

Low-Depth Quantum Symmetrization

Zhenning Liu[✉], Andrew M. Childs[✉], and Daniel Gottesman[✉]

*Department of Computer Science, Institute for Advanced Computer Studies,
and Joint Center for Quantum Information and Computer Science, University of Maryland,
College Park, Maryland 20742, USA*



(Received 2 May 2025; revised 11 March 2026; accepted 20 July 2026; published 31 August 2026)

Quantum symmetrization is the task of transforming a nonstrictly increasing list of n integers into an equal superposition of all permutations of the list (or more generally, performing this operation coherently on a superposition of such lists). This task plays a key role in initial state preparation for first-quantized simulations. Motivated by an application to fermionic systems, various algorithms have been proposed to solve a weaker version of symmetrization in which the input list is *strictly* increasing, but the general symmetrization problem with repetitions in the input list has not been well studied. We present the first efficient quantum algorithms for the general symmetrization problem. If m is the greatest possible value of the input list, our first algorithm symmetrizes any single classical input list using $\tilde{O}(\log n)$ depth and $O(n \log n + \log m)$ ancilla qubits, and our second algorithm symmetrizes an arbitrary superposition of input lists using $\tilde{O}(\log^3 n)$ depth and $O(n \log n)$ ancilla qubits. Our algorithms enable efficient simulation of bosonic quantum systems in first quantization (in particular, we describe a simulation of the Bose-Hubbard model with complexity $\text{poly}(n, \log m)$) and can prepare (superpositions of) Dicke states of any Hamming weight in $\tilde{O}(\log n)$ depth (respectively, $\tilde{O}(\log^3 n)$ depth) using $O(n \log n)$ ancilla qubits. We also propose an $\tilde{O}(\log^3 n)$ -depth quantum algorithm to transform second-quantized states to first-quantized states. Using this algorithm, QFT-based quantum telescope arrays can image brighter photon sources, extending quantum interferometric imaging systems to a new regime.

DOI: [10.1103/9qhy-ms2y](https://doi.org/10.1103/9qhy-ms2y)

Subject Areas: Astrophysics, Quantum Physics,
Quantum Information

I. INTRODUCTION

Symmetry is a central concept in physics. In particular, symmetry plays a key role in quantum simulation [1], where a faithful simulation of the many-body dynamics of indistinguishable particles should respect the symmetry of the simulated system. For fermions, due to the Pauli exclusion principle, no two particles can occupy the same mode, and the wave function must be antisymmetrized with respect to permutations of particles, leading to Fermi-Dirac statistics. For bosons, multiple particles can be in the same state, and the wave function must be symmetrized with respect to permutations, leading to Bose-Einstein statistics. A general quantum simulation procedure contains not only Hamiltonian evolution and final output, but also initial state preparation, which is often a challenging step, especially when the initial wave function has some specific structure or symmetry.

In this paper, we study a simple but subtle problem in quantum computing: transforming a list l of nonstrictly increasing integers stored in a quantum register into the equal superposition of all permutations of l . We call this “symmetrization,” since the output state is symmetric with respect to any permutation of l . Symmetrization plays a fundamental role in quantum simulation based on first quantization, an encoding of identical particles as a list of particle locations (whereas second quantization represents the state as a list of occupation numbers of particle locations). Eigenstates in first quantization must be symmetric or antisymmetric if the system is bosonic or fermionic, respectively. Because of significant interest in quantum chemistry (which involves interactions between fermions), a weaker version of (anti)symmetrization where the list is strictly increasing has been extensively studied [2–4]. In particular, work of Berry *et al.* [4] develops a logarithmic-depth algorithm. However, first-quantized simulation of bosons must allow multiple particles to be in the same mode, so the input list may contain repeated integers. Although several previous papers have claimed to handle this more general case [2,3,5–8], close examination reveals a subtlety that has not been properly addressed in the literature. The only solution for the general symmetrization

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

problem that we are aware of is a result of Nepomechie and Raveh [9]. However, this algorithm requires prohibitively deep circuits: the depth is $O(n^m)$, where n is the number of integers and m is the greatest of these integers.

As the main results of this paper, we propose new logarithmic-depth quantum algorithms for symmetrization and a polylogarithmic-depth quantum algorithm for transforming from second to first quantization. We also present three applications of these algorithms. The first application, mentioned above, is initial state preparation in first-quantized simulation of bosons, an unrecognized unsolved problem before our work. The second application is preparing a specific class of symmetric states useful in a variety of quantum information processing tasks, Dicke states, where our algorithm is the first [10] to achieve logarithmic total circuit depth using a reasonable number of ancilla qubits. The third application is in optics, where we use our algorithm to improve strategies for long-baseline quantum telescope arrays [12,13] to process multiple photons at a time, yielding a quantum interferometric imaging system that can take images in more general scenarios than previous designs.

In the remainder of this introductory section, we define the symmetrization problem formally in Sec. I A. Then, we briefly state our results in Sec. I B. Finally, in Sec. I C, we introduce the three applications of symmetrization in more detail.

We present background for our results in Sec. II. More specifically, in Sec. II A, we introduce sorting networks and their quantization as the building blocks of symmetrization. Then in Sec. II B, we describe the algorithm of Berry *et al.* [4], which is used as a subroutine of one of our symmetrization algorithms. In Sec. II C, we generalize sorting to different comparison rules and define reversible sorting networks. In Sec. II D, we present the quantization of a standard prefix sum computation procedure.

In Sec. III, we discuss the general symmetrization problem with repetitions. We first explain in Sec. III A why previous methods [2–4] are not applicable to the more general symmetrization problem by discussing the subgroup structure of permutations when considering repetitions in the permuted list, and then extend quantum sorting networks to work with repetitions in Sec. III B. Using these theoretical tools, we present our new algorithm for symmetrizing a single input list in Sec. III C.

We present our symmetrization algorithm for an arbitrary superposition of input lists in Sec. IV. We present a novel strictly increasing list symmetrization procedure in Sec. IV A based on quantization of a classical parallel algorithm from Alonso and Schott [14], which is subsequently used as a subroutine in Sec. IV B to design the nonstrictly increasing list symmetrization algorithm for superposed inputs.

In Sec. V, we propose a polylogarithmic-depth converter from a second quantized representation to a sorted list of

modes occupied by the particles. This binary encoder, together with the symmetrization algorithm, gives the first polylogarithmic-depth conversion from second quantization to first quantization.

In Sec. VI, we carry out a case study of first-quantized simulation algorithms applied to Bose-Hubbard models with n particles and m modes. We present a $\text{poly}(\log n, \log m)$ -depth simulation algorithm for the hopping term in Sec. VI B, and introduce a quantum signal processing-based algorithm that simulates the evolution of the whole Hamiltonian using $\text{poly}(n, \log m)$ gates in Sec. VI C.

As a topic of independent interest, we demonstrate our design of a quantum interferometric imaging system in Sec. VII, including a pedagogical introduction of quantum telescope arrays that process one photon a time [12,13] in Sec. VII A, followed by our extension to multiple-photon processing (using the conversion from second to first quantization) in Sec. VII B.

Finally, we summarize our results and discuss potential future works in Sec. VIII.

A. Problem setup

We design an $\tilde{O}(\log n)$ -depth quantum algorithm for symmetrizing general increasing lists that may include repeated elements, which we call nonstrictly increasing lists (NSILs). Here we define $\tilde{O}(f(n, m)) := O(f(n, m) \log \log \max(n, m))$.

Recall that this symmetrization problem has been solved for strictly increasing lists (SILs), which do not have any repetition. We formally define such lists as follows.

Definition 1. (NSIL_{*n*} and SIL_{*n*}) For any non-negative integer n ,

$$\begin{aligned} \text{NSIL}_n &:= \{l \in \mathbb{Z}^n \mid \forall i < j, l_i \leq l_j\} \\ \text{SIL}_n &:= \{l \in \mathbb{Z}^n \mid \forall i < j, l_i < l_j\} \end{aligned} \quad (1)$$

are the sets of length- n NSILs and SILs, respectively.

If the l_i s are restricted to be non-negative integers upper bounded by $m \in \mathbb{N}$, then we have

$$\begin{aligned} \text{NSIL}_n^m &:= \{l \in \{0, 1, \dots, m\}^n \mid \forall i < j, l_i \leq l_j\}, \\ \text{SIL}_n^m &:= \{l \in \{0, 1, \dots, m\}^n \mid \forall i < j, l_i < l_j\}. \end{aligned} \quad (2)$$

The input state for symmetrization is either $|l\rangle$ where l is an NSIL of size n , or more generally, a superposition of such lists, i.e.,

$$\sum_{l \in \text{NSIL}_n} \alpha_l |l\rangle, \quad (3)$$

where $\sum_l |\alpha_l|^2 = 1$.

The output state of symmetrization for $|l\rangle$ is

$$|\psi_{\text{out}}(l)\rangle \propto \sum_{\sigma \in S_n} |\sigma(l)\rangle, \quad (4)$$

where S_n is the group of permutations of n elements. Similarly, for superposed inputs, the output state is proportional to

$$\sum_{l \in \text{NSIL}_n} \alpha_l |\psi_{\text{out}}(l)\rangle. \quad (5)$$

B. Our results

We provide two NSIL symmetrization algorithms, one for single NSIL inputs and another for superposed NSIL inputs. Both algorithms use SIL symmetrization algorithms as subroutines.

The first result uses a modified version of the Berry *et al.* [4] algorithm and is designed for a single NSIL input. It takes $\tilde{O}(\log n)$ depth.

Result 1. [$\tilde{O}(\log n)$ -depth single-input NSIL symmetrization, Sec. III C] For all $a \geq 2$, there exists an $\tilde{O}(\log n)$ -depth algorithm using $O(an \log n + \log m)$ ancilla qubits that produces $|\psi_{\text{out}}(l)\rangle$ for any $l \in \text{NSIL}_n$ with success probability $1 - O(n^{-a+2})$.

Our second algorithm has slightly greater depth ($\tilde{O}(\log^3 n)$) but works for arbitrary superpositions of inputs $\sum_{l \in \text{NSIL}_n} \alpha_l |l\rangle$ and always succeeds. It uses a novel SIL symmetrization algorithm as a subroutine, which is based on quantization of a classical parallel algorithm for generating random permutations [14].

Result 2. [$\tilde{O}(\log^3 n)$ -depth superposed-input NSIL symmetrization, Sec. IV] There exists an $\tilde{O}(\log^3 n)$ -depth NSIL symmetrization algorithm using $O(n \log n)$ ancilla qubits that produces $\sum_{l \in \text{NSIL}_n} \alpha_l |\psi_{\text{out}}(l)\rangle$ for any input state $\sum_{l \in \text{NSIL}_n} \alpha_l |l\rangle$ with success probability 1.

Note that the number of ancilla qubits used in the algorithms of Result 1 and 2 does not depend on m , just as in the Berry *et al.* [4] algorithm.

We also propose an $\tilde{O}(\log^2 n)$ -depth algorithm to convert a list of occupation numbers to an NSIL of the occupied modes. Combining this algorithm and Result 2, we introduce the first polylogarithmic-depth quantum algorithm to transform a second-quantized state to a first-quantized state. Since this algorithm is a unitary composed of elementary quantum gates, one can also transform from a first-quantized state to a second-quantized state by applying the inverse gates in the reverse direction.

Result 3. [$\tilde{O}(\log^3 n)$ -depth conversion between second quantization and first quantization, Sec. V] There exists an $\tilde{O}(\log^3 n)$ -depth algorithm using $O(n \log n)$ ancilla qubits that transforms a state of n particles in m modes from a second-quantized representation to a first-quantized

representation, or from a first-quantized representation to a second-quantized representation.

C. Applications

1. Bosonic eigenstate preparation

For systems of n indistinguishable particles and m possible modes, second quantization uses $O(m \log n)$ qubits, while first quantization uses $O(n \log m)$ qubits, which is more favorable when $m \gg n$. However, for first quantization, the system symmetry must be reflected by the state, i.e., if two indistinguishable particles are exchanged, the state must be the same but with a ± 1 phase depending on the particle type. Therefore, first-quantized eigenstates must be symmetrized or antisymmetrized for bosons or fermions, respectively. Unlike for fermions, initial state preparation for first-quantized bosonic systems has not been extensively studied, even though it is a fundamental simulation primitive.

We review the history of initial state preparation for first-quantized bosonic systems. Abrams and Lloyd [2] mainly considered Fermi systems and described a poly(n)-depth symmetrization algorithm for SILs, which is improved by Ward *et al.* [3]. Although both papers claimed to be applicable for bosons, we explain why this is not the case in Sections III A and III B. More recently, Berry *et al.* [4] proposed the first $\tilde{O}(\log n)$ -depth SIL symmetrization algorithm, as discussed in Sections II B and III B. Several papers proposed using these three algorithms to simulate bosons, without noticing the subtlety caused by repeated items. This includes the recent work of Su *et al.* [5] which analyzed resource requirements for quantum chemistry simulation in first quantization and claimed that their algorithms also apply to bosons. Furthermore, Chan *et al.* [6] and Georges *et al.* [8] discussed quantum chemistry simulation and suggested that initial state preparation for bosons can be done in a similar way as for fermions.

