked', untouched'\}$ 
17 return  $\mathcal{C}$ 

```

---

where  $C_\ell$  is the CNOT-gate cost of a single  $\ell$ -controlled  $R_{out}^{in}(\theta, \phi)$  gate. The upper bound is the naive count without simplifying initial controls. The latter will be the exact CNOT count whenever the simplification is not possible because the initial state already contains a superposition, such as for the binary encoder in Sec. V. The precise values of  $C_\ell$  depend on the compilation (see Appendix A 1) and also differ for  $\phi = 0$  and  $\phi \neq 0$ . We present explicit expressions for each of these cases in Lemmas 2 and 3. We observed heuristically that choosing  $1^k 0^{n-k}$  as the initial bitstring in Algorithm 1 maximizes the removal of redundant controls in the initial iterations. Furthermore, it has the extra benefit of grouping the controls of some of the subsequent controlled RBS gates. This opens the possibility of further reducing the total CNOT cost in the circuit using one clean auxiliary qubit to control these groups of gates all at once (see Appendix C). For simplicity, we only present CNOT-gate counts for the implementation without auxiliary qubits.

It is important to mention that, for HW  $k > n/2$ , one can avoid the costly implementation of  $(k - 1)$ -controlled RBS gates that would arise from the above procedure by exploring the fact that HW- $k$  bitstrings are the negation of bitstrings of HW- $(n - k)$ . Therefore, a more efficient circuit can be built by copying the architecture of an HW- $(n - k)$  encoder circuit with the negated initial bitstring and controlled RBS gates replaced by anticcontrolled RBSs; however, this procedure only works if the initial state is not in a superposition.

Finally, the way we construct the circuits naturally yields the angles of the RBS gates in the hyperspherical coordinate system. The next two subsections cover the cases when  $\mathbf{x}$  is real- and complex-valued, respectively.

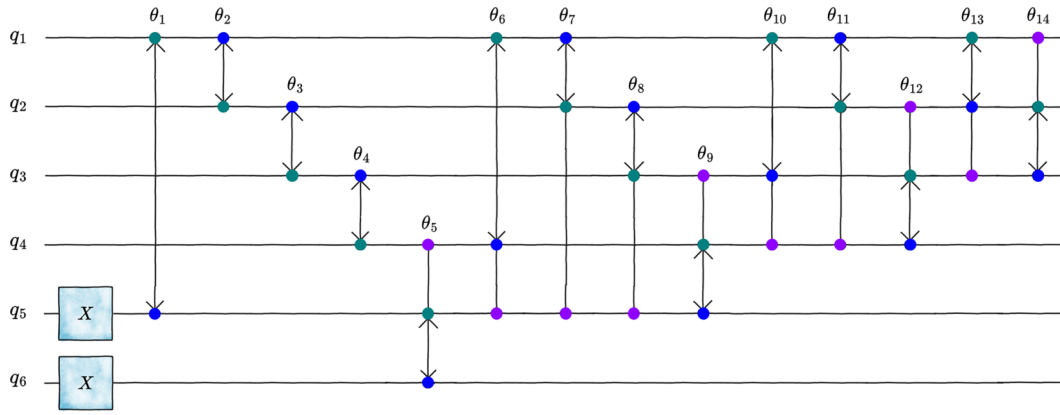


FIG. 3. Fixed-Hamming-weight amplitude encoder. Circuit generated by Algorithm 1 with  $n = 6$  and  $k = 2$  (see Table II for details). Colors are used to indicate the input (blue), output (green), and control (violet) qubits of each controlled RBS gate. Angles  $\{\theta_j\}_{j \in [14]}$  are calculated using Eq. (7).

### 1. Dense real-valued data

For  $\mathbf{x} \in \mathbb{R}^d$ , the  $d - 1$  angles of the  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta_j)$  gates used to build the circuit in Algorithm 1 are

$$\theta_j := \text{atan2} \left( \sqrt{\sum_{j'=j+1}^d x_{j'}^2}, x_j \right), \quad j \in [d-2] \quad (7)$$

$$\theta_{d-1} := \text{atan2}(x_d, x_{d-1}),$$

where the two-argument arctangent function  $-\pi < \text{atan2}(y, x) \leq \pi$  is the principal value of  $\arg(x + iy)$ . These are the hyperspherical coordinates of the normalized vector  $\mathbf{x}/\|\mathbf{x}\| \in \mathbb{S}^{d-1}$ , i.e.,

$$\begin{aligned} x_1 &= \|\mathbf{x}\| \cos(\theta_1), \\ x_2 &= \|\mathbf{x}\| \sin(\theta_1) \cos(\theta_2), \\ &\vdots \\ x_{d-1} &= \|\mathbf{x}\| \sin(\theta_1) \cdots \sin(\theta_{d-2}) \cos(\theta_{d-1}), \\ x_d &= \|\mathbf{x}\| \sin(\theta_1) \cdots \sin(\theta_{d-2}) \sin(\theta_{d-1}). \end{aligned}$$

In Fig. 3, we present an illustrative example of the resulting quantum circuit for  $k = 2$  and  $n = 6$  qubits (hence  $d = 15$ ). An explicit step-by-step sketch of the execution of Algorithm 1 for the same example is presented in Table II in Appendix B. Lemma 2 provides the total CNOT-gate count to implement  $\text{Load}_{B_k}(\mathbf{x})$  using Algorithm 1.

*Lemma 2 (Total CNOT cost of  $\text{Load}_{B_k}(\mathbf{x})$  for  $\mathbf{x} \in \mathbb{R}^d$ ).* Let  $n \geq 2$  and  $k \in [1, n/2]$  be integers,  $d = \binom{n}{k}$ , and  $\mathbf{x} \in \mathbb{R}^d$ . The HW- $k$  encoder  $\text{Load}_{B_k}(\mathbf{x})$  generated by Algorithm 1 can be implemented using a number  $\mathcal{O}(kd)$

of CNOT gates, namely

$$\begin{cases} 2(n-1), & k=1, \\ (n-2)(3n-1), & k=2, \\ \frac{1}{3}(n-3)(5n^2-6n-2), & k=3, \\ \frac{1}{12}(n-4)(13n^3-58n^2+79n-42), & k=4, \\ \frac{1}{12} \sum_{\ell=1}^4 a_\ell (n-k)^\ell + b_\ell, & k \geq 5, \end{cases} \quad (8)$$

with  $a_1 = 178$ ,  $a_2 = 239$ ,  $a_3 = 98$ ,  $a_4 = 13$ , and  $b_\ell \leq \sum_{\ell=5}^{k-1} \binom{n-(k-\ell)}{\ell+1} (16\ell - 6)$ .

*Proof.* The total CNOT-gate count is given by the general expression in Eq. (6), where  $C_\ell$  is the cost of compiling a single  $\ell$ -controlled  $R_{\text{out}}^{\text{in}}(\theta)$  gate. We calculate this cost in Lemma 2, and the results are summarized in Table I in Appendix A.1. Plugging the results in Lemma 2 in Eq. (6) immediately leads to Eq. (8). ■

### 2. Dense complex-valued data

For  $\mathbf{x} \in \mathbb{C}^d$ , we need two sets of angles,  $\{\theta_j\}_{j \in [d-1]}$  and  $\{\phi_j\}_{j \in [d]}$ , to encode the absolute values and phases of each amplitude  $\{x_j = |x_j| \exp(i \arg(x_j))\}_{j \in [d]}$ .

The  $d - 1$  angles  $\theta_j$  are responsible for the encoding of the absolute values  $|x_j|$ , and are calculated as the hyperspherical coordinates of the vector  $\{|x_1|, \dots, |x_d|\}$ , similarly to Eq. (7). Explicitly, the  $\theta_j$  angles are given by

$$\begin{aligned} \theta_j &:= \text{atan2} \left( \sqrt{\sum_{j'=j+1}^d |x_{j'}|^2}, |x_j| \right), \quad j \in [d-2]; \\ \theta_{d-1} &:= \text{atan2}(|x_d|, |x_{d-1}|). \end{aligned} \quad (9)$$

The first  $d - 1$  angles  $\phi_j$ , on the other hand, are responsible for the encoding of the complex phases of each entry  $\{x_j\}_{j \in [d-1]}$ . The first angle,  $\phi_1$ , is directly related to the complex phase of the first data entry,  $x_1$ . Inspecting Eq. (2), one can see that, while the complex RBS gate encodes a desired complex phase onto the amplitude of an “initial” computational basis state, it also encodes the complex conjugate of the same phase onto the amplitude of the “new” computational basis state created in the superposition. This leads to the accumulation of undesired phases every time a complex RBS gate is applied. To remove these undesired phases, we add correction terms to the subsequent angles,  $\phi_{j>1}$  as follows:

$$\begin{aligned}\phi_1 &:= -\arg(x_1), \\ \phi_j &:= -\arg(x_j) + \sum_{j'=1}^{j-1} \phi_{j'}, \quad j \in [2, d].\end{aligned}\quad (10)$$

To guarantee numerical stability, all the angles are taken mod  $2\pi$ . In addition to the complex RBS gate defined in Eq. (2), we also need to add an antiphase gate  $\overline{\text{Ph}}(\phi) := e^{-i\phi} R_z(\phi/2)$  at the end of the circuit to cancel out the excess complex phase created by the sequence of complex RBS gates, where  $\overline{\text{Ph}}(\phi)$  is controlled by all the qubits corresponding to a 1 value in  $b_{d-1}$  and acts on any qubit corresponding to a 0 value.

Thus, the resulting quantum circuit has the same architecture of controlled RBS gates as in the real case (see

Fig. 3), plus an extra (controlled)  $\overline{\text{Ph}}$  gate at the end of the circuit, as well as the choice of gate synthesis (see Fig. 1). Lemma 3 provides the total CNOT-gate count to implement  $\text{Load}_{B_k}(\mathbf{x})$  using Algorithm 2.

*Lemma 3* (Total CNOT-gate cost of  $\text{Load}_{B_k}(\mathbf{x})$  for  $\mathbf{x} \in \mathbb{C}^d$ ). Let  $n \geq 2$  and  $k \in [1, n/2]$  be integers,  $d = \binom{n}{k}$ , and  $\mathbf{x} \in \mathbb{C}^d$ . The HW- $k$  encoder  $\text{Load}_{B_k}(\mathbf{x})$  generated by Algorithm 1 can be implemented using a number  $\mathcal{O}(k d)$  of CNOT gates, namely

$$\begin{cases} 2(n-1), & k=1, \\ (n-2)(3n-1), & k=2, \\ \frac{1}{3}(n-3)(7n^2-12n+2), & k=3, \\ \frac{1}{12}(n-4)(19n^3-86n^2+105n-30), & k=4, \\ \frac{1}{12} \sum_{\ell=1}^4 \tilde{a}_\ell (n-k)^\ell + \tilde{b}_\ell, & k \geq 5, \end{cases} \quad (11)$$

with  $\tilde{a}_1 = 230, \tilde{a}_2 = 329, \tilde{a}_3 = 142, \tilde{a}_4 = 19$ , and  $\tilde{b}_\ell \leq \sum_{\ell=5}^{k-1} \binom{n-(k-\ell)}{\ell+1} (20\ell+4)$ .