There are also several other recent theoretical developments in first-quantized simulation of bosons that do not discuss initial state preparation, including Berry *et al.* [15], which discussed simulation of materials using the strategy in Su *et al.* [5]; and Mukhopadhyay *et al.* [16], which studies simulation of Pauli-Fierz Hamiltonians. Similarly, Tong *et al.* [17] discussed simulation of gauge theories and bosonic systems, but the authors did not consider the computational cost of the initial state preparation.

As the most straightforward application of Results 1 and 2, the NSIL symmetrization algorithms enable efficient initial state preparation for first-quantized bosonic simulation. Furthermore, as mentioned in [3], one can also prepare a first-quantized initial state by preparing a second-quantized initial state and transforming it to its first-quantized counterpart. Our Result 3 implements this transformation in polylogarithmic depth. This idea is

particularly useful if the second-quantized state is easier to prepare than the superposition of NSILs.

However, it is worth noting that there are fewer proposals for first-quantized simulation of bosons than of fermions, partially because the advantage of first quantization has not been well understood. In particular, apart from the aforementioned reduction in the qubit number, it was unknown whether the gate count could also be reduced in the $m \gg n$ scenario. In Sec. VI, as an example to showcase the advantage of first quantization, we introduce a first-quantized algorithm, based on quantum signal processing and our symmetrization methods, to simulate the dynamics of the Bose-Hubbard model using $\text{poly}(n, \log m)$ gates. In contrast, state-of-the-art second-quantized algorithms use $\text{poly}(n, m)$ gates [18], i.e., they do not achieve logarithmic scaling with the number of particles. In particular, the best known algorithms use $\text{poly}(n)$ gates even to simulate the two-mode hopping term in second quantization. Our algorithm provides a key subroutine for first-quantized simulation that could potentially enable faster simulations of more general Bose-Hubbard-type models [18] or pionful nuclear effective theory [19].

2. Dicke state preparation

The Dicke state $|D_n^k\rangle$ [20] is the equal superposition of all n -qubit computational-basis states with Hamming weight k , i.e.,

$$|D_n^k\rangle := \frac{1}{\sqrt{\binom{n}{k}}} \sum_{r \in \{0,1\}^n, |r|=k} |r\rangle. \quad (6)$$

Dicke states play an important role in various areas of quantum information science, including quantum networks [21], quantum metrology [22], quantum error correction [23], and the quantum approximate optimization algorithm [24]. Therefore, the quantum information community has sought circuits for preparing Dicke states with lower depth, gate complexity, and number of ancilla qubits. The most challenging Dicke states to prepare are those with

k close to $n/2$. Linear-depth circuits for preparing these Dicke states were proposed in Refs. [25,26], but the first $\text{poly}(\log n)$ -depth circuits were proposed by Buhrman *et al.* [27]. These circuits use a model of “local alternating quantum classical computations” in which the quantum operations performed are determined by the measurement outcomes of the previous step. However, the number of ancilla qubits used in that algorithm is huge [$\Omega(n^2)$] for large n . Piroli *et al.* [28] significantly reduce the number of ancilla qubits and circuit depth to $O(n \log n)$ and $O(1)$, respectively, or $O(n)$ and $O(\log n)$, respectively. However, the circuit in Ref. [28] needs to be repeated $O(\sqrt{n})$ times in total, which implies either a larger number of ancilla qubits $O(n^{3/2} \cdot \text{poly}(\log n))$ or polynomial total circuit depth.

Our low-depth NSIL symmetrization algorithm gives a straightforward strategy for preparing arbitrary Dicke states with $\tilde{O}(\log n)$ depth and $O(n \log n)$ ancilla qubits, with the cost of increased circuit connectivity. To prepare $|D_n^k\rangle$, it suffices to symmetrize

$$|\psi_{n,k}\rangle := |0\rangle^{\otimes n-k} \otimes |1\rangle^{\otimes k} \quad (7)$$

using our algorithm in Sec. III C, since the initial state is only one NSIL.

Superpositions of different Dicke states (with different k values) are also useful in quantum metrology. For example, Lin *et al.* [29] show that metrological entanglement advantage can be achieved with such states. We can prepare such states in $\tilde{O}(\log^3 n)$ depth by first initializing the state as a superposition of different $|\psi_{n,k}\rangle$ states and then applying the full NSIL symmetrization algorithm described in Sec. IV. While some Dicke state preparation circuits can straightforwardly prepare superpositions of Dicke states [25,26], this is not the case for previous polylogarithmic-depth circuits [27,28], which may need to be modified and might incur extra overhead.

We summarize the comparisons of our algorithm with a selection of other Dicke state preparation circuits in Table I.

TABLE I. Comparison of Dicke state preparation algorithms for Hamming weight $k = O(n)$. Note that the algorithm of Piroli *et al.* [28] must be repeated $O(\sqrt{n})$ times, which implies either a larger number of ancilla qubits ($O(n^{3/2} \cdot \text{poly}(\log n))$) or polynomial total circuit depth.

Name	Depth	Repetitions	Ancillas	Connectivity
Bäertschi and Eidenbenz [25]	$O(n)$	1	0	LNN
Bäertschi and Eidenbenz [26]	$O(n)$	1	0	LNN and all-to-all
Buhrman <i>et al.</i> [27]	$O(\log n)$	1	$\Omega(n^2)$	LNN
Piroli <i>et al.</i> [28]	$O(1)$	$O(\sqrt{n})$	$O(n \log n)$	LNN
Piroli <i>et al.</i> [28]	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	LNN
Result 1	$\tilde{O}(\log n)$	$O(1)$	$O(n \log n)$	All-to-all
Result 2	$\tilde{O}(\log^3 n)$	1	$O(n \log n)$	All-to-all

3. Quantum interferometric imaging

As a promising candidate to achieve microarcsecond resolution in astronomical observation, long-baseline telescopes based on quantum repeaters were first proposed in Gottesman *et al.* [12] and have sparked increasing research interest in both the astronomy and quantum information communities. In follow-up work, Khabiboulline *et al.* [13] design a *quantum telescope array* and employ the quantum Fourier transform (QFT) to improve the signal-to-noise ratio. However, their method can only handle the case where the number of photons per mode is restricted to be much less than 1 (i.e., the object must be extremely dim). Using our NSIL symmetrization algorithm, we extend the QFT-based scheme to cases where some modes have more than 1 photon. Since symmetrization is an intermediate step of interferometric imaging, we postpone the detailed description of this application to Sec. VII.

II. PRELIMINARIES

A. (Quantum) sorting networks for SILs

A sorting network is a circuit of comparators that can be used to implement parallel comparison-based sorting algorithms. A classical comparator between elements a_i and a_j with $i < j$ swaps the values of a_i and a_j if $a_i > a_j$, and otherwise leaves them unchanged. For instance, a sorting network for 3 elements, corresponding to the well-known bubble sort algorithm, is shown in Fig. 1. Comparators can be quantized by adding an ancilla bit, as shown in Fig. 2. This construction implements a unitary comparator U_{comp} acting as

$$U_{\text{comp}}|a, b, 0\rangle = |\min(a, b), \max(a, b), \delta[a > b]\rangle, \quad (8)$$

where $\delta[a > b]$ is 1 if $a > b$ and 0 otherwise.

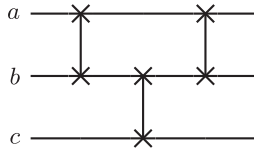


FIG. 1. Bubble sort implemented by classical comparators.

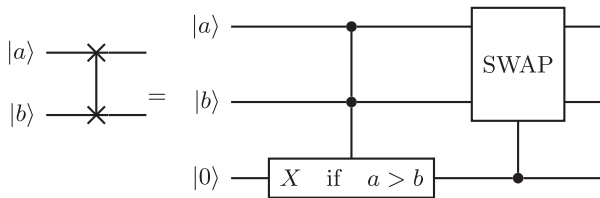


FIG. 2. A quantum comparator consists of a comparison operator and a controlled-SWAP gate. An ancilla qubit is used to store the result of comparison and ensure the reversibility of the operation.

Note that if the integer is in $[m]$, then each a or b can be represented with $\log m$ bits. The comparator unitary for $\log m$ qubits can be implemented in $O(\log \log m)$ depth, as shown in Appendix B. 2 of Ref. [4]. When analyzing the circuit depth, we ignore this double-logarithmic overhead for simplicity.

Using the unitary comparator (8), any sorting algorithm based on a sorting network of N_a comparators can be directly quantized with N_a ancilla qubits. Each ancilla qubit stores the result of the comparison of its corresponding comparator. In this paper, we combine all such ancilla qubits in a single register, and call its state after performing the sorting algorithm a record. For $l \in \text{SIL}_n$ and $\sigma \in S_n$, the record obtained when sorting $\sigma(l)$ is denoted $\text{rec}(\sigma)$. Note that $\text{rec}(\sigma)$ is independent of l since for any $l, l' \in \text{SIL}_n$, the record of sorting $\sigma(l)$ is the same as the record of sorting $\sigma(l')$. We write $\text{SORT}_{i,j}$ to represent the quantized sorting network that sorts a permuted SIL in register i and stores the record in register j . Its operation is simply

$$\text{SORT}_{i,j}|\sigma(l)\rangle_i|0\rangle_j = |l\rangle_i|\text{rec}(\sigma)\rangle_j. \quad (9)$$

As this is a linear operation, it can also be applied to a superposition of input states.

Apart from $\text{SORT}_{i,j}$, we make use of several other quantum operations based on quantum sorting networks. By running all gates of SORT (which are either comparisons or CSWAPS) in the opposite direction, one can implement $\text{UNSORT}_{i,j}$, the inverse of $\text{SORT}_{i,j}$, which acts as

$$\text{UNSORT}_{i,j}|l\rangle_i|\text{rec}(\sigma)\rangle_j = |\sigma(l)\rangle_i|0\rangle_j. \quad (10)$$

Furthermore, if we remove all comparison gates and only keep the CSWAP gates in UNSORT, we obtain an operation denoted $\text{SHUFFLE}_{i,j}$, which applies the permutation stored in $\text{rec}(\tau)$ to the input list:

$$\text{SHUFFLE}_{i,j}|\sigma(l)\rangle_i|\text{rec}(\tau)\rangle_j = |\tau \circ \sigma(l)\rangle_i|\text{rec}(\tau)\rangle_j. \quad (11)$$

Here the input list is not necessarily an SIL, so it is written as $\sigma(l)$. Since no comparison is performed in SHUFFLE, it has no dependence on the comparison rule.

Notice that SORT essentially applies σ^{-1} to $\sigma(l)$ to obtain l using the information in the record generated by the comparison gates. If a record of τ is given and only the CSWAP gates in SORT are performed, we get UNSHUFFLE, the inverse operation of SHUFFLE:

$$\text{UNSHUFFLE}_{i,j}|\sigma(l)\rangle_i|\text{rec}(\tau)\rangle_j = |\tau^{-1} \circ \sigma(l)\rangle_i|\text{rec}(\tau)\rangle_j. \quad (12)$$

Multiple sorting networks with sub-linear depth have been proposed, including the well-known bitonic sort [30] and merge sort [31], both featuring $O(\log^2 n)$ depth for a

list of n elements [and $N_a = O(n \log^2 n)$ ancilla qubits when quantized]. As far as we are aware, the only known $O(\log n)$ -depth sorting network is the AKS algorithm proposed in Ref. [32]. However, this algorithm is impractical due to its large constant factor.

In this paper, we employ the AKS algorithm to design our symmetrization algorithms, but we emphasize that all sorting networks we use can be replaced by bitonic or merge sort if the number of elements to be sorted is not too large. While the asymptotic depth using bitonic or merge sort will have another factor of $\log n$, the performance may be better in practice. Since all four operations (SORT, UNSORT, SHUFFLE, UNSHUFFLE) only use as many gates as running a quantized sorting network, they can all be performed in $\tilde{O}(\log n)$ depth.

B. Berry *et al.* SIL symmetrization algorithms

Symmetrization and antisymmetrization algorithms are fundamental building blocks of first-quantized quantum simulation for bosonic and fermionic systems. Abrams and Lloyd [2] proposed the first quantum (anti-)symmetrization algorithm. This procedure has $\text{poly}(n)$ depth. Berry *et al.* [4] improved their algorithm by introducing $\log(n)$ -depth sorting networks and a $\log(n)$ -depth method for generating random permutations. However, both algorithms are only applicable to SIL inputs, although this was not explicitly pointed out in their papers. In this subsection, we briefly introduce the algorithm of Berry *et al.* [4], as it is a building block of our NSIL symmetrization algorithm.

Let $l \in \text{SIL}_n^m$ be the input list. The algorithm uses 3 registers, and begins with the state

$$|l\rangle_1 |0\dots 0\rangle_2 |0\dots 0\rangle_3. \quad (13)$$

Register 2 is designed to hold a list of n non-negative integers, where each integer is upper bounded by a preselected number $f(n) \gg n^2$. Therefore register 2 consists of $O(n \log n)$ qubits. Register 3 is the record register and also has $O(n \log n)$ qubits.

The algorithm starts by transforming the state in register 2 from $|00\dots 0\rangle$ into the product state

$$\begin{aligned} |\phi_2\rangle &\propto \sum_{r_1=0}^{f(n)} |r_1\rangle \otimes \sum_{r_2=0}^{f(n)} |r_2\rangle \otimes \dots \otimes \sum_{r_n=0}^{f(n)} |r_n\rangle \\ &= \sum_{r_i \in [f(n)] \forall i \in [n]} |r_1 r_2 \dots r_n\rangle. \end{aligned} \quad (14)$$

Write $r := (r_1, r_2, \dots, r_n)$. Note that we can divide the rs into 2 categories:

- (i) r is nonrepetitive if $r_i \neq r_j \forall i \neq j$;
 - (ii) r is repetitive if $\exists i, j \in [n], i \neq j$ such that $r_i = r_j$.
- The probability of r being repetitive is at most $O(n^{-a+2})$ if each r_i is uniformly randomly sampled from $[f(n)]$ and

$f(n) \gg n^a$. Hence if we choose $a \geq 3$, then the probability of r being repetitive is negligible. Moreover, if r is non-repetitive, then there must be $\sigma \in S_n$ and $l' \in \text{SIL}_n$ such that $r = \sigma(l')$. Therefore, we can rewrite $|\phi_2\rangle$ as

$$|\phi_2\rangle \propto \sum_{l' \in \text{SIL}_n^m} \sum_{\sigma \in S_n} |\sigma(l')\rangle + |\perp\rangle, \quad (15)$$

where $|\perp\rangle$ is the superposition of repetitive rs . Note that Berry *et al.* [4] include a measure-and-postselect step to remove the repetitive part. We can also choose to use $a \geq 3$ and keep the repetitive part since it only leads to a negligible error.

Next, sort register 2 using the $\tilde{O}(\log n)$ -depth sorting network and store the record in register 3:

$$\begin{aligned} &|l\rangle_1 |\phi_2\rangle_2 |0\dots 0\rangle_3 \\ &\xrightarrow{\text{SORT}_{2,3}} |l\rangle_1 \left(\sum_{l' \in \text{SIL}_n^m} |l'\rangle_2 \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_3 + |\perp\rangle_{23} \right). \end{aligned} \quad (16)$$

Now the 3 registers are almost independent. If register 2 is discarded, then with high probability $(1 - O(n^{-a+2}))$ the remaining part is $|l\rangle_1 \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_3$ and the desired symmetrized state can be produced using UNSORT:

$$|l\rangle_1 \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_3 \xrightarrow{\text{UNSORT}_{1,3}} \sum_{\sigma \in S_n} |\sigma(l)\rangle_1 |00\dots 0\rangle_3. \quad (17)$$

In fact, the last step of the Abrams and Lloyd [2] algorithm is also unsorting the input list using the superposition of all records in S_n , but the UNSORT operation is implemented based on $\text{poly}(n)$ -depth sorting algorithms.

C. (Quantum) reversible sorting

Even though sorting is an irreversible operation, we can define a notion of “reversible sorting” as a transformation between lists that are promised to be sorted in different ways. First, we define a comparison rule Γ as a function

$$\Gamma: \mathbb{R}^k \times \mathbb{R}^k \rightarrow \{-1, 0, 1\}, \quad (18)$$

satisfying

$$\Gamma(x, y) = 0 \quad \text{iff } x = y \quad (19)$$

and

$$\Gamma(x, y) = 1 \quad \text{AND} \quad \Gamma(y, z) = 1 \Rightarrow \Gamma(x, z) = 1, \quad (20)$$

where $k \in \mathbb{Z}^+$. Every comparison rule defines a total order of numbers (or tuples of numbers). For instance, the ascending order of real numbers correspond to a comparison rule defined as $\Gamma(x, y) = 1$ iff $x < y$ and $\Gamma(x, y) = -1$

iff $x > y$. We can now define Γ -strictly increasing lists (or Γ -sorted lists) as follows.

Definition 2. [Γ -strictly increasing lists (Γ SIL)] Suppose l is a list of tuples of k real numbers and Γ is a comparison rule of k tuples. Then l is a Γ SIL if and only if $\Gamma(l_i, l_j) = 1$ for all $i < j$.

We can also define comparison operators $<_\Gamma, =_\Gamma, >_\Gamma$ using this comparison rule:

$$\begin{aligned} x <_\Gamma y & \text{ iff } \Gamma(x, y) = 1, \\ x =_\Gamma y & \text{ iff } \Gamma(x, y) = 0, \\ x >_\Gamma y & \text{ iff } \Gamma(x, y) = -1. \end{aligned} \quad (21)$$

A Γ -sorting algorithm, denoted by $\mathcal{A}_\Gamma$, is a classical algorithm transforming a list to a Γ -sorted list. Such algorithms can be constructed for any Γ using any classical comparison-based algorithm by replacing the usual $<$ operator with $<_\Gamma$. Now, suppose Γ_1, Γ_2 are comparison rules, and a list l is Γ_1 sorted. Then sorting l using $\mathcal{A}_{\Gamma_2}$ leads to a Γ_2 -sorted list, denoted by $\mathcal{A}_{\Gamma_2}(l)$. This list can be sorted back to l using $\mathcal{A}_{\Gamma_1}$, i.e., $\mathcal{A}_{\Gamma_1}(\mathcal{A}_{\Gamma_2}(l)) = l$. Since the sorting process can be inverted if both orders are known, we call this process reversible sorting.

An example of reversible sorting is as follows. Consider two comparison operators $<_{\Gamma_1}, <_{\Gamma_2}$ for integers:

$$x <_{\Gamma_1} y \text{ iff } x < y \quad (22)$$

and

$$\begin{aligned} x <_{\Gamma_2} y & \text{ iff } [\text{par}(x) = 1 \text{ AND } \text{par}(y) = 0] \\ & \text{ OR } [\text{par}(x) = \text{par}(y) \text{ AND } x < y], \end{aligned} \quad (23)$$

where $\text{par}(x)$ is the parity of x . Letting $l_1 = (1, 2, 3, 4, 5, 7, 8)$ and $l_2 = (1, 3, 5, 7, 2, 4, 8)$, one can observe that l_1 is Γ_1 sorted, l_2 is Γ_2 sorted, and we can transform between them using Γ_1 - and Γ_2 -sorting algorithms, respectively:

$$\begin{aligned} \mathcal{A}_{\Gamma_1}(1, 3, 5, 7, 2, 4, 8) &= (1, 2, 3, 4, 5, 7, 8), \\ \mathcal{A}_{\Gamma_2}(1, 2, 3, 4, 5, 7, 8) &= (1, 3, 5, 7, 2, 4, 8). \end{aligned} \quad (24)$$

Observe that comparators can be generalized for any arbitrary comparison rule Γ : the unitary comparator U_{comp}^Γ acts as

$$U_{\text{comp}}^\Gamma |a, b, 0\rangle = |\min(a, b), \max(a, b), \delta[a >_\Gamma b]\rangle, \quad (25)$$

where $\delta[a >_\Gamma b]$ is 1 if $a >_\Gamma b$ and 0 otherwise. Replacing U_{comp} by U_{comp}^Γ , one can also implement SORT^Γ and UNSORT^Γ which sort and unsort tuple lists with respect to Γ , respectively.

Reversible sorting can be implemented using sorting networks. Reversibility implies that, in principle, we should

be able to get rid of the record register, meaning that the reversible sorting operation $\text{REVSORT}_i^{\Gamma_1, \Gamma_2}$ mapping a Γ_1 -sorted list to a Γ_2 -sorted list should perform

$$\text{REVSORT}_i^{\Gamma_1, \Gamma_2} \sum_{l_1 \in \Gamma_1 \text{SIL}} \alpha_{l_1} |l_1\rangle = \sum_{l_1 \in \Gamma_1 \text{SIL}} \alpha_{l_1} |\mathcal{A}_{\Gamma_2}(l_1)\rangle. \quad (26)$$

This can be implemented by uncomputing the record using sorting network operations with 3 ancilla registers, as follows:

$$\begin{aligned} & |\sigma(l)\rangle_1 |0\rangle_2 |0\rangle_3 |12\dots n\rangle_4 \\ & \xrightarrow{\text{SORT}_{1,2}^{\Gamma_2}} |l\rangle_1 |\text{rec}(\sigma)\rangle_2 |0\rangle_3 |12\dots n\rangle_4 \\ & \xrightarrow{\text{SORT}_{1,3}^{\Gamma_1}} |\sigma(l)\rangle_1 |\text{rec}(\sigma)\rangle_2 |\text{rec}(\sigma^{-1})\rangle_3 |12\dots n\rangle_4 \\ & \xrightarrow{\text{UNSORT}_{4,2}^{\Gamma_2}} |\sigma(l)\rangle_1 |0\rangle_2 |\text{rec}(\sigma^{-1})\rangle_3 |\sigma(12\dots n)\rangle_4 \\ & \xrightarrow{\text{SHUFFLE}_{4,3}} |\sigma(l)\rangle_1 |0\rangle_2 |\text{rec}(\sigma^{-1})\rangle_3 |12\dots n\rangle_4 \\ & \xrightarrow{\text{UNSORT}_{1,3}^{\Gamma_1}} |l\rangle_1 |0\rangle_2 |0\rangle_3 |12\dots n\rangle_4. \end{aligned} \quad (27)$$

Note that l is a Γ_2 SIL and $\sigma(l)$ is a Γ_1 SIL.

D. (Quantum) parallel prefix sum computation

In this section, we describe a standard parallel algorithm for computing the prefix sum of an array of n integers, in a reversible manner. For a list of integers $(d_1, d_2, \dots, d_n)$, its prefix sum is

$$\left(d_1, d_1 + d_2, \dots, \sum_{j=1}^i d_j, \dots, \sum_{j=1}^n d_j \right). \quad (28)$$

This algorithm, first proposed in [33], is based on the divide-and-conquer paradigm and uses $O(\log^2 n)$ circuit depth.

This algorithm works on n integer registers. The i th register, denoted by d_i , is initialized as the i th value of the input array. The algorithm contains $O(\log n)$ layers of subroutines, denoted by $D(h, t)$, where h and t (“head” and “tail”) are indices of d . Each $D(h, t)$ is applied after $D(h, h + t/2)$ and $D((h + t)/2, t)$.

We describe $D(h, t)$ in Algorithm 1.

ALGORITHM 1: Subroutine $D(h, t)$.

```

if  $h < t$  then
  for  $i = (h + t)/2 + 1$  to  $t$  in parallel do
     $d_i \leftarrow d_i + d_{(h+t)/2}$ 
  end for
end if
    
```

ALGORITHM 2: Parallel prefix sum computation.

```

for  $i = 1$  to  $\log n$  do
  for  $j = 0$  to  $n/2^i - 1$  in parallel do
     $D(j \cdot 2^i + 1, (j + 1) \cdot 2^i)$ 
  end for
end if

```

Without loss of generality, we only consider the case where n is a power of 2 (note that we can always add up to $O(n)$ zeros at the end of the input list to make the total number a power of 2). Now, the prefix sums can be computed classically using $\log n$ layers of $D(h, t)$, as shown in Algorithm 2.

The subroutine $D(h, t)$ can be implemented in $O(\log n)$ depth since it involves adding $d_{(h+t)/2}$ to $(t-h)/2$ registers. Hence the total circuit depth is $\tilde{O}(\log^2 n)$.

It is straightforward to quantize the prefix sum computation by quantizing each $D(h, t)$. An example is shown in Fig. 3. The quantum circuit depth is also $O(\log^2 n)$.

III. SYMMETRIZATION OF NONSTRICTLY INCREASING LISTS

In this section, we consider symmetrizing nonstrictly increasing lists that may contain duplicated items, i.e., NSILs. In Sec. III A, we introduce some useful subsets and subgroups of S_n related to NSILs. In Sec. III B, we describe the behaviors of SORT and UNSORT on NSILs, and explain why the SIL symmetrization algorithms do not apply to NSILs. In Sec. III C, we describe our new algorithm for symmetrizing a single l where $l \in \text{NSIL}$. We defer the algorithm for superpositions of NSILs (i.e., inputs of the form $\sum_i \alpha_i |l\rangle$) to Sec. IV.