*Proof.* The total CNOT-gate count is given by the general expression (6), where  $C_\ell$  here is the cost of compiling a single  $\ell$ -controlled  $R_{\text{out}}^{\text{in}}(\theta, \phi \neq 0)$  gate. This is computed in Lemma 2 in Appendix A 1. Substituting this expression immediately yields Eq. (11). ■

#### ALGORITHM 2. HW- $k$ encoder for dense $\mathbf{x} \in \mathbb{C}^d$ .

---

```

1 Input: number of qubits  $n$ , Hamming weight  $k$ , and
  data vector  $\mathbf{x}$  of dimension  $d \leq \binom{n}{k}$ 
2 Output: Quantum circuit  $\mathcal{C} = \text{Load}_{B_k}(\mathbf{x})$ 
3
4 HWENCODER ( $n, k, \mathbf{x}$ )
5  $\mathcal{C} \leftarrow \text{Circuit}(n)$ 
6 for  $i \leftarrow n - k + 1$  to  $n$  do
7    $\perp$  Add gate  $X$  on qubit  $i$  to  $\mathcal{C}$ 
8    $b \leftarrow 1^k 0^{n-k}$   $\triangleright$  initial bitstring
9    $\text{marked} \leftarrow \{1, \dots, k\}$   $\triangleright$  indices of marked bits
10   $\text{untouched} \leftarrow \{n - k + 1, \dots, n\}$   $\triangleright$  indices of ones that
    have never been flipped
11 for  $j \leftarrow 1$  to  $d - 1$  do
12    $\{b', \text{marked}'\} \leftarrow \text{NEXTBS}(b, \text{marked})$ 
13    $\{\text{in}, \text{out}, \text{ctrl}, \text{untouched}'\} \leftarrow$ 
     $\text{GATEADDRESSES}(b, b', \text{untouched})$ 
14   Calculate  $\theta_j$  and  $\phi_j$  via Eqs. (9) and (10)
15   Add gate  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta_j, \phi_j)$  to  $\mathcal{C}$ 
16    $\{b, \text{marked}, \text{untouched}\} \leftarrow \{b', \text{marked}', \text{untouched}'\}$ 
17 Calculate  $\phi_d$  via Eq. (10)
18  $\text{ctrl} \leftarrow$  set of  $k$  indices where the bits of  $b'$  are 1
19  $\text{in} \leftarrow$  any index where  $b'$  has a 0
20 Add gate  $c_{\text{ctrl}} \overline{\text{Ph}}(\phi_d)$  acting on qubit  $\text{in}$  to  $\mathcal{C}$ 
21 return  $\mathcal{C}$ 

```

---

#### IV. SPARSE ENCODER

Here, we extend the construction of Sec. III to the case of sparse data. For a  $d$ -dimensional data vector  $\mathbf{x}$  having  $s \ll d$  nonzero entries, the HW- $k$  encoder of Sec. III still uses  $d - 1$  parameterized gates; however, in this case, a parameter-optimal amplitude encoder, as expressed in Definition 3, would require only  $s - 1$  gate parameters.

In this setting, we consider a sparse-access model, where the data vector  $\mathbf{y}$  to be encoded is of the form

$$\mathbf{y} := \{(x_1, b_1), (x_2, b_2), \dots, (x_s, b_s)\}, \quad (12)$$

with  $\{x_j\}_{j \in [s]}$  being the nonzero components of  $\mathbf{x}$  and  $\{b_j\}_{j \in [s]}$  being the addresses (in bitstring format) associated with these values. The goal is to prepare the state

$$s\text{-Load}(\mathbf{y}) |0\rangle^{\otimes n} := \frac{1}{\|\mathbf{x}\|} \sum_{j \in [s]} x_j |b_j\rangle, \quad (13)$$

similarly to the fixed-HW encoders discussed in Sec. III; however, in the sparse-access model,  $b_j$  and  $b_{j+1}$  do not need to satisfy any constraints, such as having equal HWs or fixed Hamming distances. Consequently, the complex RBS gate used in the previous sections is not sufficient to

## ALGORITHM 3. Amplitude encoder for sparse data.

---

```

1 Input: number of qubits  $n$  and tuple  $y := \{(x_i, b_i)\}_{i \in [s]}$ 
  of non-zero data values  $x_i$  and their addresses  $b_i$ 
2 Output: Quantum circuit  $\mathcal{C} = s\text{-Load}(y)$ 
3
4 SPARSEHWENCODER( $n, y$ )
5  $bs \leftarrow b_1$   $\triangleright$  address  $b$  of the 1st element of the  $y$  tuple
6  $\mathcal{I} \leftarrow$  set of indices where the bits of  $bs$  have value 1
7  $\mathcal{C} \leftarrow \text{Circuit}(n)$ 
8 for  $i \leftarrow 1$  to  $|\mathcal{I}|$  do
9    $\lfloor$  Add gate  $X$  on qubit  $\mathcal{I}_i$  to  $\mathcal{C}$ 
10  $untouched \leftarrow [n] \setminus \mathcal{I}$ 
11 for  $j \leftarrow 2$  to  $s$  do
12    $bs' \leftarrow b_j$ 
13    $\{in, out, ctrl, untouched'\} \leftarrow$ 
     GATEADDRESSES( $bs, bs', untouched$ )
14   Calculate  $\theta_j$  and  $\phi_j$  via Eqs. (9) and (10)
      $\triangleright$  using  $x$  values from  $y$ 
15   Add gate  $c_{ctrl} R_{out}^{in}(\theta_j, \phi_j)$  to  $\mathcal{C}$ 
16    $\{b, untouched\} \leftarrow \{b', untouched'\}$ 
17
18 Calculate  $\phi_d$  via Eq. (10)
19  $ctrl \leftarrow$  set of  $k$  indices where the bits of  $b'$  are 1
20  $in \leftarrow$  any index where  $b'$  has a 0
21 Add gate  $c_{ctrl} Ph(\phi_d)$  acting on qubit  $in$  to  $\mathcal{C}$ 
22 return  $\mathcal{C}$ 

```

---

cover all possible sparsity configurations. To this end, we need the generalized RBS gate,  $R_{out}^{in}(\theta, \phi)$  in Eq. (3). With this  $(m + m')$ -qubit gate, it is possible to create superpositions of computational basis states of different Hamming weights.

The procedure to build the circuit is given in Algorithm 3 for complex data and proceeds as follows. Based on the first bitstring address  $b_1$ , we apply Pauli- $X$  gates to generate the initial state  $|b_1\rangle$ . Then, we use Algorithm 2 to extract the generalized RBS gate parameters (input, output, and control qubits) needed to generate the superposition between  $|b_1\rangle$  and  $|b_2\rangle$ , and add the gate  $R_{out_1, \dots, out_m}^{in_1, \dots, in_m}(\theta_1, \phi_1)$  to the circuit; the angles are computed from Eqs. (9) and (10). The same procedure is repeated for all elements in the tuple in Eq. (12), adding  $s - 1$  (possibly controlled and generalized) RBS gates to the resulting circuit. Algorithm 2 can output lists of inputs and outputs with different lengths (i.e.,  $m \neq m'$ ), depending on whether an increase or decrease of Hamming weight is needed to superpose  $|b_j\rangle$  and  $|b_{j+1}\rangle$ . The auxiliary-qubit-free circuit architecture output by Algorithm 3 will depend on the particular sparsity structure of the data vector  $y$ . In the case of real-data encoding, we compute the angles  $\theta_j$  from Eq. (7) instead of calculating  $\theta_j$  and  $\phi_j$  via Eqs. (9) and (10), in addition to not executing lines 18–21 of Algorithm 3. The worst-case bound on the total number of CNOTs can be calculated using the compilation of gRBS gates presented in Table I in Appendix A 1. An explicit example for  $n = 6$

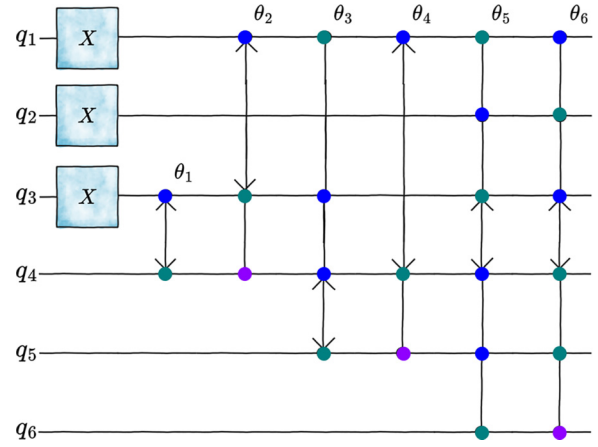


FIG. 4. Sparse quantum data loader. An example with  $n = 6$  qubits and sparsity  $s = 7$  with  $x_j \neq 0$  at positions  $b_j \in \{7, 11, 14, 19, 26, 37, 58\}$  for real data. Colors are used to indicate the input (blue), output (green), and control (violet) qubits of each gRBS gate. Angles are calculated using Eq. (7) for the nonzero entries. The execution of Algorithm 3 to construct this circuit is shown in Table III in Appendix B. This circuit can be compiled using at most 174 CNOTs (see Table I in Appendix A 1).

qubits and sparsity  $s = 7$  is illustrated in Fig. 4, and in Table III in Appendix B.

In Appendix A 2, we show a numerical comparison between the encoders from Algorithm 3, Ref. [36] (as implemented in the QCLIB package [37]), and Ref. [38] (as implemented in the QISKIT package [39]). We compare, as a function of the number of qubits  $n$ , the average CNOT-gate count to encode 100 random  $2^n$ -dimensional data vectors  $y$  of sparsity  $s \in \{n, n^2\}$ . For each instance, we used a Haar-random vector of nonvanishing amplitudes  $x$  and uniformly sampled computational basis state addresses  $|b_j\rangle$ . The results indicate that the average CNOT-gate count of Algorithm 3 and of Ref. [36] scales as  $\mathcal{O}(\text{poly}(n))$  for both sparsity levels  $s$ . Meanwhile, the algorithm from Ref. [38] scales as  $\mathcal{O}(2^n)$  regardless of the sparsity  $s$ . Moreover, Ref. [36]’s encoder displays an improvement of  $\mathcal{O}(n)$  when compared to the results for Algorithm 3; however, while our sparse encoder is free of auxiliary qubits, the sparse encoder from Refs. [36,37] uses  $\mathcal{O}(n)$  auxiliary qubits.