A. Degenerate permutations for NSILs

There is a kind of degeneracy of permutations for $l \in [m]^n$ with repetitions (in this paper, we mostly study $l \in \text{NSIL}_n^m \subset [m]^n$): different permutations $\sigma_1, \sigma_2 \in S_n$ may lead to the same permuted list, i.e., $\sigma_1(l) = \sigma_2(l)$. In this

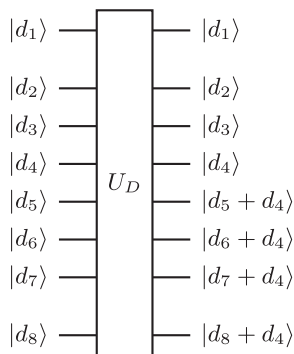


FIG. 3. Quantization of $D(1, 8)$. This unitary can be implemented in $O(\log n)$ depth.

subsection, we show that this degeneracy is the fundamental reason why the Berry *et al.* [4] algorithm does not apply to NSILs. In particular, let H_l denote a subset of the permutation group S_n which contains only permutations preserving the input $l \in [m]^n$, i.e.,

$$H_l := \{h : h(l) = l\}. \quad (29)$$

Then $\sigma \circ h(l) = \sigma(l)$ for all $\sigma \in S_n, h \in H_l$. Note that all $h \in H_l$ must only permute duplicated elements in l . For instance, if $l = 12334$, then H_l contains only the identity permutation and the swap between the 3rd and the 4th elements. One can show that H_l is a subgroup of S_n since $h_1 \circ h_2(l) = l$ if $h_1, h_2 \in H_l$. Furthermore, every permutation $\sigma \in S_n$ can be uniquely decomposed as $\sigma = \sigma' \circ h$ where $h \in H_l$ and σ' is a permutation that preserves the relative order of duplicated items. For the same example of $l = 12334$ where elements 3 and 4 are duplicated, consider a permutation σ satisfying $\sigma(12345) = 51423$. The relative position of 3 and 4 is exchanged in σ , which can be uniquely extracted as another permutation h satisfying $h(12345) = 12435$. Now, the fact that 3 and 4 are out of order in σ can be “corrected,” and we can write $\sigma = \sigma' \circ h$ where $\sigma'(12345) = 51324$. In fact, all elements of a left coset of H_l in S_n permute l in the same way:

$$\sigma H_l(l) = \{\sigma(l)\}. \quad (30)$$

Since every element of S_n must belong to exactly one left coset of H in S_n , we conclude that for any $l \in [m]^n$,

$$S_n = R_l H_l := \{\sigma' \circ h : \sigma' \in R_l, h \in H_l\}, \quad (31)$$

where R_l denotes the set of all permutations preserving the relative orders of duplicated elements in the list l . Note that R_l is a set of canonical elements of all left cosets of H in S_n , which can be chosen arbitrarily.

This leads to another observation: a superposition of all permutations in S_n acting on l and $123\dots n$, respectively, can be represented as

$$\sum_{\sigma \in S_n} |\sigma(l)\rangle |\sigma(12\dots n)\rangle = \sum_{\sigma' \in R_l} |\sigma'(l)\rangle \sum_{h \in H_l} |\sigma' \circ h(12\dots n)\rangle. \quad (32)$$

B. Quantum sorting networks for NSILs

Recall that when introducing sorting-network-related operations in Sec. II, SORT is designed only to sort $\sigma(l)$, where l is strictly increasing, and UNSORT is also only designed to act on SILs. Here we discuss what happens if both operations are applied to NSILs. When l contains duplicates, the SORT operation still transforms $\sigma(l)$ to l , but what is stored in the record depends on the stability of the sorting algorithm. Recall from Sec. II that

we are free to choose the sorting algorithm. A sorting algorithm is called stable if it preserves the relative order of duplicated elements. Therefore, sorting any permutation in a given coset using a stable algorithm produces the record corresponding to the canonical element we picked (R_l), i.e., the record of sorting $\sigma(l)$ will be $\text{rec}(\sigma')$ where $\sigma'(l) = \sigma(l)$ and $\sigma' \in R_l$. [Recall that $\text{rec}(\sigma)$ is the record obtained from sorting $\sigma(l_S)$ where l_S can be any SIL.] On the other hand, unstable sorting networks shuffle the relative order of duplicated elements, so the record of sorting $\sigma(l)$ will be $\text{rec}(\sigma' \circ h')$ where h' is an unknown permutation in H_l depending on σ' . Typically, when a sorting algorithm involves long-range comparisons and swaps, it is more likely to change the relative order of duplicated items. In this paper, we consider general unstable sorting networks, since many efficient sorting networks, including bitonic sort [32], are unstable.

The effect of UNSORT is less clear. We consider two cases:

- (1) If the record is generated by performing SORT on $|\sigma'(l)\rangle|00\dots 0\rangle$, then the record is $|\text{rec}(\sigma' \circ h')\rangle$, where $\sigma' \in R_l, h' \in H_l$. The operation of UNSORT is

$$\text{UNSORT}_{1,2}|l\rangle_1|\text{rec}(\sigma' \circ h')\rangle_2 = |\sigma'(l)\rangle_1|00\dots 0\rangle_2 \quad (33)$$

since UNSORT is the inverse of SORT.

- (2) If the record is $\text{rec}(\sigma' \circ h_S)$ with $\sigma' \in R_l, h_S \in H_l$ and $h_S \neq h'$ [as can be generated by sorting $\sigma' \circ h_S(l_S)$ where $l_S \in \text{SIL}$], then $\sigma'(l)$ can still be generated, but the record register is not reset to $|00\dots 0\rangle$. Instead, UNSORT acts as

$$\begin{aligned} \text{UNSORT}_{1,2}|l\rangle_1|\text{rec}(\sigma' \circ h_S)\rangle_2 \\ = |\sigma'(l)\rangle_1|\text{tra}(\sigma', h_S, l)\rangle_2, \end{aligned} \quad (34)$$

where $|\text{tra}(\sigma', h_S, l)\rangle$ is a trash state depending on the choice of σ', h_S , and l , and must be orthogonal to $|00\dots 0\rangle$. The is because UNSORT is a unitary and

$\langle \text{rec}(\sigma' \circ h') | \text{rec}(\sigma' \circ h_S) \rangle = 0$. The trash state contains unremovable 1's in the record that are generated by swaps between inversions of i and j in $\sigma' \circ h_S(12\dots n)$ with $l_i = l_j$.

To illustrate the problem, we give a very simple example for the bubble sort network shown in Fig. 1. If the record is generated by sorting $\sigma(l_S) = 312$, then $\text{rec}(\sigma) = 110$. (In the first comparator, $3 > 1$, hence its record is 1; in the second comparator, $3 > 2$, hence its record is 1; in the last comparator, $1 < 2$, hence its record is 0.) Now, if $l = 122$, unsorting 122 using the record leads to the following transcript: in the last comparator, no swap is performed and $1 < 2$, hence its record is unchanged; in the second comparator, a swap is performed due to the record, but since $2 = 2$, the record is unchanged and still 1; in the first comparator, a swap is performed and $2 > 1$, so the record is flipped to 0. The state of the record after unsorting is 010, instead of the desired 000. In Table II, we give a complete list of bubble-sort-based UNSORT operations on 122 for all possible records generated by sorting $\sigma(123)$.

Recall that the common final step of existing SIL symmetrization algorithms is unsorting the input $l \in \text{SIL}$ using the superposition of all records of $\sigma \in S_n$. The observations above indicate that, if instead $l \in \text{NSIL}$, these algorithms would, in general, not work, because the scratch qubits in register 2 are entangled with register 1, such that one cannot obtain the desired superposition by discarding them.

Let us consider what kind of sorting algorithms can make UNSORT leave registers 1 and 2 independent, so the state in register 1 is the desired superposition. This can only be achieved if the set of trash states obtained by sorting the members of a left coset of H_l in S_n (such as $\sigma'H_l$ with $\sigma' \in R_l$) is the same for all those left cosets, that is to say,

$$\{\text{tra}(\sigma', h, l) | \forall h \in H_l\} \quad (35)$$

is the same set for all $\sigma' \in R_l$. In this case, register 1 is in the superposition of permuted lists over all left cosets $\sigma'H_l$, and register 2 is in the superposition of all possible trash states

TABLE II. The full table of the action of $\text{UNSORT}_{1,2}$ (based on bubble sort) on $l = 122$ and all possible records generated by permutations of three items. For each permutation σ , we present its operation on the SIL input, $\sigma(123)$, and the corresponding record $\text{rec}(\sigma)$. We then present a step-by-step explanation of the UNSORT procedure from the initial state to the final state. Note that the bold bit in register 2 is the record of the comparator being considered, and the bold digits in register 1 are the elements corresponding to the comparator.

$\sigma(123)$	$\text{rec}(\sigma)$	Initial state checking comparator 3	Step 2 checking comparator 2	Step 3 checking comparator 1	Final state
123	000	$ \mathbf{122}\rangle_1 000\rangle_2$	$ \mathbf{122}\rangle_1 000\rangle_2$	$ \mathbf{122}\rangle_1 000\rangle_2$	$ \mathbf{122}\rangle_1 000\rangle_2$
132	010	$ \mathbf{122}\rangle_1 010\rangle_2$	$ \mathbf{122}\rangle_1 010\rangle_2$	$ \mathbf{122}\rangle_1 010\rangle_2$	$ \mathbf{122}\rangle_1 010\rangle_2$
213	100	$ \mathbf{122}\rangle_1 100\rangle_2$	$ \mathbf{122}\rangle_1 100\rangle_2$	$ \mathbf{122}\rangle_1 100\rangle_2$	$ \mathbf{212}\rangle_1 000\rangle_2$
231	011	$ \mathbf{122}\rangle_1 011\rangle_2$	$ \mathbf{212}\rangle_1 010\rangle_2$	$ \mathbf{221}\rangle_1 000\rangle_2$	$ \mathbf{221}\rangle_1 000\rangle_2$
312	110	$ \mathbf{122}\rangle_1 110\rangle_2$	$ \mathbf{122}\rangle_1 110\rangle_2$	$ \mathbf{122}\rangle_1 110\rangle_2$	$ \mathbf{212}\rangle_1 010\rangle_2$
321	111	$ \mathbf{122}\rangle_1 111\rangle_2$	$ \mathbf{212}\rangle_1 110\rangle_2$	$ \mathbf{221}\rangle_1 100\rangle_2$	$ \mathbf{221}\rangle_1 100\rangle_2$

that are independent of the choice of coset in register 1. Although we cannot rule out the possibility that such sorting algorithms exist, it does not seem straightforward to find them. We thus leave this as an open problem.

Moreover, even if such a sorting algorithm exists, the corresponding SIL symmetrization algorithm will still fail when the input list is a superposition of different NSILs with different patterns of repetition, as the trash states also depend on the choice of l .

C. NSIL symmetrization for a single input

Here we describe our algorithm for symmetrizing NSILs for input state $|l\rangle$ using our observations mentioned in Secs. III A and III B. The algorithm extensively uses all four sorting network operations introduced in Sec. II. We use three registers (the latter two are ancilla registers): register 1 stores the input NSIL l , register 2 stores all records for the sorting algorithm and is initialized as all zero, and register 3 stores the superposition of all $\sigma(12\dots n)$ initially, which can be generated using either the Berry *et al.* [4] SIL symmetrization algorithm [with $\tilde{O}(\log n)$ depth] or our new algorithm described in Sec. IV A [with $\tilde{O}(\log^3(n))$ depth]. Register 3 is used as a platform to mediate multiplications of permutations, which eventually allows us to extract the subgroup structure corresponding to NSILs.

As the first step, we sort each $\sigma(12\dots n)$ reversibly using sorting networks and store the records of swaps:

$$|l\rangle_1 |00\dots 0\rangle_2 \sum_{\sigma \in S_n} |\sigma(12\dots n)\rangle_3 \xrightarrow{\text{SORT}_{1,2}} |l\rangle_1 \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_2 |12\dots n\rangle_3. \quad (36)$$

Next, we apply the swaps stored in the record to shuffle register 1, and then unsort register 3:

$$\xrightarrow{\text{SHUFFLE}_{1,2}} \sum_{\sigma \in S_n} |\sigma(l)\rangle_1 |\text{rec}(\sigma)\rangle_2 |12\dots n\rangle_3 \xrightarrow{\text{UNSORT}_{3,2}} \sum_{\sigma \in S_n} |\sigma(l)\rangle_1 |00\dots 0\rangle_2 |\sigma(12\dots n)\rangle_3. \quad (37)$$

The decomposition of a permutation for nonstrictly increasing lists can now be applied to rewrite the above state:

$$\sum_{\sigma \in S_n} |\sigma(l)\rangle_1 |00\dots 0\rangle_2 |\sigma(12\dots n)\rangle_3 = \sum_{\sigma' \in R_l} |\sigma'(l)\rangle_1 |00\dots 0\rangle_2 \sum_{h \in H_l} |\sigma' \circ h(12\dots n)\rangle_3. \quad (38)$$

Now, we perform sorting network operations on the state above to sort the $\sigma'(l)$ strings:

$$\sum_{\sigma' \in R_l} |\sigma'(l)\rangle_1 |00\dots 0\rangle_2 \sum_{h \in H_l} |\sigma' \circ h(12\dots n)\rangle_3 \xrightarrow{\text{SORT}_{1,2}} \sum_{\sigma' \in R_l} |l\rangle_1 |\text{rec}(\sigma' \circ h')\rangle_2 \sum_{h \in H_l} |\sigma' \circ h(12\dots n)\rangle_3. \quad (39)$$

Next, we apply the swaps recorded in register 2 to unshuffle register 3:

$$\xrightarrow{\text{UNSHUFFLE}_{3,2}} \sum_{\sigma' \in R_l} |l\rangle_1 |\text{rec}(\sigma' \circ h')\rangle_2 \cdot \sum_{h \in H_l} |h'^{-1} \circ \sigma'^{-1} \circ \sigma' \circ h(12\dots n)\rangle_3 = \sum_{\sigma' \in R_l} |l\rangle_1 |\text{rec}(\sigma' \circ h')\rangle_2 \cdot \sum_{h \in H_l} |h'^{-1} \circ h(12\dots n)\rangle_3 = \sum_{\sigma' \in R_l} |l\rangle_1 |\text{rec}(\sigma' \circ h')\rangle_2 \sum_{h \in H_l} |h(12\dots n)\rangle_3, \quad (40)$$

where in the last line we used the fact that $h'^{-1} \in H_l$ and $h'^{-1}H_l = H_l$ when $h' \in H_l$.

Now, to obtain a superposition of $\sigma(l)$ strings, it suffices to unsort l using the records:

$$\xrightarrow{\text{UNSORT}_{1,2}} \sum_{\sigma' \in R_l} |\sigma'(l)\rangle_1 |00\dots 0\rangle_2 \sum_{h \in H_l} |h(12\dots n)\rangle_3. \quad (41)$$

We notice that the strings in register 3 are now only superpositions of $h(12\dots n)$ over $h \in H_l$, and are independent of which permutation is applied in register 1. Therefore, if the input is a single NSIL basis state $|l\rangle$, one can simply discard registers 2 and 3, and the state in register 1 is $\sum_{\sigma \in S_n} |\sigma(l)\rangle$. The output fidelity of single-input symmetrization is 1 if the resource state $\sum_{\sigma} |\sigma(12\dots n)\rangle$ is prepared exactly. The Berry *et al.* [4] algorithm can produce this state with high probability, as mentioned in Sec. II B.

IV. NSIL SYMMETRIZATION FOR A SUPERPOSED INPUT

If the input to the above procedure is some arbitrary superposition of different NSILs with different patterns of repetitions, registers 1 and 3 will be entangled. As a simple example, if $|\psi_{\text{in}}\rangle = (1/\sqrt{3})|1223\rangle + \sqrt{2/3}|1333\rangle$, then the state of the algorithm at the step of Eq. (41) is

$$\frac{1}{\sqrt{3}} \sum_{\sigma' \in S_{1223}} |\sigma'(1223)\rangle_1 |00\dots 0\rangle_2 \sum_{h \in H_{1223}} |h(1234)\rangle_3 + \sqrt{\frac{2}{3}} \sum_{\sigma' \in S_{1333}} |\sigma'(1333)\rangle_1 |00\dots 0\rangle_2 \sum_{h \in H_{1333}} |h(1234)\rangle_3, \quad (42)$$

from which we cannot obtain $|\psi_{\text{out}}\rangle$ by simply discarding register 3. Hence, we must erase the scratch qubits in register 3 for all possible input strings l coherently. This can be done by inverting the circuit preparing the superposition of the H_l subgroup ($\sum_{h \in H_l} |h(12\dots n)\rangle$) from $|l\rangle$. Therefore, in this section, we design a unitary to implement the following subgroup superposition preparation procedure:

$$\sum_l \alpha_l |l\rangle |12\dots n\rangle \mapsto \sum_l \alpha_l |l\rangle \frac{1}{\sqrt{|H_l|}} \sum_{h \in H_l} |h(12\dots n)\rangle. \quad (43)$$

In this section, we first design a poly(log n)-depth quantum algorithm for SIL symmetrization with perfect success probability in Sec. IV A, which is used as a subroutine to solve the subgroup superposition preparation problem in Sec. IV B.

A. An exact SIL symmetrization algorithm

Our exact SIL symmetrization algorithm relies on a classical parallel algorithm for generating a random permutation proposed by [14]. However, that algorithm was introduced in a nonreversible way. Therefore, in this subsection, we present that algorithm in a pedagogical manner and indicate along the way how to make it reversible.

1. Lower exceeding sequences

The central tool used in the Alonso and Schott [14] algorithm is the concept of lower exceeding sequences (LESSs).

Definition 3. (Lower exceeding sequences) A sequence $s \in \{1, 2, \dots, n\}^n$ is a lower exceeding sequence if and only if $1 \leq s_i \leq i$ for all $i \in [n]$.

The number of LESSs of length n is $n!$, while the number of permutations of n items is also $|S_n| = n!$. Reference [14] efficiently constructs a bijective mapping between these sets and uses it to generate a random permutation by generating an LES and then applying the mapping. Our quantum algorithm is based on a similar approach.

The mapping of [14] is defined as follows.

Definition 4. (s^σ as in [14]) The LES $s^\sigma \in [n]^n$ corresponding to a permutation $\sigma \in S_n$ has i th element

$$s_i^\sigma = |\{j: \sigma^{-1}(j) \leq \sigma^{-1}(i); j \leq i\}|$$

for all $i \in [n]$.

It is easy to verify that s^σ is an LES for any $\sigma \in S_n$, but it is less intuitive to explain the meaning of the values

in s^σ . Fortunately, Alonso and Schott [14] give an intuitive way to understand the meaning of s^σ diagrammatically, as follows.

First, from each permutation $\sigma \in S_n$, one can compute s^σ . Consider a permutation σ acting on the list of first n positive integers, giving $\sigma(123\dots n)$. Let $\sigma(123\dots n)_i$ denote the i th element of $\sigma(123\dots n)$. One can draw the permutation diagrammatically on an $n \times n$ grid: if $\sigma(12\dots n)_i = j$, then block (i, j) of the grid is filled, where i is the row index and j is the column index. We call this a permutation-LES diagram. The j th element of the generated LES is simply the number of filled blocks in the rectangle between $(1, 1)$ and (i, j) . Since each column and row has and only has 1 filled block, the number of filled blocks between $(1, 1)$ and (i, j) is guaranteed to be at most j . Fig. 4 depicts an example with $n = 6$.

Conversely, from each LES, one can generate its corresponding permutation by again drawing the permutation-LES diagram. Suppose the LES is $(s_1 s_2 \dots s_n)$ and our goal is to compute y_j , the y coordinate of the filled block in the j th column, for all $j \in \{1, \dots, n\}$. We run through the LES from the last element s_n to the first element s_1 . To have exactly s_n filled blocks between $(1, 1)$ and (y_n, n) , we must fill the s_n th block of the last column, which means $y_n = s_n$. Since every row and column in the diagram should only have 1 filled block, we call the y_n th row *used*. Next, we observe that the s_{n-1} th unused row must be filled in the second-to-last column such that there are exactly s_{n-1} filled blocks between $(1, 1)$ and $(y_{n-1}, n-1)$, because all of the first $s_{n-1} - 1$ unused rows

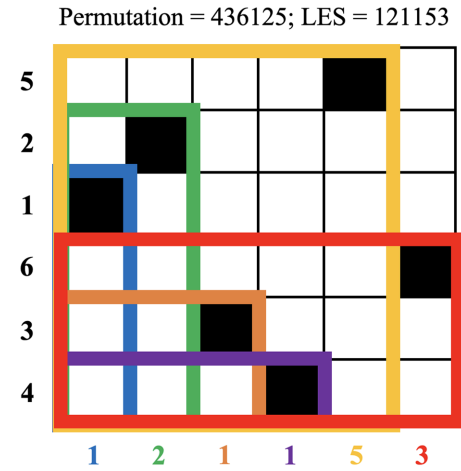


FIG. 4. The permutation-LES diagram for $\sigma(123456) = 436125$, which appears on the vertical axis. The corresponding LES is 121153, as shown on the horizontal axis. To determine the element of the LES in a given column from the permutation, we draw a rectangle whose bottom left corner is at the origin and whose top right corner is at the filled square in the corresponding column, and count the number of filled blocks in that rectangle. The rectangles are colored to indicate the corresponding element of the LES (i.e., the column).

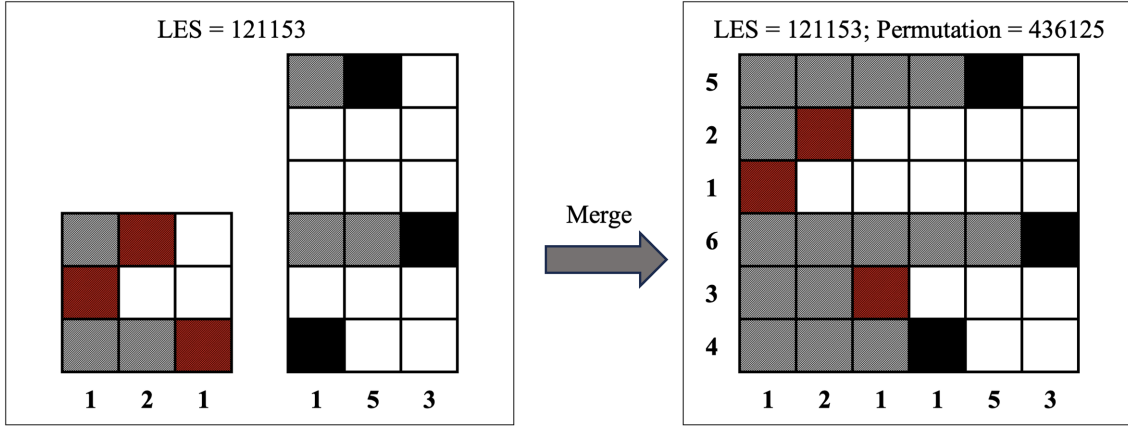


FIG. 5. Action of the merging algorithm on the permutation $\sigma(123456) = 436125$ (whose permutation-LES diagram is illustrated in Fig. 4). The gray squares are “unavailable” since each row can only have one filled square. Dark red squares indicate filled blocks of the left diagram. When merging the two subdiagrams, the i th row in the left diagram is inserted into the i th unfilled row of the right diagram.

will eventually be used and counted when determining the $(n-1)$ st element of the LES. Similarly, we can find all other values of y_j from $j = n-2$ to $j = 1$ by filling the s_j th unused row of the j th column. Finally, the permutation can be read out directly from the diagram.

A naive implementation of the above procedure for converting an LES to its corresponding permutation takes $O(n \log n)$ depth. Fortunately, [14] provides a $\text{poly}(\log n)$ -depth algorithm to do the conversion, which is outlined next.

2. From LES to permutation with reversible merging

Alonso and Schott [14] show that the blocks can be filled in a divide-and-conquer manner, as follows. Divide the LES into two parts at the middle, make the subdiagrams for both parts individually, and merge them together. The first two steps are straightforward, but the “merging” step is nontrivial. A simple merging strategy is not hard to see: one can insert every row of the left diagram into the unfilled rows of the right diagram. More specifically, we first concatenate two diagrams together and keep the right diagram unchanged. Then, if the j th column of the left diagram has the y_j th block filled, we move this block to the y_j th unfilled row of the right diagram.

This merging strategy is illustrated in Fig. 5 where the same permutation as in Fig. 4 [$\sigma(123456) = 436125$] is recovered.

Performing this merging naively takes $O(n)$ steps for each pair of subdiagrams. To reduce it to $O(\log n)$ steps, Alonso and Schott [14] introduce another way of representing the diagrams: a sorted list of (r_i, c_i) tuples, where each (r_i, c_i) represents the row and column coordinates of a filled block in the diagram. We describe the general algorithm together with a running example that shows how it works in the case depicted in Fig. 4.

By default, the filled blocks are sorted by their row coordinates, i.e., the comparison rule Γ_f is

$$(r_i, c_i) <_{\Gamma_f} (r_j, c_j) \quad \text{iff } r_i < r_j. \quad (44)$$

We can reduce the problem of merging diagrams to calculating the final Γ_f -sorted list, denoted by T_f , which contains all row indices from 1 to n , i.e.,

$$T_f = (1, z_1)(2, z_2) \dots (n, z_n). \quad (45)$$

Once such a list is obtained, the corresponding permutation is simply $\sigma(12 \dots n) = z_1 z_2 \dots z_n$. For instance, the T_f corresponding to $\sigma(123456) = 436125$ in Fig. 5 is

$$T_f = (1, 4)(2, 3)(3, 6)(4, 1)(5, 2)(6, 5). \quad (46)$$

The key strategy of Alonso and Schott [14] is to first calculate the Γ_f -sorted tuple lists for the left and right subdiagrams, and merge them together to build the whole diagram. Note that each subdiagram’s list can also be calculated recursively using the same divide-and-conquer method. Without loss of generality, we only describe the last step of the algorithm, i.e., merging the last two subdiagrams to obtain T_f , since all previous steps use the same procedure. We also emphasize that we describe the whole algorithm in a reversible manner—unlike in the original paper [14]—so that it can be directly quantized.