## V. BINARY ENCODER

In this section, we show how to stack together the dense HW- $k$  encoders introduced in Sec. III to encode data on multiple constant-HW subspaces (e.g.,  $\text{Load}_{B_{k_1} \cup B_{k_2}}$  with  $k_1 < k_2$ ). The strategy is to populate the different fixed-HW subspaces in ascending order while using a generalized RBS gate to connect the two subsequent subspaces. For instance,  $\text{Load}_{B_{k_1} \cup B_{k_2}}$  can be built as  $\text{Load}_{B_{k_1}}$  followed by a gRBS gate that increases  $k_1$  to  $k_2$ , and then  $\text{Load}_{B_{k_2}}$ ; in

the special case of successive HWs, i.e.,  $k_2 = k_1 + 1$ , the gRBS can be replaced by a  $k_1$ -controlled  $R_y$  gate.

In particular, this strategy can be used to build a full binary-basis amplitude encoder by stacking dense HW- $k$  encoders of all possible  $0 \leq k \leq n$  in ascending order. We will first explain the general ideal for real-valued data and will later generalize it to complex data. The procedure is as follows: (i) first, initialize the circuit in the state  $|0^n\rangle$ ; (ii) apply an  $R_y$  gate in the last qubit to generate the superposition  $\cos \theta_1 |0^n\rangle + \sin \theta_1 |10^{n-1}\rangle$  according to Eq. (1); (iii) use  $10^{n-1}$  as the initial bitstring of the Ehrlich algorithm (Algorithm 1), creating the circuit  $\text{Load}_{B_1}$ , which is responsible for adding all HW-1 computational basis states to the superposition; (iv) apply a controlled  $R_y$  rotation to add the first computational basis state of HW-2 to the superposition; (v) use the state encoded in the previous step as the input of the algorithm that will generate  $\text{Load}_{B_2}$ ; (vi) iterate over steps (iv) and (v) until all HWs are populated, and the last multicontrolled  $R_y$  generates the superposition with  $|1^n\rangle$ . The (multi)controlled  $R_y$  gates applied between  $\text{Load}_{B_k}$  and  $\text{Load}_{B_{k+1}}$  should be chosen such that the last computational basis state added to the superposition is associated to a bitstring that is a viable input for the Ehrlich algorithm (Theorem 2.4 in Ref. [34]). Alternatively, we can use the Hamming-weight-increasing step

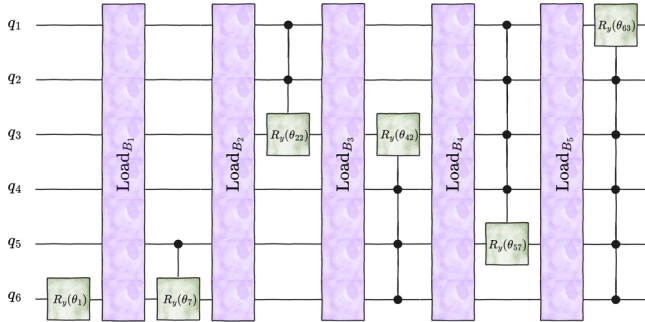


FIG. 5. Binary-basis amplitude encoder based on HW- $k$  encoders. Encoding of a real, 64-dimensional vector  $\mathbf{x}$  on a six-qubit quantum state. The initial state is the zero state,  $|000000\rangle$ , and the circuit is constructed by stacking HW- $k$  encoders,  $\text{Load}_{B_k}$ , with  $k \in [0, 6]$ , and (controlled)  $R_y$  rotations. An initial  $R_y(\theta_1)$  gate creates a superposition between  $|000000\rangle$  and  $|100000\rangle$ . Subsequently, the bitstring  $10^5$  serves as the input of the Ehrlich algorithm (Algorithm 1) that generates the circuit  $\text{Load}_{B_1}$ , responsible for the encoding of amplitudes associated to all six computational basis states of HW-1. Next, based on the last computational basis state added to the superposition by the previous  $\text{Load}_{B_1}$ , the controlled  $R_y(\theta_7)$  rotation adds the first computational basis state of HW-2. The bitstring associated with this state is then used as input for the next Ehrlich algorithm. The procedure is repeated until the last amplitude is encoded in the  $|111111\rangle$  state. The gate parameters  $\{\theta_j\}_{j \in [63]}$  are calculated using Eq. (7). See Table IV in Appendix B for more details. This circuit can be compiled using at most 1048 CNOTs (see Table I in Appendix A 1).

(iv) using an HW-mixing gRBS gate  $R_{\text{out}}^{\text{in}}(\theta, \phi)$  introduced in Eq. (3); however, since the operations necessary for the binary encoder are local, we chose to use the cheaper multicontrolled  $R_y$  gates (see Table I in Appendix A 1).

In case of complex amplitudes, the (controlled)  $R_y(\theta_j)$  rotations must be replaced by (controlled)  $\text{Ph}(\phi_j)R_y(\theta_j)$  to generate the superposition with the correct phases. The angles  $\{\theta_j\}_{j \in [d-1]}$  and  $\{\phi_j\}_{j \in [d-2]}$  are calculated using Eqs. (9) and (10). The last multicontrolled  $R_y$  gate, however, must be replaced by a multicontrolled  $\text{SU}(2)$  rotation that can be decomposed as  $R_z(\phi_{d-1})R_y(\theta_{d-1})R_z(\lambda)$ . The angles  $\phi_{d-1}$  and  $\lambda$  are calculated as

$$\begin{aligned}\phi_{d-1} &= \frac{1}{2} (\arg(x_d) - \arg(x_{d-1})); \\ \lambda &= -\frac{1}{2} (\arg(x_d) + \arg(x_{d-1})) + \sum_{j=1}^{d-3} \phi_j.\end{aligned}\quad (14)$$

An example of the binary encoder for  $n = 6$  qubits is shown in Fig. 5. The total CNOT-gate count is given in Lemma 4.

*Lemma 4* (CNOT count of the binary encoder). Our binary encoder for  $\mathbf{x} \in \mathbb{R}^d$  uses

$$\begin{aligned}\# \text{CNOTs} &\leq \frac{1}{12} (13n^4 - 58n^3 + 119n^2 - 50n + 120) \\ &\quad + \sum_{k=5}^{n-1} \left[ \binom{n}{k} (16k - 22) - 2 \right].\end{aligned}\quad (15)$$

*Proof.* First, it is important to notice that the two “tricks” mentioned in Sec. III to reduce the number of controls (i.e., eliminating initial controls and constructing an HW- $(n-k)$  encoder from the corresponding HW- $k$  encoder) cannot be used here, since the initial state for each  $\text{Load}_{B_k}$  in Fig. 5 contains a superposition of states. Therefore, the cost of each  $\text{Load}_{B_k}$  is given by the upper bound  $\left[ \binom{n}{k} - 1 \right] C_{k-1}$  in the general expression in Eq. (6), where  $C_{k-1}$  is the cost of compiling a single  $(k-1)$ -controlled  $R_{\text{out}}^{\text{in}}(\theta)$  gate. After each  $\text{Load}_{B_k}$  is applied, one needs a  $k$ -controlled  $R_y$  gate to increase the HW to  $k+1$ . The total CNOT count of the circuit is therefore  $\sum_{k=0}^{n-1} \left\{ \left[ \binom{n}{k} - 1 \right] C_{k-1} + \chi_k \right\}$ , where  $\chi_k$  is the CNOT cost of a  $k$ -controlled  $R_y$ . Substituting the values of  $C_{k-1}$  and  $\chi_k$  calculated in Appendix A 1 immediately leads to Eq. (15). (See Table I for a summary, and Lemmas A1 and A2 for the derivation.) The first line contains exact CNOT counts coming from  $k = 1$  to 4, while the second line contains upper bounds coming from  $k \geq 5$ . ■

Numerically, we observe the CNOT-gate count of Eq. (15) to be  $\leq 8n2^n$  (see Appendix A 2). Asymptotically, this is the same scaling obtained by the sparse encoder

from Ref. [36]. While the CNOT-gate count of Ref. [22] is  $\mathcal{O}(2^n)$ , they perform their amplitude encoding in the Schmidt basis with at most  $2^{n/2}$  coefficients. In contrast, we explore all  $2^n$  computational basis states.

## VI. HARDWARE DEPLOYMENT AND NUMERICAL RESULTS

In this section, we show a proof-of-principle deployment of the dense HW- $k$  encoder on IonQ’s Aria-1 quantum processor. We also numerically demonstrate that the encoder circuits can be used as an ansatz for a variational quantum algorithm. Then, we analyze the fidelity of the encoder under circuit noise.

### A. Quantum hardware demonstration

To demonstrate our encoding protocol, we encode a  $q$ -Gaussian probability distribution on the Aria-1 quantum processor from IonQ [30]. The  $q$ -Gaussian probability density function  $p_{q,\beta}(x)$ , where  $q \in (-\infty, 3)$  and  $\beta \in (0, \infty)$ , is proportional to the  $q$ -exponential function  $e_q(-\beta x^2)$ , defined as  $e^x$  if  $q = 1$ ;  $[1 + (1 - q)x]^{1/(1-q)}$  if  $q \neq 1$  and  $1 + (1 - q)x > 0$ ; and  $0^{1/(1-q)}$  otherwise [40–42]. This family of distributions has a wide range of applications, e.g., nonextensive statistical mechanics [43], finance [44, 45], metrology [46], and biology [47]. It is worth noting that in general, the  $q$ -Gaussian is a non-log-concave distribution, falling outside of the scope of other encoding strategies [12]. Here, we chose the parameters  $q = 3/2$  and  $\beta = 2$ .

We use the circuit in Fig. 3 for six qubits and Hamming weight  $k = 2$ , which allows us to encode a data vector of size  $d = \binom{6}{2} = 15$ . This vector corresponds to a discretization of the distribution truncated to the interval  $x \in [-2, 2]$ . For this circuit, the first four parameterized rotations are uncontrolled RBS gates, while the last ten RBSs require one control each. Each RBS gate is compiled using two CNOT gates, while each controlled RBS requires six CNOTs, leading to a circuit with 68 CNOTs in total (see Table I in Appendix A 1); however, the Aria-1 processor uses the Mølmer-Sørensen (MS) gate as the two-qubit native gate [30]. In this case, it is possible to implement one-controlled  $R_y$  rotations using only one MS gate. This allows us to implement the aforementioned circuit using 48 MS gates.

In Fig. 6, we show the results of the implementation on quantum hardware. In the top panel, we plot the probability distribution estimated from the demonstration by running the circuit  $10^4$  times and measuring on the computational basis on each qubit to recover the encoded data. The solid orange line shows the empirical probability distribution estimated from the raw data from the Aria-1 quantum processor. The solid purple line shows the estimated probability distribution after error mitigation using *Clifford data regression* (CDR) [31,32]. CDR models hardware noise by simulating near-Clifford circuits similar to the one at

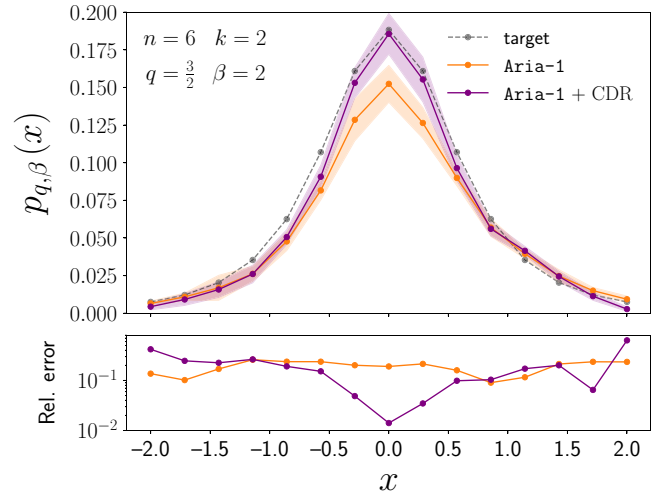


FIG. 6. Deployment of demonstration on IonQ. Top: implementation of the encoding of a  $q$ -Gaussian probability distribution  $p_{q,\beta}(x)$  on IonQ’s Aria-1 processor. The domain of  $p_{q,\beta}(x)$  was truncated to  $x \in [-2, 2]$  and discretized into  $d = 15$  points (dashed black line). The circuit deployed used  $n = 6$  and  $k = 2$  and is shown in Fig. 3. The solid purple (orange) line represents results with (without) error mitigation using Clifford data regression (CDR). The shaded areas represent the uncertainty regions estimated via bootstrapping. Bottom: relative error of the implementation of  $p_{q,\beta}(x)$  with respect to the target function, in log scale.

hand and then applying the trained response function to the raw data. The training of the response functions was performed using IonQ’s capabilities for classical simulation of noisy circuits (see Appendix D for details). The dashed black line shows the ideal target distribution. All relative errors concerning the target distribution are plotted in Fig. 6 (bottom) in log scale. We see that the data recovered from the demonstration differs from the target distribution by 10%–30% in relative error. After CDR is applied to the hardware data, relative errors can be improved. For instance, the CDR protocol reduced the relative error at the peak of the distribution by one order of magnitude. This improvement can also be seen in the overall state infidelity  $1 - \text{Tr}(\rho\sigma)$  ( $\sigma$  being the state prepared by the noisy circuit), which decreased from 0.17 to 0.08 after error mitigation. We attribute these discrepancies in the hardware results to the fact that Aria-1 is a noisy intermediate-scale quantum (NISQ) device. The noise present in the circuit creates amplitudes “outside” of the subspace of interest. Consequently, there is a reduction in the probability density “inside” said subspace. We showed that this effect can be reduced by the application of error-mitigation techniques such as CDR.

### B. HW- $k$ encoder under local depolarizing noise

Motivated by the results in Sec. VIA, in this section, we numerically investigate the robustness of the HW- $k$

encoder in the presence of circuit noise as a function of both  $k$  and the noise strength. For the noise model, given a pure quantum state  $\rho$  and any pair of qubits labeled by  $A$ , we assume every CNOT gate acting on  $A$  is followed by a depolarizing channel  $\mathcal{D}_p$ ,

$$\mathcal{D}_p(\rho) = (1 - p)\rho + p\text{Tr}_A(\rho) \otimes \frac{\mathbb{1}}{4}, \quad (16)$$

where  $p$  is the depolarizing strength,  $\mathbb{1}$  is the two-qubit identity matrix, and  $\text{Tr}_A(\cdot)$  is the partial trace over  $A$ . This noise model is compatible with reported noise models derived from randomized benchmarking techniques [48]. Since two-qubit gate errors are the main source of circuit noise in NISQ devices, we assume that all single-qubit gates are noiseless. We compiled the circuits resulting from Algorithm 1 into CNOTs, single-qubit gates, and multicontrolled  $R_y$  rotations as described in Secs. II and III B 1, and Appendix A 1. While the circuits were simulated using the QIBO package [49], the multicontrolled  $R_y$  gates were compiled into CNOTs and single-qubit gates using the integration between the QCLIB library [37] and the QISKIT package [39].

In Fig. 7 (left panel), we show a log-log plot of the resulting infidelity for fixed  $k = 3$  as a function of the depolarizing strength  $p$  for  $n \in [5, 9]$ . We observed that for fixed  $n$  and  $k$ , the average infidelity scales polynomially with  $p$ , namely  $\mathcal{O}(p^\gamma)$  with  $\gamma \approx 1$ . For  $p \sim 10^{-3}$ , which is a per-gate noise level compatible with several of the currently available quantum platforms [50–53], the infidelities in the range of  $n$  studied are between  $10^{-2}$  and  $10^{-1}$ . In Fig. 7 (right panel), we plot the infidelity (in log scale) as a function of  $k$  for fixed  $n = 8$ , and  $p \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ . We observed that for fixed  $n$  and  $p$ , the average infidelity scales exponentially with  $k$  as  $\mathcal{O}((1 - p)^{N_{\text{CNOT}}(n,k)})$ , where  $N_{\text{CNOT}}(n, k)$  is the total CNOT gate count given by Theorem 2. This scaling can be intuitively explained from the fact that the depolarizing channel is isotropic; specifically, the composition of  $N_{\text{CNOT}}$  local depolarizing channels is equivalent to a global depolarizing channel with strength  $1 - (1 - p)^{N_{\text{CNOT}}}$ . As expected from NISQ devices, the accumulation of errors in deep circuits leads to the preparation of states with high infidelity compared to the target states; however, as reflected by the linear scaling in  $p$ , we see that each order-of-magnitude reduction in  $p$  also yields an order-of-magnitude reduction in infidelity.

### C. HW- $k$ encoder as a variational quantum ansatz

Here, we demonstrate the use of the encoders presented in Secs. III–V as a variational quantum ansatz. We focus on the HW- $k$  encoder of dense real-valued data, though results immediately extend to complex-valued or sparse data, as well as binary encoders.

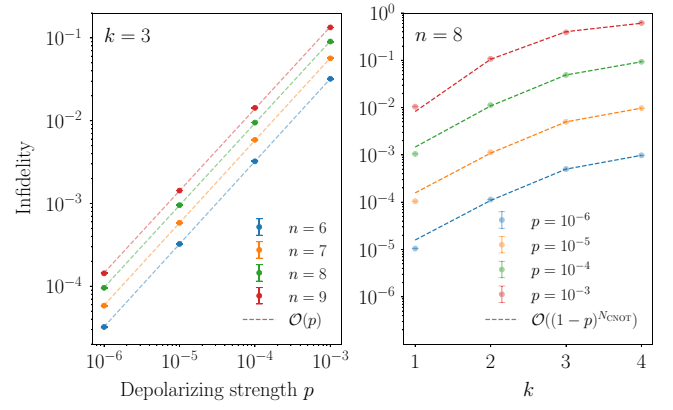


FIG. 7. Performance of HW- $k$  encoder under local depolarizing noise model. Circuits for real-valued data are compiled into CNOTs and single-qubit gates (see Secs. II and III B 1). Each CNOT is followed by a two-qubit depolarizing channel of strength  $p$ . Left: state infidelity as a function of depolarizing strength  $p$ , averaged over ten Haar-random instances, for fixed  $k = 3$  and  $n \in [5, 9]$ . Dashed lines represent a linear fit  $\mathcal{O}(p)$ . Right: state infidelity as a function of  $k$ , averaged over ten Haar-random instances for fixed  $p \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$  and  $n = 8$ . Dashed lines represent the curve fit of the function  $\mathcal{O}((1 - p)^{N_{\text{CNOT}}})$  to the numerical data, where  $N_{\text{CNOT}} \equiv N_{\text{CNOT}}(n, k) = \mathcal{O}(k^{\binom{n}{k}})$  is the number of CNOTs in each circuit for a given  $n$  and  $k$ .

We first recall that, given  $n$  and  $k$ , the quantum circuits generated by Algorithm 1 parameterize the subspace spanned by the basis  $B_k$  of size  $d = \binom{n}{k}$  defined in Eq. (5). Therefore, this parametrization is ideally suited for solving optimization tasks that have some type of Hamming-weight constraints. For instance, this is the case for ground-state energy estimation of molecular Hamiltonians [1, 2], which have particle-number-preserving symmetry, as well as portfolio optimization constrained by a fixed number of assets [27]. Without loss of generality, here, we illustrate this feature using a quantum chemistry example. Given a molecule having  $k$  electrons and a basis set describing its orbitals, the Jordan-Wigner transform maps its fermionic Hamiltonian into a qubit Hamiltonian  $H$  where each qubit corresponds to one of the  $n$  molecular spin orbitals and the state  $|1\rangle$  ( $|0\rangle$ ) is associated with an occupied (unoccupied) spin orbital. The ground state of the qubitized molecular Hamiltonian is therefore constrained to a subspace  $B_k$  of the form in Eq. (5), and one can use the corresponding HW- $k$  circuit  $\text{Load}_{B_k}$  as a variational ansatz to approximate the ground state by classically optimizing the RBS gate angles  $\theta := \{\theta_j\}_{j \in [d-1]}$  to minimize the energy function

$$E(\theta) := \langle 0^n | \text{Load}_{B_k}^\dagger(\theta) H \text{Load}_{B_k}(\theta) | 0^n \rangle. \quad (17)$$

Below, we numerically demonstrate this by finding the ground-state energy of the  $\text{H}_4$  molecule, with the qubitized

Hamiltonian obtained via the quantum dataset from PENNY-LANE [54,55]. Using the STO-3G basis set, this molecule can be described by an  $n = 8$  qubit Hamiltonian (corresponding to eight spin orbitals) with  $k = 4$  electrons in the ground state. The results are presented in Fig. 8. In the main panel, we plot the ground-state energy (in Hartree) of the  $H_4$  molecule as a function of the bond length (in Angstrom) between the hydrogen atoms. We estimate the energies using our amplitude encoder  $\text{Load}_{B_4}$  as an ansatz (solid purple line) and compare our results with the “pyramid” HW-preserving ansatz from Ref. [28] (solid orange line), composed solely of uncontrolled RBS gates. For a fair comparison, we allow this ansatz to have the same number of parameters as ours, i.e.,  $\binom{n}{k} - 1 = 69$  parameters. We did not observe any improvement in expressivity of the pyramid ansatz by adding more parameters beyond this number, hence we did not compare the two ansätze by equating their number of two-qubit gates. We leveraged the integration between QIBO and PYTORCH [56] and used the Adam optimizer [57] with a fixed learning rate  $\eta = 10^{-3}$  as the gradient-descent method of choice for both ansätze. We plot the best energy results after 1000 epochs, averaged over 50 Haar-random initializations. The

“exact” ground-state energies, calculated via brute-force diagonalization of the corresponding Hamiltonians, are plotted with a solid black line. We see that our encoder approximates the ground-state energy of the  $H_4$  molecule very well throughout the entire energy curve, unlike the pyramid ansatz, whose results worsen as the bond length increases. The inset of Fig. 8 shows the energy estimation errors relative to the exact ground-state energies. We see that after 1000 epochs, the HW- $k$  ansatz, on average, reached chemical accuracy, while the pyramid ansatz remained at least an order of magnitude away throughout the entire bond-length range.