We use T_0 to denote the direct concatenation of two Γ_f -sorted lists of both subdiagrams. If both lists have k elements, then

$$T_0 = (r_1, c_1)(r_2, c_2) \dots (r_k, c_k)(r_{k+1}, c_{k+1}) \dots (r_{2k}, c_{2k}), \quad (47)$$

where the first half and the second half are both Γ_f sorted. T_0 is sorted under the following comparison rule, denoted by Γ_0 : $(r_i, c_i) <_{\Gamma_0} (r_j, c_j)$, if either of the following two conditions is satisfied:

- (i) $c_i \leq k$ AND $c_j > k$;
- (ii) $r_i < r_j$ AND $(c_i, c_j \leq k$ OR $c_i, c_j > k)$.

For the example in Fig. 5, $k = 3$, and the Γ_f -sorted tuple list of the first subfigure is $(1, 3)(2, 1)(3, 2)$ for the left-hand side and $(1, 4)(3, 6)(6, 5)$ for the right-hand side. Therefore,

$$T_0 = (1, 3)(2, 1)(3, 2)(1, 4)(3, 6)(6, 5). \quad (48)$$

Our objective is to convert T_0 to T_f in a reversible manner. This can be done by first sorting T_0 reversibly such that every c_i is at its correct position, i.e., we need to permute T_0 using some permutation ω such that $c_{\omega(i)} = z_i$. This sorted list, denoted by T_1 , is

$$\begin{aligned} T_1 &= (r_{\omega(1)}, c_{\omega(1)})(r_{\omega(2)}, c_{\omega(2)}) \dots (r_{\omega(2k)}, c_{\omega(2k)}) \\ &= (r_{\omega(1)}, z_1)(r_{\omega(2)}, z_2) \dots (r_{\omega(2k)}, z_{2k}). \end{aligned} \quad (49)$$

For the example in Fig. 5,

$$T_1 = (1, 4)(1, 3)(3, 6)(2, 1)(3, 2)(6, 5). \quad (50)$$

Note that the sorting procedure plays the same role as inserting elements from the left-hand side in the diagrams in Fig. 5 to the right-hand side. In the second step, we change the $r_{\omega(i)}$ values in T_1 to i reversibly so that T_f is obtained. We explain how to implement both steps as follows.

Step 1: $T_0 \mapsto T_1$. We reversibly sort T_0 to obtain T_1 using comparison-based sorting algorithms, based on the rules given by Alonso and Schott [14]. First, we define $\tilde{T}_0$ and $\tilde{T}_1$ by concatenating each tuple in the lists with its index:

$$\begin{aligned} \tilde{T}_0 &= (1, r_1, c_1)(2, r_2, c_2) \dots (2k, r_{2k}, c_{2k}) \\ \tilde{T}_1 &= (\omega(1), r_{\omega(1)}, c_{\omega(1)})(\omega(2), r_{\omega(2)}, c_{\omega(2)}) \dots \\ &\quad \dots (\omega(2k), r_{\omega(2k)}, c_{\omega(2k)}) \\ &= (\omega(1), r_{\omega(1)}, z_1)(\omega(2), r_{\omega(2)}, z_2) \dots \\ &\quad \dots (\omega(2k), r_{\omega(2k)}, z_{2k}). \end{aligned} \quad (51)$$

It is trivial to reversibly construct $\tilde{T}_0$ from T_0 , and $\tilde{T}_0$ is also Γ_0 sorted. For the example in Fig. 5,

$$\begin{aligned} \tilde{T}_0 &= (1, 1, 3)(2, 2, 1)(3, 3, 2)(4, 1, 4)(5, 3, 6)(6, 6, 5), \\ \tilde{T}_1 &= (4, 1, 4)(1, 1, 3)(5, 3, 6)(2, 2, 1)(3, 3, 2)(6, 6, 5). \end{aligned} \quad (52)$$

According to Ref. [14], $\tilde{T}_1$ is a Γ_1 -sorted tuple list, where Γ_1 is defined as follows. (Here we simply describe Γ_1

without proving why $\tilde{T}_1$ is Γ_1 sorted; see Ref. [14] for the proof.) For $\omega(i) \neq \omega(j)$, $(\omega(i), r_{\omega(i)}, c_{\omega(i)}) <_{\Gamma_1} (\omega(j), r_{\omega(j)}, c_{\omega(j)})$ if and only if any of the following conditions is satisfied:

- (1) $\omega(i), \omega(j) \leq k$ AND $\omega(i) < \omega(j)$;
- (2) $\omega(i), \omega(j) > k$ AND $\omega(i) < \omega(j)$;
- (3) $\omega(i) \leq k, \omega(j) > k$ AND $r_{\omega(i)} + [\omega(j) - k] \leq r_{\omega(j)}$;
- (4) $\omega(i) > k, \omega(j) \leq k$ AND $r_{\omega(j)} + [\omega(i) - k] \leq r_{\omega(i)}$.

One can now get $\tilde{T}_1$ by sorting $\tilde{T}_0$ using any reversible sorting network without knowing ω explicitly, since $\tilde{T}_1 = \mathcal{A}_{\Gamma_1}(\tilde{T}_0)$ and $\tilde{T}_0 = \mathcal{A}_{\Gamma_0}(\tilde{T}_1)$. This step takes $\tilde{O}(\log n)$ depth.

Next, we convert $\tilde{T}_1$ to T_1 by removing the first element of each tuple, $\omega(i)$, reversibly. This requires us to compute the position of the tuple in the $\tilde{T}_0$ list using i , $r_{\omega(i)}$, and $c_{\omega(i)}$. A key observation is that, if $(i, r_i, c_i) <_{\Gamma_0} (j, r_j, c_j)$ and $c_i, c_j \leq k$, then $(\omega(i), r_{\omega(i)}, c_{\omega(i)}) <_{\Gamma_1} (\omega(j), r_{\omega(j)}, c_{\omega(j)})$, which also holds when $c_i, c_j > k$. This implies that the relative orders of each subdiagram are preserved by sorting under Γ_1 . Therefore, we can check whether $(\omega(i), r_{\omega(i)}, c_{\omega(i)})$ is from the left diagram or the right diagram (which is easy by comparing $c_{\omega(i)}$ and k) and compute its rank in its subdiagram (denoted by $d_{L, \omega(i)}$ if it is in the left subdiagram or $d_{R, \omega(i)}$ if it is in the right subdiagram) and compute

$$\omega(i) = \begin{cases} d_{L, \omega(i)}, & c_{\omega(i)} \leq k \\ d_{R, \omega(i)} + k, & c_{\omega(i)} > k. \end{cases} \quad (53)$$

Now, it suffices to compute all $d_{L, \omega(i)}$ and $d_{R, \omega(i)}$ values. To do so, we first initialize both arrays by

$$\begin{aligned} d_{L, \omega(i)} &\leftarrow 1, & d_{R, \omega(i)} &\leftarrow 0 \text{ if } c_{\omega(i)} \leq k, \\ d_{L, \omega(i)} &\leftarrow 0, & d_{R, \omega(i)} &\leftarrow 1 \text{ if } c_{\omega(i)} > k. \end{aligned} \quad (54)$$

Then, using the prefix sum algorithm described in Sec. IID, both arrays can be computed in $O(\log^2 n)$ depth.

Step 2: $T_1 \mapsto T_f$. Converting T_1 to T_f is essentially performing $r_{\omega(i)} \mapsto i$. Observe that if $c_{\omega(i)} > k$, then $r_{\omega(i)} = i$ and nothing needs to be done, since filled blocks in the right sub-diagram stay in the same rows during the merge. For $c_{\omega(i)} \leq k$, this update can be easily done by $r_{\omega(i)} \leftarrow r_{\omega(i)} + d_{R, \omega(i)}$ since a filled block in the left subdiagram is moved to another row in the final diagram, and the new row is its row index in the left subdiagram ($r_{\omega(i)}$) plus the number of rows below ($d_{R, \omega(i)}$) that are filled by blocks in the right subdiagram. This process takes $O(\log^2 n)$ depth since it involves computing $d_{R, \omega(i)}$.

3. The SIL symmetrization algorithm

Since we can implement the merging process ($T_0 \mapsto T_1 \mapsto T_f$) in depth $\tilde{O}(\log^2 n)$ ($\tilde{O}(\log n)$ for $\tilde{T}_0 \mapsto \tilde{T}_1$, $\tilde{O}(\log^2 n)$ for $\tilde{T}_1 \mapsto T_1$, and $\tilde{O}(\log^2 n)$ for $T_1 \mapsto T_f$), and this divide-and-conquer algorithm uses $O(\log n)$ layers of merging, the total depth of the LES-to-permutation conversion is $\tilde{O}(\log^3 n)$. To generate the superposition of all permutations of the nontrivial input $l \in \text{SIL}$ with $|l| = n$ and $l \neq (123\dots n)$ in low depth, we “move” the permutations stored in $\sigma(12\dots n)$ to l using the following quantum algorithm. Note that all ancilla qubits involved are omitted.

- (1) Generate the equal superposition of all LESs of length n : $|0\rangle^{\otimes n} \mapsto (1/\sqrt{n!}) \sum_{s_i=1}^i |s_i\rangle$.
- (2) Generate permutations from LESs using the quantized Alonso and Schott [14] algorithm: $(1/\sqrt{n!}) \sum_{s_i=1}^i |s_i\rangle \mapsto (1/\sqrt{n!}) \sum_{\sigma \in S_n} |\sigma(12\dots n)\rangle$.
- (3) Sort each permuted list using any (irreversible) sorting network:

$$\begin{aligned} & \frac{1}{\sqrt{n!}} |00\dots 0\rangle_1 \sum_{\sigma \in S_n} |\sigma(12\dots n)\rangle_2 \\ & \xrightarrow{\text{SORT}_{2,1}} \frac{1}{\sqrt{n!}} \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_1 |12\dots n\rangle_2, \end{aligned} \quad (55)$$

where register 2 can be dropped since it is independent of σ .

- (4) Unsort the input list l using the records:

$$\begin{aligned} & \frac{1}{\sqrt{n!}} \sum_{\sigma \in S_n} |\text{rec}(\sigma)\rangle_1 |l\rangle_2 \\ & \xrightarrow{\text{UNSORT}_{2,1}} \frac{1}{\sqrt{n!}} \sum_{\sigma \in S_n} |00\dots 0\rangle_1 |\sigma(l)\rangle_2, \end{aligned} \quad (56)$$

where register 1 can be dropped since it is independent of $\sigma(l)$.

The whole process can be implemented as a single unitary, so it is guaranteed to produce the ideal symmetrized state.

B. NSIL symmetrization for superposed inputs

Recall that our algorithm for NSIL symmetrization involves preparing the equal superposition over the H_l subgroup. Our strategy consists of two steps: detecting duplicates and preparing random permutations of duplicates in superposition.

1. Duplicate detection algorithm

We now describe an $\tilde{O}(\log n)$ -depth quantum algorithm to detect all duplicated items in an ordered list. This algorithm takes as input $|l\rangle_1 |0\rangle_2$ where l is an NSIL and

each integer in l is assigned $O(\log n)$ qubits in register 2 to record how many times this integer is repeated in l . The output of this algorithm is $|l\rangle |\text{dup}(l)\rangle$ where $\text{dup}(l)$ records all repetitions: if there exists $n_i > 1$ for i such that $l_{i-1} < l_i = l_{i+1} = \dots = l_{i+n_i-1} < l_{i+n_i}$, then $\text{dup}(l)_i = n_i$; for all other cases, $\text{dup}(l)_i = 0$. For example, if $l = (13333338)$, then $\text{dup}(l) = (06000000)$.

We start by describing the classical algorithm to find $\text{dup}(l)$, which can be quantized afterwards. The basic element of this algorithm is a block, denoted by B . Each block labeled by h, t (head and tail) processes a range $[h, t]$ of l . The algorithm receives information from 2 blocks in the previous layer processing its subranges $[h, (h+t)/2]$ and $[(h+t)/2+1, t]$, and checks whether there is a range of repeated elements across the 2 subranges by checking whether $l[(h+t)/2] = l[(h+t)/2+1]$. Each block outputs 2 integers to the next layer, n_h and n_t , which are the numbers of elements equal to l_h and l_t , respectively, in the range $[h, t]$. The algorithm takes 4 integers from the previous layer, which are the output values of $B_{h,(h+t)/2}$, denoted by $n_{h,1}, n_{t,1}$, and of $B_{(h+t)/2+1,t}$, denoted by $n_{h,2}, n_{t,2}$. A complete description of $B_{h,t}$ can be found in Algorithm 3.