## VII. CONCLUSIONS

We provide an efficient and explicit classical algorithm to construct, gate by gate, a quantum circuit that uploads an arbitrary data vector into a subspace of fixed-Hamming-weight quantum states. The quantum circuit uses the minimum number of parameterized gates needed to express generic data of a given dimension. The construction uses (generalized) RBS gates and allows us to precisely state the quantum resources needed for its execution, as well as to deploy a proof-of-concept instance on real quantum hardware. We also provide the tools needed to further explore quantum encoders for this subspace and beyond.

As shown for the sparse and binary encoders, our HW- $k$  encoder can be used as a subroutine to power more complex algorithms. These two examples are but a small sample, and future work will explore more avenues. Still, we remark on the importance of binary encoding, as most algorithms would benefit from robust state preparation work on this basis. Other encoding schemes, such as the unary-basis encoder, can deploy basis-change circuits [58] to allow for further postprocessing. A generalization of this circuit to the HW- $k$  basis would open possibilities for more efficient general state preparation.

Furthermore, our algorithm brings interesting implications in the context of quantum machine learning. Variational ansätze that explore a constrained space are popular due to their ability to mitigate barren plateaus [3]. As shown by the numerical demonstration with the  $H_4$  molecule, our HW- $k$  encoder, due to its expressivity, has the potential to improve the performance of optimizations that involve particle-preserving symmetry. Further investigation of this potential is needed, and we leave it as an open question for future work. Algorithms with space compression beyond linear, such as our HW- $k$  encoder, also represent a promising direction for other machine-learning schemes [59] and show how exploring these subspaces can be successful.

Lastly, we believe that explicit constructive algorithms for state preparation, accompanied by precise analysis of the quantum resources required, are essential for achieving a useful quantum advantage.

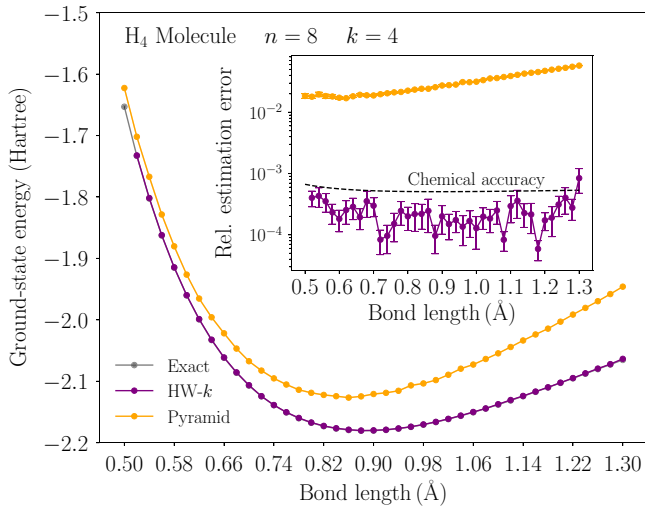


FIG. 8. Hamming-weight- $k$  encoder as a variational quantum ansatz. Ground-state energy (in Hartree) for the  $H_4$  molecule as a function of the bond length (in Angstrom). The purple curve represents the energies obtained using Algorithm 1 with  $n = 8$  and  $k = 4$ , while the orange curve was obtained using the “pyramid” circuit architecture from Ref. [4]. The number of gradient-descent epochs is fixed in both cases to 1000, the results are averaged over 50 random initializations, and the error bars are smaller than the markers. The *exact* energy curve calculated from brute-force diagonalization is plotted in gray and is nearly indistinguishable from the purple curve. Inset: energy estimation error relative to the exact ground-state energy as a function of the bond length. The color scheme is the same as that in the main plot. Relative chemical accuracy is plotted as a dashed black line.

*Note added.* We note that during the completion of this paper, an independent work [60] appeared within the context of Bethe state preparation for integrable spin chains. Their results have partial overlap with our HW- $k$  encoders for dense data presented in Sec. III.

## ACKNOWLEDGMENTS

We thank Ariel Bendersky and André J. Ferreira-Martins for insightful discussions. We also thank Jadwiga Wilkens for the availability of the *Quantum Circuit Library* [61].

## DATA AVAILABILITY

The data that support the findings of this article are not publicly available because of legal restrictions preventing unrestricted public distribution. The data are available from the authors upon reasonable request.

## APPENDIX A: GATE COMPILATIONS AND CNOT-GATE COUNTS

### 1. Compilation of multicontrolled gates

Here, we calculate the CNOT cost of the RBS gate, the gRBS gate, and their multicontrolled versions. For this, Lemma 1 will be useful.

*Lemma 5* (CNOT cost of a multicontrolled  $U(2)$ ). Let  $\chi_\ell(U)$  be the number of CNOT gates required to implement an  $\ell$ -controlled single-qubit gate  $U \in U(2)$ , where  $\ell = 0$  corresponds to no controls. Then,  $\chi_0(U) = 0$ ,  $\chi_1(U) = 2$ ,  $\chi_2(U) = 4$ ,  $\chi_3(U) = 12$ ,  $\chi_4(U) = 36$ , and, for  $\ell \geq 5$ ,  $\chi_\ell(U) \leq 20\ell - 18$ . Moreover, if  $U = R_y(\theta)$  or  $U = R_z(\theta)$ , the bound for  $\ell \geq 5$  can be improved to  $\chi_\ell(U) \leq 16\ell - 24$ .

*Proof.* Exact compilations for  $\ell \leq 4$  follow from Refs. [19,63]. The upper bounds for  $\ell \geq 5$  are given by Ref. [62]. ■

Depending on the chosen decomposition for the RBS gate, the gRBS gate admits different decompositions into

controlled  $R_y$  and  $R_z$  gates (e.g., see Fig. 1). Using Lemma 1, we can now calculate the CNOT cost of a multicontrolled gRBS gate as follows.

*Lemma 6* (CNOT cost of a multicontrolled complex gRBS gate). Let  $\chi_\ell$  be as defined in Lemma 1,  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}(\theta, \phi) \equiv c_{\text{ctrl}_1, \dots, \text{ctrl}_\ell} R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}$  be as in Sec. II, and  $\mu = \ell + m + m'$ . For  $\phi = 0$ , the number of CNOTs to compile this  $\mu$ -qubit gate is

$$2(m + m' - 1) + \min(2\chi_{\mu-2}(R_y), \chi_{\mu-1}(R_y)).$$

If  $\phi \neq 0$ , then the cost is

$$2(m + m' - 1) + \min(4\chi_{\mu-2}(R_y), \chi_{\mu-1}(U)).$$

*Proof.* (i) Case  $\phi = 0$ : Since the  $R_z(\phi)$  gates are absent, the  $\ell$  controls can act directly on the  $R_y$  gates (see Fig. 2). As a result,  $c_{\text{ctrl}_1, \dots, \text{ctrl}_\ell} R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}$  is equivalent to  $2(m - 1) + 2(m' - 1) + 2$  CNOTs and two  $(\mu - 2)$ -controlled  $R_y$  gates for the compilation in Fig. 1 (top), or  $2(m - 1) + 2(m' - 1) + 2$  CNOTs and a single  $(\mu - 1)$ -controlled  $R_y$  gate for the compilation in Fig. 1 (bottom). The claim then follows from compiling the controlled  $R_y$  gates using Lemma 1 and choosing the less costly compilation between the two. Figure 1 (top) is the best compilation for  $\mu \leq 4$ , while Fig. 1 (bottom) is the best otherwise.

(ii) Case  $\phi \neq 0$ : When using the RBS compilation in Fig. 1 (top) to build a complex gRBS, it is necessary to perform 2  $(\mu - 2)$ -controlled  $R_y$  gates as well as 2  $(\mu - 2)$ -controlled  $R_z$  gates. Since these gates have the same CNOT cost [62], we express the total cost as four times the cost of one  $(\mu - 2)$ -controlled  $R_y$  gate. On the other hand, the product  $R_z(\phi)R_y(\theta)$  in the compilation shown in Fig. 1 (bottom) corresponds to a single  $SU(2)$  of the form  $U = e^{i\lambda W}$ , where  $\lambda(\theta, \phi) = \arccos(\cos \theta \cos \phi)$  and  $W(\theta, \phi) = 1/\sin \lambda(\theta, \phi)(-\sin \theta \sin \phi X + \sin \theta \cos \phi Y + \cos \theta \sin \phi Z)$ . The proof then proceeds identically to Case (i). Figure 1 (top) is the best compilation for  $\ell = 0$  and  $m = m' = 1$ , while Fig. 1 (bottom) is the best otherwise. ■

TABLE I. CNOT count of multicontrolled  $X$ ,  $R_y$ , RBS, and gRBS gates used throughout this work. Details are provided in Sec. A 1. For each multicontrolled RBS and gRBS gate, the CNOT count corresponds to the best between the two compilations in Figs. 1 and 2. For multicontrolled  $R_y$  gates, we use the CNOT counts provided in Refs. [19,62]. The numbers in the second column are the  $\chi_\ell$  in Lemma 1, while the second and third columns correspond to  $C_\ell$  used in Eq. (6) for the real and complex cases, respectively.

| No. of controls | Gate               |                    |                                      |                                            |                                                                                      |                                                                                            |
|-----------------|--------------------|--------------------|--------------------------------------|--------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|                 | $X$                | $R_y(\theta)$      | $R_{\text{out}}^{\text{in}}(\theta)$ | $R_{\text{out}}^{\text{in}}(\theta, \phi)$ | $R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}(\theta)$ | $R_{\text{out}_1, \dots, \text{out}_{m'}}^{\text{in}_1, \dots, \text{in}_m}(\theta, \phi)$ |
| $\ell = 0$      | 0                  | 0                  | 2                                    | 2                                          | $\leq 18(m + m') - 42$                                                               | $\leq 22(m + m') - 20$                                                                     |
| $\ell = 1$      | 1                  | 2                  | 6                                    | 6                                          | $\leq 18(m + m') - 26$                                                               | $\leq 22(m + m')$                                                                          |
| $\ell = 2$      | 6                  | 4                  | 10                                   | 14                                         | $\leq 18(m + m') - 10$                                                               | $\leq 22(m + m') + 20$                                                                     |
| $\ell = 3$      | $\leq 24$          | 12                 | 26                                   | 38                                         | $\leq 18(m + m') + 6$                                                                | $\leq 22(m + m') + 40$                                                                     |
| $\ell = 4$      | $\leq 40$          | 36                 | $\leq 58$                            | $\leq 84$                                  | $\leq 18(m + m') + 22$                                                               | $\leq 22(m + m') + 60$                                                                     |
| $\ell \geq 5$   | $\leq 16\ell - 24$ | $\leq 16\ell - 24$ | $\leq 16\ell - 6$                    | $\leq 20\ell + 4$                          | $\leq 18(m + m') + 16\ell - 42$                                                      | $\leq 22(m + m') + 20\ell - 20$                                                            |

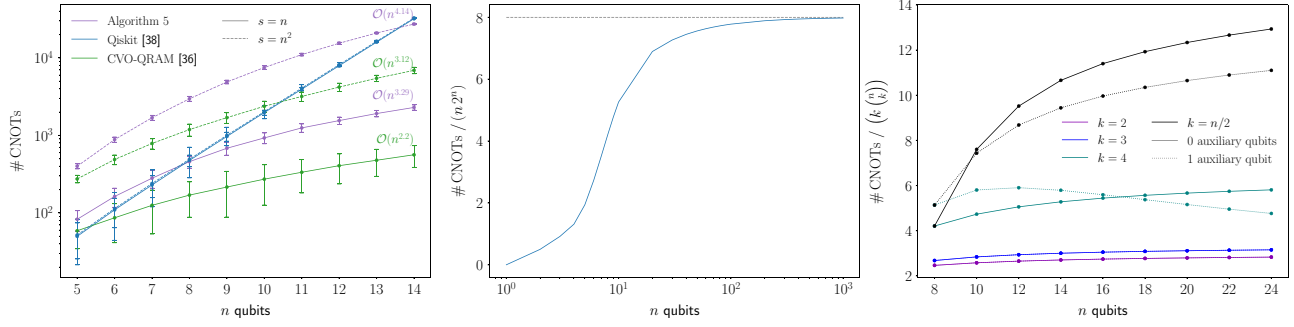


FIG. 9. Left: comparison of the average CNOT cost of Algorithm 3 with Refs. [36,38]. CNOT-gate count (in log scale) as a function of the number of qubits  $n$ . We compare the costs of Algorithm 3 (purple lines) with the algorithms from Refs. [36] (green lines) and [38] (blue lines). We present results averaged over 100 Haar-random states  $\in \mathbb{C}^s$ , each embedded onto a data vector  $\mathbf{x} \in \mathbb{C}^{2^n}$ . Sparsity levels  $s \in \{n, n^2\}$  are represented by solid and dashed lines, respectively. Error bars indicate one standard deviation. Middle: CNOT-gate count of the binary encoder. Count based on Eq. (15) in Lemma 4 as a function of the number of qubits  $n$  (in log scale). We used the compilation of controlled  $R_y$  rotations from Table I. We observe an asymptotic behavior toward  $8n2^n$  for large  $n$ . Right: CNOT cost of Algorithm 1 with the addition of one auxiliary qubit. We compare the CNOT cost of Algorithm 1 (solid lines) with the CNOT cost of the same algorithm with the additional step of including a clean auxiliary qubit and using it to heuristically remove redundant controlled operations (dotted lines). The comparison is performed for Hamming weights  $k \in \{2, 3, 4, n/2\}$ . For  $k \in \{2, 3\}$  (purple and blue lines, respectively), the auxiliary qubit does not help decrease the CNOT count significantly, if at all, regardless of  $n$ . For  $k = 4$  (green lines), the addition of an auxiliary qubit increases the number of CNOTs in the circuits for  $n \leq 18$ . This is due to the cost of additional controlled Toffoli gates. For  $n > 18$ , the heuristic algorithm starts removing CNOTs from the original circuit, and the advantage with respect to the auxiliary-qubit-free circuit increases with  $n$ . For  $k = n/2$  (black lines), we can see that the auxiliary qubit starts contributing at  $n > 12$ , and the benefit increases with  $n$ .

Explicit CNOT counts for all the gates used throughout this work are summarized in Table I.

## 2. CNOT cost of sparse and binary encoders

Using the results of Sec. A 1, we numerically investigate the CNOT cost of implementing the sparse and binary encoders from Secs. IV and V, respectively.

Figure 9(a) compares, as a function of the number of qubits  $n$ , the average CNOT cost of the sparse encoder presented in Algorithm 3 with the QRAM-like sparse encoder from Ref. [36] as well as the auxiliary-qubit-free state preparation from Ref. [38]. While the former is implemented in the QCLIB package [37], the latter is currently implemented in the QISKIT package [39]. We used the QIBO package [49] to implement Algorithm 3. The CNOT cost is averaged over the encoding of 100 random sparse vectors  $\mathbf{y}$  of dimension  $2^n$  and sparsity  $s \in \{n, n^2\}$ . Each  $\mathbf{y}$  was obtained by sampling a Haar-random vector  $\mathbf{x} \in \mathbb{C}^s$  and embedding it into a  $2^n$ -dimensional sparse data vector by associating each entry  $x_j$  with a random computational basis state  $|b_j\rangle$  sampled from the uniform distribution. Reference [38] displayed a scaling of  $\mathcal{O}(2^n)$  in CNOT-gate count, independent of the sparsity  $s$  (solid and dashed blue lines). Even though Ref. [38] provides a method for state preparation without auxiliary qubits, it is shown not to be well suited to the preparation of sparse states. The sparse encoder from Ref. [36] presented a scaling of approximately  $\mathcal{O}(n^{2.2})$  for  $s = n$  (solid green line) and of approximately  $\mathcal{O}(n^{3.12})$  for  $s = n^2$  (dashed green line). This is a

significant improvement when compared with Ref. [38]; however, the algorithm from Ref. [36] requires  $\mathcal{O}(n)$  auxiliary qubits to harness that improvement. Meanwhile, the sparse encoder in Algorithm 3 scales as approximately  $\mathcal{O}(n^{3.29})$  for  $s = n$  (solid purple line) and as approximately  $\mathcal{O}(n^{4.14})$  for  $s = n^2$  (dashed purple line). These scalings are approximately  $\mathcal{O}(n)$  inferior to those from Ref. [36] for both sparsity levels tested. We highlight, however, that Algorithm 3 is free of auxiliary qubits. This points to a trade-off between auxiliary qubits and CNOT-gate cost. Algorithm 3 allows one to have a sparse amplitude encoder that is both poly( $n$ ) in CNOT cost and auxiliary-qubit free, at the price of having to implement linearly deeper circuits.

Figure 9(b) shows the asymptotic behavior  $\sim 8n2^n$  of the total gate count for the binary encoder (see Lemma 4 in the main text).

## APPENDIX B: EXPLICIT EXAMPLES

Here, we present three tables that provide explicit examples demonstrating the execution of the classical algorithms used to generate the quantum circuits for our HW encoder algorithms. Each table corresponds to one of the figures present in the main text, namely: Table II illustrates the real dense encoder (Algorithm 1) and corresponds to Fig. 3; Table III illustrates the real sparse encoder (Algorithm 3) and corresponds to Fig. 4; finally, Table IV illustrates the binary encoder that combines the dense HW- $k$  for all possible values of  $k \in [0, n]$  and corresponds to Fig. 5.

TABLE II. Illustration of the dense HW- $k$  encoder (Algorithm 1) for  $n = 6$  and  $k = 2$ . Data vector  $\mathbf{x} \in \mathbb{R}^d$ , with  $d = \binom{6}{2} = 15$ . The order of the bitstrings is given by Algorithm 1. Marked bits are represented with an overline. Qubit indices are enumerated from right to left,  $c_{\text{ctrl}} R_{\text{out}}^{\text{in}}$  denotes an RBS acting on input qubit **in** and output qubit **out**, controlled by qubit(s) **ctrl**, with  $\theta_j$  given by Eq. (7). Unnecessary **ctrl** qubits removed by Algorithm 2 are represented by  $\overline{\text{ctrl}}$ . Each gate in a given row represents the operation required to extend the superposition to include the computational basis state represented by the bitstring in the next row.

| $q_6$          | $q_5$          | $q_4$          | $q_3$          | $q_2$          | $q_1$ | Gate                                 |
|----------------|----------------|----------------|----------------|----------------|-------|--------------------------------------|
| $\overline{1}$ | $\overline{1}$ | 0              | 0              | 0              | 0     | $\overline{c_8} R_{1,2}^5(\theta_1)$ |
| $\overline{1}$ | 0              | $\overline{0}$ | $\overline{0}$ | $\overline{0}$ | 1     | $\overline{c_8} R_{1,2}^1(\theta_2)$ |
| $\overline{1}$ | 0              | $\overline{0}$ | $\overline{0}$ | 1              | 0     | $\overline{c_8} R_{1,2}^2(\theta_3)$ |
| $\overline{1}$ | 0              | $\overline{0}$ | 1              | 0              | 0     | $\overline{c_8} R_{1,2}^3(\theta_4)$ |
| $\overline{1}$ | 0              | 1              | 0              | 0              | 0     | $c_4 R_5^6(\theta_5)$                |
| 0              | $\overline{1}$ | $\overline{1}$ | 0              | 0              | 0     | $c_5 R_4^1(\theta_6)$                |
| 0              | $\overline{1}$ | 0              | $\overline{0}$ | $\overline{0}$ | 1     | $c_5 R_2^1(\theta_7)$                |
| 0              | $\overline{1}$ | 0              | $\overline{0}$ | 1              | 0     | $c_5 R_2^2(\theta_8)$                |
| 0              | $\overline{1}$ | 0              | 1              | 0              | 0     | $c_3 R_2^3(\theta_9)$                |
| 0              | 0              | $\overline{1}$ | $\overline{1}$ | 0              | 0     | $c_4 R_1^1(\theta_{10})$             |
| 0              | 0              | $\overline{1}$ | 0              | $\overline{0}$ | 1     | $c_4 R_1^2(\theta_{11})$             |
| 0              | 0              | $\overline{1}$ | 0              | 1              | 0     | $c_2 R_3^4(\theta_{12})$             |
| 0              | 0              | 0              | $\overline{1}$ | $\overline{1}$ | 0     | $c_3 R_1^2(\theta_{13})$             |
| 0              | 0              | 0              | $\overline{1}$ | 0              | 1     | $c_1 R_2^3(\theta_{14})$             |
| 0              | 0              | 0              | 0              | 1              | 1     |                                      |

### APPENDIX C: USING ONE AUXILIARY QUBIT TO REDUCE CNOT COSTS

Here, we present a heuristic method that uses one auxiliary qubit to remove controlled operations in the later

TABLE III. Illustration of the sparse encoder (Algorithm 3) for  $n = 6$  and  $s = 7$ . Data vector  $\mathbf{y} \in \mathbb{R}^d$ , with  $d = 2^6$  and  $s = 7$  nonzero entries. The addresses  $b_j$  are sorted in ascending order of HW (a necessary condition). As a byproduct, this sorting also minimizes the CNOT cost of Algorithm 2. The angles  $\theta_j$  are given by Eq. (7). Each gate in a given row represents the operation needed to add the computational basis state corresponding to the bitstring in the next row to the superposition. The first six elements have Hamming weight  $k = 3$ , leading to the initial controlled gRBS gates being HW-preserving. Meanwhile, the last element has  $k = 4$  and requires a gRBS with  $m \neq m'$  to increase the HW by 1.