The whole algorithm contains $O(\log n)$ layers of blocks to find all ranges of duplicates bottom to top. A block in the i th layer of this algorithm processes a range of 2^{i-1} bits. To quantize the above duplicate detection algorithm, it suffices to quantize every block B using a unitary U_B with ancilla qubits, as shown in Fig. 6.

2. Subgroup superposition preparation

Now that the positions and sizes of repetitions have been recorded, we can modify the exact SIL symmetrization algorithm to prepare $\sum_{h \in H_l} |h(12\dots n)\rangle$ as follows. Note that H_l only contains permutations of repeated items, and all other items must stay at their initial positions. We claim that H_l corresponds to a specific class of LESs, denoted by LES_l , which is a set of LESs of length n satisfying the following conditions:

- (i) if $l_{i-1} < l_i < l_{i+1}$, then $s_i = i$;
- (ii) if $l_{a-1} < l_a = l_{a+1} = \dots = l_b < l_{b+1}$, then $a \leq s_i \leq i$ for all $a \leq i \leq b$.

More formally,

$$\text{LES}_l := \{s : \forall i \in [n], \min_{j=l_i} j \leq s_i \leq \max_{j=l_i} j\}. \quad (57)$$

Theorem 1. Each $h \in H_l$ can be generated from an $s \in \text{LES}_l$, and each $s \in \text{LES}_l$ can be computed from an $h \in H_l$, using the rules in Sec. IV A.

Proof. To show that there is a one-to-one mapping, we verify that $|H_l| = |\text{LES}_l|$. Suppose V_l is the set of all different elements in l , and each $v \in V_l$ appears n_v times in l . Then

ALGORITHM 3: Subroutine $B_{h,t}(n_{h,1}, n_{t,1}, n_{h,2}, n_{t,2})$.

```

if  $h = t$  then
   $n_h \leftarrow 1$ 
   $h_t \leftarrow h$ 
else
   $h_1 \leftarrow h$ 
   $t_1 \leftarrow (h + t)/2$ 
   $h_2 \leftarrow (h + t)/2 + 1$ 
   $t_2 \leftarrow t$ 
  if  $l_{t_1} = l_{h_2}$  then
    if  $l_{t_1} \neq l_{h_1}$  AND  $l_{h_2} \neq l_{t_2}$  then
       $\text{dup}(l)_{t_1-n_{t,1}} \leftarrow n_{t,1} + n_{h,2}$ 
       $n_h \leftarrow n_{h,1}$ 
       $n_t \leftarrow n_{t,2}$ 
    else if  $l_{t_1} \neq l_{h_1}$  AND  $l_{h_2} = l_{t_2}$  then
       $n_h \leftarrow n_{h,1}$ 
       $n_t \leftarrow n_{t,2} + n_{t,1}$ 
    else if  $l_{t_1} = l_{h_1}$  AND  $l_{h_2} \neq l_{t_2}$  then
       $n_h \leftarrow n_{h,1} + n_{h,2}$ 
       $n_t \leftarrow n_{t,2}$ 
    else if  $l_{t_1} = l_{h_1}$  AND  $l_{h_2} = l_{t_2}$  then
       $n_h \leftarrow n_{h,1} + n_{h,2}$ 
       $n_t \leftarrow n_{h,1} + n_{h,2}$ 
    end if
  else
     $n_h \leftarrow n_{h,1}$ 
     $n_t \leftarrow n_{t,2}$ 
    if  $1 < n_{t,1} < (t - h)/2$  then
       $\text{dup}(l)_{t_1-n_{t,1}} \leftarrow n_{t,1}$ 
    end if
    if  $1 < n_{h,2} < (t - h)/2$  then
       $\text{dup}(l)_{h_2} \leftarrow n_{h,2}$ 
    end if
  end if
end if

```

$$|H_l| = \prod_{v \in V_l} n_v!. \quad (58)$$

Similarly, for any v with $n_v = 1$ and $l_i = v$, the corresponding $s \in \text{LES}_l$ must satisfy $s_i = i$. If $n_v > 1$, then there

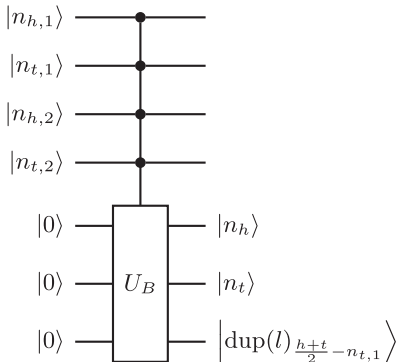


FIG. 6. Quantization of B .

are n_v vs in l and there are $n_v!$ possible values allowed. Therefore, we also have

$$|\text{LES}_l| = \prod_{v \in V_l} n_v! = |H_l|. \quad (59)$$

One can use the rules in Sec. IV A to calculate the corresponding $s^h \in \text{LES}_l$ from an $h \in H_l$ as follows. Since permutations only exist between repeated elements, and repeated elements are adjacent in $l \in \text{NSIL}_n$, we observe that

$$\min_{l_j=l_i} j \leq h(12\dots n)_i \leq \max_{l_j=l_i} j, \quad (60)$$

and

$$h(12\dots n)_j < h(12\dots n)_i < h(12\dots n)_k \quad (61)$$

if $l_j < l_i < l_k$. Recall that $s^h_{h(12\dots)_i}$ in the permutation-LES diagram of $h(12\dots n)$ is determined by counting the number of filled blocks from $(1, 1)$ to block $[h(12\dots)_i, i]$. The equation above implies that for all $l_j < l_i$, $h(12\dots)_j$ must be between $(1, 1)$ and $(h(12\dots)_i, i)$, so

$$s_i^h \geq 1 + \max_{l_j < l_i} j = \min_{l_j=l_i} j. \quad (62)$$

For all $l_j = l_i$, since permutations are allowed, all those repeated items may be included in the range, so

$$s_i^h \leq n_{l_i} + \max_{l_j < l_i} j = \max_{l_j=l_i} j. \quad (63)$$

Therefore, $s^h \in \text{LES}_l$. This immediately implies that one can use the rules in Sec. IV A to find the corresponding $h \in H_l$ from s^h . ■

Therefore, to produce a superposition over H_l , it suffices to employ the SIL symmetrization algorithm with the LES state preparation step modified such that the initial state is

$$\sum_{l \in \text{NSIL}_n} \alpha_l |l\rangle \sum_{s \in \text{LES}_l} |s\rangle. \quad (64)$$

Since the pattern of repetitions in l has been found, this state can still be prepared as a product state conditioned on the repetitions in l . The circuit depth is $O(1)$.

V. EFFICIENT CONVERSION FROM SECOND QUANTIZATION TO FIRST QUANTIZATION

To convert any given second-quantized state to its first-quantized counterpart, we first transform the second-quantized state to an NSIL before employing the (anti-)symmetrization algorithm on the increasing lists. To do so, we design a novel reversible circuit that takes as input a list of occupation numbers (i.e., a

second-quantized representation) and outputs an increasing list of positions (or modes) occupied, where each mode is assigned with a number written in its binary representation. More specifically, our circuit transforms $W_0 := n_0 n_1 \dots n_{m-1}$ (where $n_i \geq 0$ is the occupation number of mode i) into $W_f := 0^{n_0} 1^{n_1} \dots (m-1)^{n_{m-1}}$. Our approach is based on reversible sorting networks and takes $\tilde{O}(\log^2 n)$ depth where $n = \sum_{i=0}^{m-1} n_i$. The main idea for converting W_0 to W_f reversibly is to first generate n empty registers; then use sorting networks to move n_i of them in between of the $(i-1)$ th register and the i th register, for all $i \in [m]$; and use the new positions of the empty registers to compute the index i locally for each of them. In more detail, the algorithm proceeds as follows.

First, we calculate the prefix sums of W_0 and produce the list of 2 tuples:

$$W_1 = (n_0, n_0)(n_1, n_0 + n_1)(n_2, n_0 + n_1 + n_2) \dots (n_{m-1}, n). \quad (65)$$

This can be done in $O(\log^2 n)$ depth as described in Sec. IID. We also add indices to all elements in W_1 , giving a 3-tuple list

$$\tilde{W}_1 = (0, n_0, n_0)(1, n_1, n_0 + n_1)(2, n_2, n_0 + n_1 + n_2) \dots (m-1, n_{m-1}, n). \quad (66)$$

Next, since the output list contains n integers, we add another n elements with $(m, 0, j)$ where $j \in \{0, 1, \dots, n-1\}$ is the index of the new element added. The resulting list is

$$\tilde{W}_2 = (0, n_0, n_0)(1, n_1, n_0 + n_1) \dots \times (m-1, n_{m-1}, n)(m, 0, 0)(m, 0, 1) \dots (m, 0, n-1). \quad (67)$$

Note that $\tilde{W}_2$ is Γ_{W_2} -sorted where $(a_1, b_1, c_1) <_{\Gamma_{W_2}} (a_2, b_2, c_2)$ if and only if either of the following 2 conditions is satisfied:

- (i) $a_1 < a_2$;
- (ii) $a_1 = a_2 = m$ AND $c_1 < c_2$.

Now, using reversible sorting networks, we sort $\tilde{W}_2$ to obtain

$$\tilde{W}_3 = (m, 0, 0)(m, 0, 1) \dots (m, 0, n_0 - 1)(0, n_0, n_0)(m, 0, n_0) \dots (m, 0, n_0 + n_1 - 1)(1, n_1, n_0 + n_1)(m, 0, n_0 + n_1) \dots (m, 0, n-1)(m-1, n_{m-1}, n), \quad (68)$$

which is Γ_{W_3} sorted, where $(a_1, b_1, c_1) <_{\Gamma_{W_3}} (a_2, b_2, c_2)$ if and only if either of the following 2 conditions is satisfied:

- (i) $c_1 < c_2$,
- (ii) $c_1 = c_2$ AND $a_1 < a_2$.

Now, we have allocated n_i such elements (with first element being m) between $(i-1, n_{i-1}, \sum_{j=0}^{i-1} n_j)$ and $(i, n_i, \sum_{j=0}^i n_j)$, and we need to assign the value i to each of them. For the k th element in $\tilde{W}_3$, if the first value is m (meaning that this is a newly added element) and the third value is i_k (the index of new elements), then the corresponding i value is simply $k - i_k$ (the difference between the index in $\tilde{W}_3$ and the index in all new elements). We add this value to the second value for each new element, obtaining

$$\begin{aligned} \tilde{W}_4 = & (m, 0, 0)(m, 0, 1) \dots (m, 0, n_0 - 1)(0, n_0, n_0) \\ & (m, 1, n_0) \dots (m, 1, n_0 + n_1 - 1)(1, n_1, n_0 + n_1) \\ & (m, 2, n_0 + n_1) \dots (m, m-1, n-1) \\ & (m-1, n_{m-1}, n). \end{aligned} \quad (69)$$

$\tilde{W}_4$ is also Γ_{W_3} -sorted since we only modify the second values, which do not play a role in the Γ_{W_3} comparison rule. Now, if we only consider elements whose first value is m , we have already obtained the desired W_f list in their second values. Hence, in the final stage, we remove all other elements reversibly. We first sort $\tilde{W}_4$ under Γ_{W_2} to group together elements with first values less than m , giving

$$\begin{aligned} \tilde{W}_5 = & (0, n_0, n_0)(1, n_1, n_0 + n_1) \\ & \dots (m-1, n_{m-1}, n)(m, 0, 0) \dots \end{aligned} \quad (70)$$

We can first remove all second values for the first m elements by subtracting the difference between $\sum_{j=0}^i n_j$ and $\sum_{j=0}^{i-1} n_j$, which is equal to n_i . For $i=0$, it is trivial to implement $(0, n_0, n_0) \mapsto (0, 0, n_0)$. This gives us

$$\tilde{W}_6 = (0, 0, n_0)(1, 0, n_0 + n_1) \dots (m-1, 0, n)(m, 0, 0) \dots, \quad (71)$$

which is still Γ_{W_2} sorted. We sort it again under Γ_{W_3} to obtain

$$\begin{aligned} \tilde{W}_7 = & (m, 0, 0) \dots (m, 0, n_0 - 1)(0, 0, n_0)(m, 1, n_0) \\ & \dots (m, 1, n_0 + n_1 - 1)(1, 0, n_0 + n_1)(m, 2, n_0 + n_1) \\ & \dots (m, m-1, n-1)(m-1, 0, n). \end{aligned} \quad (72)$$

Next, we notice that $(i, 0, \sum_{j=0}^i n_i)$ must be the $(\sum_{j=0}^i n_i + i)$ th element in $\tilde{W}_7$ for $i \in \{0, 1, m-1\}$. We can therefore subtract all $\sum_{j=0}^i n_i$ values and obtain

$$\begin{aligned} \tilde{W}_8 = & (m, 0, 0) \dots (m, 0, n_0 - 1)(0, 0, 0)(m, 1, n_0) \\ & \dots (m, 1, n_0 + n_1 - 1)(1, 0, 0)(m, 2, n_0 + n_1) \\ & \dots (m, m-1, n-1)(m-1, 0, 0). \end{aligned} \quad (73)$$

Next, we can sort $\tilde{W}_8$ again under Γ_{W_2} to obtain

$$\begin{aligned} \tilde{W}_f = & (0, 0, 0) \dots (m-1, 0, 0)(m, 0, 0) \\ & \dots (m, 0, n_0-1)(m, 1, n_0) \dots (m, 1, n_0+n_1-1) \\ & \dots (m, m-1, n-1). \end{aligned} \quad (74)$$

We now have a list where the first m elements are just counting up to m , the first values of all remaining elements are m , and the third values of the remaining elements are all just the indices. All of this unnecessary information can be reversibly removed, yielding the desired output list

$$W_f = 0^{n_0} 1^{n_1} \dots (m-1)^{n_{m-1}}. \quad (75)$$

Since all operations in this process are either reversible sorting or reversible local data modification, the whole process including symmetrization takes $\tilde{O}(\log^3 n)$ depth and can be implemented as a reversible circuit.