| $q_6$ | $q_5$ | $q_4$ | $q_3$ | $q_2$ | $q_1$ | Gate                              |
|-------|-------|-------|-------|-------|-------|-----------------------------------|
| 0     | 0     | 0     | 1     | 1     | 1     | $c_{1,4} R_4^3(\theta_1)$         |
| 0     | 0     | 1     | 0     | 1     | 1     | $c_{1,4} R_3^1(\theta_2)$         |
| 0     | 0     | 1     | 1     | 1     | 0     | $c_{1,4} R_{1,5}^{3,4}(\theta_3)$ |
| 0     | 1     | 0     | 0     | 1     | 1     | $c_{1,5} R_4^1(\theta_4)$         |
| 0     | 1     | 1     | 0     | 1     | 0     | $R_{1,3,6}^{2,4,5}(\theta_5)$     |
| 1     | 0     | 0     | 1     | 0     | 1     | $c_6 R_{2,4,5}^{1,3}(\theta_6)$   |
| 1     | 1     | 1     | 0     | 1     | 0     |                                   |

TABLE IV. Illustration of the binary encoder for  $n = 6$  qubits. A gate (or collection of gates) in a given row represents the operation required to add the computational basis state corresponding to the bitstring in the next row to the superposition. The initial bitstring has  $k = 0$  and (controlled)  $R_y$  gates are used to increase the HW by 1. The first bitstring with HW- $(k + 1)$  has a one-bit difference from the final bitstring of the Ehrlich algorithm with HW- $k$ . Operators  $\text{Load}_{B_k}$  fill up the subspace of HW- $k$ , and the algorithm stops after reaching  $k = n$ . Angles  $\theta_j$  are given by Eq. (7),  $\theta_j$  are used in  $R_y$  gates, and  $\theta_j$  are used in  $\text{Load}_{B_k}$ . The total number of parameters is  $2^n - 1$ .

| $q_6$          | $q_5$          | $q_4$          | $q_3$          | $q_2$    | $q_1$    | Operator                              | $k$ |
|----------------|----------------|----------------|----------------|----------|----------|---------------------------------------|-----|
| 0              | 0              | 0              | 0              | 0        | 0        | $R_y^6(\theta_1)$                     | 0   |
| $\overline{1}$ | 0              | 0              | 0              | 0        | 0        | $\text{Load}_{B_1}$                   | 1   |
| $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$ | $\vdots$ | $\{\theta_2, \dots, \theta_6\}$       | 1   |
| 0              | 1              | 0              | 0              | 0        | 0        | $c_5 R_y^6(\theta_7)$                 | 1   |
| $\overline{1}$ | $\overline{1}$ | 0              | 0              | 0        | 0        | $\text{Load}_{B_2}$                   | 2   |
| $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$ | $\vdots$ | $\{\theta_8, \dots, \theta_{21}\}$    | 2   |
| 0              | 0              | 0              | 0              | 1        | 1        | $c_{1,2} R_y^3(\theta_{22})$          | 2   |
| $\overline{0}$ | $\overline{0}$ | $\overline{0}$ | 1              | 1        | 1        | $\text{Load}_{B_3}$                   | 3   |
| $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$ | $\vdots$ | $\{\theta_{23}, \dots, \theta_{41}\}$ | 3   |
| 1              | 1              | 1              | 0              | 0        | 0        | $c_{4,5,6} R_y^3(\theta_{42})$        | 3   |
| $\overline{1}$ | $\overline{1}$ | $\overline{1}$ | $\overline{1}$ | 0        | 0        | $\text{Load}_{B_4}$                   | 4   |
| $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$ | $\vdots$ | $\{\theta_{43}, \dots, \theta_{56}\}$ | 4   |
| 0              | 0              | 1              | 1              | 1        | 1        | $c_{1,2,3,4} R_y^5(\theta_{57})$      | 4   |
| $\overline{0}$ | 1              | 1              | 1              | 1        | 1        | $\text{Load}_{B_5}$                   | 5   |
| $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$       | $\vdots$ | $\vdots$ | $\{\theta_{58}, \dots, \theta_{62}\}$ | 5   |
| 1              | 1              | 1              | 1              | 1        | 0        | $c_{2,3,4,5,6} R_y^1(\theta_{63})$    | 5   |
| 1              | 1              | 1              | 1              | 1        | 1        |                                       | 6   |

stages of the circuits generated by Algorithms 1 and 2, as mentioned in Sec. III. The results are shown in Fig. 9(c). For a given  $k$ , the maximum number of indices in the **ctrl** set is  $k - 1$ . Thus, consecutive controlled RBS gates that share the same  $k - 1$  indices in their respective **ctrl** sets can be implemented together. The “auxiliary-qubit trick” proceeds as follows. First, we add one auxiliary qubit to the system. Then, a controlled Toffoli gate is added before the target consecutive controlled RBSs. This Toffoli gate acts on the auxiliary qubit and is controlled by the same qubits that are in the aforementioned **ctrl** set. The next step is the replacement of the set of controls **ctrl** of the target RBS gates by **ctrl** =  $\{n + 1\}$ , where  $n + 1$  is the index of the newly added auxiliary qubit. Finally, the same Toffoli gate is added to the circuit after the stacked RBS gates, undoing the previous Toffoli operation. The same “trick” can be applied to stacked RBS gates that have the same  $k - 2$  indices in **ctrl**, and so on. The process ends when adding the two controlled Toffoli gates is more costly than implementing the original gates.

## APPENDIX D: CLIFFORD DATA REGRESSION

Clifford data regression (CDR) is a data-driven protocol to mitigate errors on expectation values estimated on NISQ devices [31,32]. The protocol to implement CDR takes as inputs a quantum circuit  $\mathcal{C}$  and an expectation value  $\mu^{(0)}$  and repeats the following primitive: (i) construct a near-Clifford circuit by randomly replacing most, if not all, non-Clifford gates in the original circuit with Clifford gates; (ii) run both classical simulation and noisy implementation of the near-Clifford circuit, obtaining a set  $\mathcal{S}$  containing new expectation values  $\mu_{\text{noiseless}}$  and  $\mu_{\text{noisy}}$  coming from the noiseless and noisy circuits, respectively,

$$\mathcal{S} := \left\{ \left( \mu_{\text{noisy}}^{(j)}, \mu_{\text{noiseless}}^{(j)} \right) \right\}_{j \in [|S|]}, \quad (\text{D1})$$

where  $|S|$  is the cardinality of the set  $\mathcal{S}$ . The next step is to fit a regression model  $f$  on  $\mathcal{S}$  such that

$$\mu_{\text{noiseless}}^{(j)} \approx f_{\mathcal{S}} \left( \mu_{\text{noisy}}^{(j)} \right), \quad \forall j \in [|S|]. \quad (\text{D2})$$

With that at hand, after executing the original circuit on the noisy hardware and obtaining the noisy expectation value  $\mu_{\text{noisy}}^{(0)}$ , the last step is to use the fitted model to obtain an error-mitigated version of  $\mu_{\text{noisy}}^{(0)}$ , i.e.,

$$\mu_{\text{mitigated}}^{(0)} \approx f_{\mathcal{S}} \left( \mu_{\text{noisy}}^{(0)} \right). \quad (\text{D3})$$

The error-mitigated expectation value  $\mu_{\text{mitigated}}^{(0)}$  is then the final empirical estimator of  $\mu^{(0)}$ .

In our proof-of-principle demonstration described in Sec. VIA, we implemented CDR following the protocol detailed above, training a linear model for each expectation value separately. In doing so, we first compiled the original circuit in a way that the only non-Clifford gates present were  $R_y$  rotations with at most one control qubit. We define the replacement rate,  $r \in (0, 1)$ , as the fraction of  $R_y$  gates replaced in the original circuit to generate each near-Clifford circuit  $j$ . We created our set of near-Clifford circuits using replacement rates  $r \in \{0.79, 0.82, 0.89, 0.93, 1.00\}$ , sampling 300 circuits per replacement rate, for a total of  $|S| = 1500$  data points. The random Clifford gates to be inserted were sampled uniformly from the set of all possible single- and two-qubit Clifford gates. We also uniformly sampled the positions in the circuit of the  $R_y$  gates that were replaced. The set (D1) was generated by classically simulating noiseless circuits locally while simulating the same circuits using proprietary noise models on IonQ's Quantum Cloud [50]. Every probability estimated from the hardware deployment was treated as an expectation value over a rank-1 projector in the computational basis. The mitigation procedure then

followed as described above, with a separate regression model fitted for each probability.

- 
- [1] G.-L. R. Anselmetti, D. Wierichs, C. Gogolin, and R. M. Parrish, Local, expressive, quantum-number-preserving VQE ansätze for fermionic systems, *New J. Phys.* **23**, 113010 (2021).
  - [2] J. M. Arrazola, O. D. Matteo, N. Quesada, S. Jahangiri, A. Delgado, and N. Killoran, Universal quantum circuits for quantum chemistry, *Quantum* **6**, 742 (2022).
  - [3] L. Monbroussou, E. Z. Mamon, J. Landman, A. B. Grilo, R. Kukla, and E. Kashefi, Trainability and expressivity of Hamming-weight preserving quantum circuits for machine learning, *ArXiv:2309.15547*.
  - [4] E. A. Cherrat, I. Kerenidis, N. Mathur, J. Landman, M. Strahm, and Y. Y. Li, Quantum vision transformers, *Quantum* **8**, 1265 (2024).
  - [5] S. Ramos-Calderer, A. Pérez-Salinas, D. García-Martín, C. Bravo-Prieto, J. Cortada, J. Planagumà, and J. I. Latorre, Quantum unary approach to option pricing, *Phys. Rev. A* **103**, 032414 (2021).
  - [6] S. Johri, S. Debnath, A. Mocherla, A. Singh, A. Prakash, J. Kim, and I. Kerenidis, Nearest centroid classification on a trapped ion quantum computer, *npj Quantum Inf.* **7**, 122 (2021).
  - [7] I. Kerenidis, J. Landman, and N. Mathur, Classical and quantum algorithms for orthogonal neural networks, *ArXiv:2106.07198*.
  - [8] I. Kerenidis and A. Prakash, Quantum machine learning with subspace states, *ArXiv:2202.00054*.
  - [9] H. Zhang, S. Ramos-Calderer, Y. Zhan, W.-K. Mok, H. Cai, F. Gao, X. Luo, G.-Q. Lo, L. C. Kwek, J. I. Latorre, and A. Q. Liu, Efficient option pricing with a unary-based photonic computing chip and generative adversarial learning, *Photon. Res.* **11**, 1703 (2023).
  - [10] D. Ventura and T. Martinez, Initializing the amplitude distribution of a quantum state, *Found. Phys. Lett.* **12**, 547 (1999).
  - [11] G.-L. Long and Y. Sun, Efficient scheme for initializing a quantum register with an arbitrary superposed state, *Phys. Rev. A* **64**, 014303 (2001).
  - [12] L. Grover and T. Rudolph, Creating superpositions that correspond to efficiently integrable probability distributions, *ArXiv:quant-ph/0208112*.
  - [13] S. Lloyd and C. Weedbrook, Quantum generative adversarial learning, *Phys. Rev. Lett.* **121**, 040502 (2018).
  - [14] P.-L. Dallaire-Demers and N. Killoran, Quantum generative adversarial networks, *Phys. Rev. A* **98**, 012324 (2018).
  - [15] K. Mitarai, M. Kitagawa, and K. Fujii, Quantum analog-digital conversion, *Phys. Rev. A* **99**, 012301 (2019).
  - [16] A. Holmes and A. Y. Matsuura, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, 2020), p. 169.
  - [17] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, A divide-and-conquer algorithm for quantum state preparation, *Sci. Rep.* **11**, 6329 (2021).
  - [18] S. Jaques and A. G. Rattew, QRAM: A survey and critique, *ArXiv:2305.10310*.

- [19] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, Elementary gates for quantum computation, *Phys. Rev. A* **52**, 3457 (1995).
- [20] A. Y. Kitaev, Quantum computations: Algorithms and error correction, *Russ. Math. Surv.* **52**, 1191 (1997).
- [21] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, On universal and fault-tolerant quantum computing, *ArXiv:quant-ph/9906054*.
- [22] M. Plesch and Č. Brukner, Quantum-state preparation with universal gate decompositions, *Phys. Rev. A* **83**, 032302 (2011).
- [23] W. J. Huggins, S. McArdle, T. E. O'Brien, J. Lee, N. C. Rubin, S. Boixo, K. B. Whaley, R. Babbush, and J. R. McClean, Virtual distillation for quantum error mitigation, *Phys. Rev. X* **11**, 041036 (2021).
- [24] J. Gibbs, K. Gili, Z. Holmes, B. Commeau, A. Arrasmith, L. Cincio, P. J. Coles, and A. Sornborger, Long-time simulations with high fidelity on quantum hardware, *npj Quantum Inf.* **8**, 135 (2022).
- [25] L. Zhao, J. Goings, K. Wright, J. Nguyen, J. Kim, S. Johri, K. Shin, W. Kyoung, J. I. Fuks, J.-K. K. Rhee, and Y. M. Rhee, Orbital-optimized pair-correlated electron simulations on trapped-ion quantum computers, *npj Quantum Inf.* **9**, 60 (2023).
- [26] N. Jain, J. Landman, N. Mathur, and I. Kerenidis, Quantum Fourier networks for solving parametric PDEs, *Quantum Sci. Technol.* **9**, 035026 (2024).
- [27] Z. He, R. Shaydulin, S. Chakrabarti, D. Herman, C. Li, Y. Sun, and M. Pistoia, Alignment between initial state and mixer improves QAOA performance for constrained optimization, *npj Quantum Inf.* **9**, 121 (2023).
- [28] E. A. Cherratt, S. Raj, I. Kerenidis, A. Shekhar, B. Wood, J. Dee, S. Chakrabarti, R. Chen, D. Herman, S. Hu, *et al.*, Quantum deep hedging, *Quantum* **7**, 1191 (2023).
- [29] M. Ragone, B. N. Bakalov, F. Sauvage, A. F. Kemper, C. O. Marrero, M. Larocca, and M. Cerezo, A Lie algebraic theory of barren plateaus for deep parametrized quantum circuits, *Nat. Commun.* **15**, 7172 (2024).
- [30] IonQ Inc., IonQ Aria, <https://ionq.com/quantum-systems/aria>, 2024.
- [31] P. Czarnik, A. Arrasmith, P. J. Coles, and L. Cincio, Error mitigation with Clifford quantum-circuit data, *Quantum* **5**, 592 (2021).
- [32] A. Lowe, M. H. Gordon, P. Czarnik, A. Arrasmith, P. J. Coles, and L. Cincio, Unified approach to data-driven quantum error mitigation, *Phys. Rev. Res.* **3**, 033098 (2021).
- [33] The gate  $R_{\text{out}}^{\text{in}}(-\theta, 0)$  is preferred by some authors and referred to as a Givens rotation.
- [34] S. Even, *Algorithmic Combinatorics* (The Macmillan Company, 1973).
- [35] J. Landman, N. Mathur, Y. Y. Li, M. Strahm, S. Kazdaghi, A. Prakash, and I. Kerenidis, Quantum methods for neural networks and application to medical image classification, *Quantum* **6**, 881 (2022).
- [36] T. M. L. de Veras, L. D. da Silva, and A. J. da Silva, Double sparse quantum state preparation, *Quantum Inf. Process.* **21**, 204 (2022).
- [37] I. F. Araujo, I. C. S. Araújo, L. D. da Silva, C. Blank, and A. J. da Silva, Quantum computing library, <https://github.com/qc/lib/qc/lib>, 2023.
- [38] V. Shende, S. Bullock, and I. Markov, Synthesis of quantum-logic circuits, *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.* **25**, 1000 (2006).
- [39] 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, *et al.*, Quantum computing with Qiskit, *ArXiv:2405.08810*.
- [40] C. Tsallis, Possible generalization of Boltzmann-Gibbs statistics, *J. Stat. Phys.* **52**, 479 (1988).
- [41] C. Tsallis, S. V. F. Levy, A. M. C. Souza, and R. Maynard, Statistical-mechanical foundation of the ubiquity of Lévy distributions in nature, *Phys. Rev. Lett.* **75**, 3589 (1995).
- [42] D. Prato and C. Tsallis, Nonextensive foundation of Lévy distributions, *Phys. Rev. E* **60**, 2398 (1999).
- [43] C. Tsallis, Nonadditive entropy and nonextensive statistical mechanics—An overview after 20 years, *Braz. J. Phys.* **39**, 337 (2009).
- [44] L. Borland, Option pricing formulas based on a non-Gaussian stock price model, *Phys. Rev. Lett.* **89**, 098701 (2002).
- [45] M. Gell-Mann and C. Tsallis, in *Nonextensive Entropy: Interdisciplinary Applications* (Oxford University Press, 2004).
- [46] V. Witkovský, Characteristic function of the Tsallis  $q$ -Gaussian and its applications in measurement and metrology, *Metrology* **3**, 222 (2023).
- [47] F. Fernández-Navarro, C. Hervás-Martínez, M. Cruz-Ramírez, P. A. Gutiérrez, and A. Valero, Evolutionary  $q$ -Gaussian Radial Basis Function Neural Network to determine the microbial growth/no growth interface of *Staphylococcus aureus*, *Appl. Soft. Comput.* **11**, 3012 (2011).
- [48] J. Helsen, I. Roth, E. Onorati, A. Werner, and J. Eisert, General framework for randomized benchmarking, *PRX Quantum* **3**, 020357 (2022).
- [49] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. García-Saez, J. I. Latorre, and S. Carrazza, Qibo: A framework for quantum simulation with hardware acceleration, *Quantum Sci. Technol.* **7**, 015018 (2021).
- [50] IonQ Inc., IonQ quantum cloud, <https://ionq.com/quantum-cloud>, 2024.
- [51] IBM Inc., IBM Quantum, <https://quantum-computing.ibm.com>, 2025.
- [52] Quantinuum, Quantinuum H series, [https://docs.quantinuum.com/h-series/user\\_guide/emulator\\_user\\_guide/noise\\_model.html](https://docs.quantinuum.com/h-series/user_guide/emulator_user_guide/noise_model.html), 2025.
- [53] G. Q. AI and Collaborators, Quantum error correction below the surface code threshold, *Nature* **638**, 920 (2024).
- [54] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. Akash-Narayanan, A. Asadi, *et al.*, PennyLane: Automatic differentiation of hybrid quantum-classical computations, *ArXiv:1811.04968*.
- [55] U. Azad and S. Fomichev, PennyLane quantum chemistry datasets, <https://pennylane.ai/datasets/h4-molecule>, 2023.
- [56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, PyTorch: An imperative style, high-performance deep learning library, *ArXiv:1912.01703*.
- [57] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, *ArXiv:1412.6980*.

- [58] S. Ramos-Calderer, Efficient quantum interpolation of natural data, *Phys. Rev. A* **106**, 062427 (2022).
- [59] M. Sciorilli, L. Borges, T. L. Patti, D. García-Martín, G. Camilo, A. Anandkumar, and L. Aolita, Towards large-scale quantum optimization solvers with few qubits, *Nat. Commun.* **16**, 476 (2025).
- [60] D. Raveh and R. I. Nepomechie, Deterministic Bethe state preparation, *Quantum* **8**, 1510 (2024).
- [61] J. Wilkens, Quantum Circuit Library, <https://github.com/wilkensJ/drawio-library>, 2023.
- [62] R. Vale, T. M. D. Azevedo, I. C. S. Araújo, I. F. Araujo, and A. J. da Silva, Decomposition of multi-controlled special unitary single-qubit gates, *ArXiv:2302.06377*.
- [63] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, Quantum circuits for isometries, *Phys. Rev. A* **93**, 032318 (2016).