Because of the reversibility of this algorithm, we can also transform a (superposition) of NSILs back to a second-quantized state by inverting the above procedure. Since our NSIL symmetrization algorithm is also reversible, combining both algorithms gives a mechanism to efficiently convert between first quantization and second quantization.

VI. CASE STUDY: EFFICIENT QUANTUM SIMULATION OF THE BOSE-HUBBARD MODEL

To showcase potential asymptotic advantage in quantum simulation using first quantization, we present a case study of the Bose-Hubbard model on a bounded-degree graph with m sites. We use d_s to denote the maximum degree of the graph. Since the Bose-Hubbard model is particle-number conserving, Hilbert spaces with different numbers of bosons are decoupled from each other. Therefore, in this section, we focus on the evolution within a specific n -particle subspace.

We start with a formal definition of the model. The adjacency matrix of graph G has $A(G)_{j,k} = 1$ if there is an edge between vertices j, k , and $A(G)_{j,k} = 0$ otherwise. In second quantization, the Bose-Hubbard model on G has the Hamiltonian

$$\begin{aligned} H_2 = & H_{\text{hop}}^{(2)} + H_{\text{int}}^{(2)} \\ := & J_{\text{hop}} \sum_{j,k=0}^{m-1} A(G)_{j,k} a_j^\dagger a_k + J_{\text{int}} \sum_{j=0}^{m-1} n_j(n_j-1), \end{aligned} \quad (76)$$

where $a_j, a_j^\dagger$ are ladder operators corresponding to mode j , $n_j = a_j^\dagger a_j$ is the number operator of mode j , and $J_{\text{hop}}, J_{\text{int}} \in \mathbb{R}$ are constants corresponding to hopping and interaction strengths, respectively.

In first quantization, we use n registers to store the site indices of n bosons. In this setting, the Hamiltonian can be written as [34]

$$\begin{aligned} H_1 = & H_{\text{hop}} + H_{\text{int}} \\ := & J_{\text{hop}} \sum_{i=1}^n A^{(i)}(G_{\text{sq}}) + J_{\text{int}} \sum_{j=0}^{m-1} n_j(n_j-1), \end{aligned} \quad (77)$$

where $A^{(i)}(G) := I^{\otimes i-1} \otimes A(G) \otimes I^{\otimes n-i}$ and $n_j = \sum_{i=1}^n |j\rangle\langle j|^{(i)}$.

Recall that for a model of n bosons in m modes, first and second quantization differ in the number of qubits used to store one wave function [$O(n \log m)$ and $O(m \log n)$ qubits, respectively]. Intuitively, one may expect that their corresponding quantum simulation algorithms have gate complexities $\text{poly}(n, \log m)$ and $\text{poly}(m, \log n)$, respectively. However, this is not straightforward. As we present in Sec. VI A, while a recently developed second-quantized algorithm can simulate the hopping part using $\text{poly}(m, \log n)$ gates, all quantum algorithms that we are aware of use $\text{poly}(n, m)$ gates to simulate the full Bose-Hubbard model. Using first quantization instead, we give a simulation of the hopping part using only $\text{poly}(n, \log m)$ gates in $\text{poly}(\log n, \log m)$ depth. We also develop novel algorithms using technical tools introduced in this paper to simulate the interaction part with $\text{poly}(n, \log m)$ gates and $\text{poly}(\log n, \log m)$ depth. Then, in Sec. VI C, we give an algorithm to simulate $e^{-iH_1 t}$ directly with overall gate complexity $\text{poly}(n, \log m, t)$. To the best of our knowledge, this is the only algorithm for simulating Bose-Hubbard models featuring logarithmic scaling with one of n, m .

A. Second-quantized simulation algorithms

We begin by briefly explaining why it is not straightforward to achieve $\text{poly}(m, \log n)$ gate complexity for a second-quantized simulation of the Bose-Hubbard model. In fact, the evolution of the hopping term and that of the interaction term can each be individually fast forwarded. Second-quantized simulation of the interaction term only requires a controlled phase gate because $e^{-iH_{\text{int}} t}$ is a diagonal unitary. For the hopping term, a recent breakthrough provides a second-quantized algorithm to simulate $e^{-iH_{\text{hop}}^{(2)} t}$ with $\text{poly}(m, \log n)$ gate complexity [35]. The algorithm relies on the observation that bosonic hopping operators $a_j^\dagger a_k$ act as the generators of $\mathfrak{su}(m)$ on fixed-particle-number subspaces and uses the quantum Hermite transform algorithm [36] as a key subroutine.

However, fast forwarding of individual terms does not directly enable fast-forwarding of their sum. The most straightforward approach to simulating $e^{-iH_2 t}$ using implementations of $e^{-iH_{\text{hop}}^{(2)} \tau}$ and $e^{-iH_{\text{int}}^{(2)} \tau}$ is Trotterization (i.e., product formulas). The state-of-the-art upper bound on

Trotter error has commutator scaling [37], i.e.,

$$\|e^{-i(H_{\text{hop}}^{(2)}+H_{\text{int}}^{(2)})t} - [S_p(t/r)]^r\| = O(\tilde{\alpha}_{\text{comm}} t^{p+1}/r^p), \quad (78)$$

where $S_p(t)$ is a p th-order product formula, r is the number of Trotter steps, and $\tilde{\alpha}_{\text{comm}}$ is a quantity determined by the norms of p th-order nested commutators. In our case,

$$\begin{aligned} \tilde{\alpha}_{\text{comm}} &= \sum_{\gamma_1, \dots, \gamma_{p+1} \in \{H_{\text{hop}}^{(2)}, H_{\text{int}}^{(2)}\}} \|[H_{\gamma_{p+1}}, \dots, [H_{\gamma_2}, H_{\gamma_1}]]\| \\ &= \text{poly}(n, m). \end{aligned} \quad (79)$$

Therefore, to achieve constant Trotter error, we need $r = O(\tilde{\alpha}_{\text{comm}}^{1/p} t^{(p+1)/p}) = \text{poly}(n, m)$. This implies that the overall gate complexity also scales polynomially with both n and m .

Another option is to use quantum signal processing [38]. However, even if we focus on the simplest example where $m = 2$ with a fixed particle number n , it is not straightforward to achieve $\text{poly}(\log n)$ gate complexity. In this case, the second-quantized encoding uses two registers storing non-negative integers, and any basis state is of the form $|n_0\rangle_0 |n - n_0\rangle_1$. One can further simplify the problem by defining $|n_0\rangle := |n_0\rangle_0 |n - n_0\rangle_1$, with

$$\begin{aligned} a_0^\dagger a_1 |n_0\rangle &= \sqrt{(n_0 + 1)(n - n_0)} |n_0 + 1\rangle, \\ a_1^\dagger a_0 |n_0\rangle &= \sqrt{n_0(n - n_0 + 1)} |n_0 - 1\rangle, \\ a_0^\dagger a_0 |n_0\rangle &= n_0 |n_0\rangle, \\ a_1^\dagger a_1 |n_0\rangle &= (n - n_0) |n_0\rangle. \end{aligned} \quad (80)$$

The Hamiltonian H_2 in second quantization can be written as

$$\begin{pmatrix} J_{\text{int}}(n^2 - n) & J_{\text{hop}}\sqrt{n} & 0 & \cdots \\ J_{\text{hop}}\sqrt{n} & J_{\text{int}}(n^2 - 3n + 2) & J_{\text{hop}}\sqrt{2(n-1)} & \cdots \\ 0 & J_{\text{hop}}\sqrt{2(n-1)} & J_{\text{int}}(n^2 - 5n + 8) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \quad (81)$$

Observe that H_2 is 3-sparse, and $\|H_2\|_{\text{max}} = O(n^2)$. The query complexity of the algorithm in Ref. [38] depends linearly on the product of the sparsity and the maximum-entry norm. Therefore, it uses $O(n^2)$ queries.

More generally, we are not aware of any second-quantized algorithm that achieves $\text{poly}(\log n)$ gate complexity for this two-mode example, or $\text{poly}(m, \log n)$ gate complexity for arbitrary m . Searching for better second-quantized algorithms for the Bose-Hubbard model is a natural direction for future work.

B. Low-depth individual-term simulations with optimal m scaling

1. First-quantized simulation of H_{hop}

Since H_{hop} acts independently on every boson, one can simulate it by simply simulating the evolution according to $A(G)$ for all bosons in parallel. We have $\|A(G)\|_{\text{max}} = \max_{i,j} |A(G)_{ij}| = 1$, and for a graph G of maximum degree d_s , $A(G)$ is d_s sparse. These observations suggest using sparse-Hamiltonian simulation algorithms [38,39]. We develop a preliminary first-quantized algorithm for Bose-Hubbard models using quantum signal processing (QSP), which can implement $e^{-i\tau H}$ for any d_s -sparse Hamiltonian $H \in \mathbb{C}^{m \times m}$ with error ϵ and success probability at least $1 - 2\epsilon$ using

$$O\left(d_s \tau \|H\|_{\text{max}} + \frac{\log(1/\epsilon)}{\log \log(1/\epsilon)}\right) \quad (82)$$

queries to certain oracles (described below) and $O(\log m)$ additional gates [38] (Theorem 3). Note that there could be simpler algorithms for simulating only the hopping term since it acts independently on each register. However, we first present a QSP-based algorithm for the hopping term alone as a warm-up for the simulation of $H_1 = H_{\text{hop}} + H_{\text{int}}$ in Sec. VI C.

There are two data access oracles to implement. The first oracle $\hat{O}_F$ acts as

$$\hat{O}_F |i\rangle |l\rangle |0\rangle = |i\rangle |l\rangle |j(i, l)\rangle, \quad (83)$$

where $j(i, l)$ is the column index of the l th nonzero element in the i th row of H . If there is an efficient ($\text{poly}(\log m)$ gate complexity) strategy to find any of the d_s neighbors of a given vertex in G , then $\hat{O}_F$ can be implemented efficiently. Note that $O(\log m)$ qubits and gates suffice for many widely used graphs in many-body physics, including regular lattices.

The other oracle $\hat{O}_H$ returns a specific entry of $A(G)$:

$$\hat{O}_H |i\rangle |j\rangle |0\rangle = |i\rangle |j\rangle |A(G)_{ij}\rangle. \quad (84)$$

This can be implemented by checking if i, j are neighbors, so it has the same cost as implementing $\hat{O}_F$. Assuming that cost is $O(\log m)$, the overall gate complexity of simulating the evolution of $A^{(i)}(G)$ for time τ is

$$O\left[\left(d_s \tau + \frac{\log(1/\epsilon)}{\log \log(1/\epsilon)}\right) \log m\right]. \quad (85)$$

Since the hopping term acts independently on each boson, one can naively run the same algorithm for all i s in parallel to simulate the whole hopping part. To ensure low overall error and failure probability, we can reduce the error and failure probability for the individual terms to ϵ/n and $2\epsilon/n$,

respectively. Therefore, the total gate complexity for the simulation with time τ and error ϵ is

$$O\left[\left(J_{\text{hop}}D\tau + \frac{\log(n/\epsilon)}{\log \log(n/\epsilon)}\right)n \log m\right], \quad (86)$$

and the circuit depth is

$$O\left[\left(J_{\text{hop}}d_s\tau + \frac{\log(n/\epsilon)}{\log \log(n/\epsilon)}\right)\log m\right], \quad (87)$$

which scales only logarithmically with both n and m .

2. First-quantized simulation of H_{int}

Next, we investigate the simulation of the interaction part, $e^{-i\tau H_{\text{int}}}$. Observe that every basis state in first quantization is an eigenstate of every n_j operator (just like in second quantization), which means that $e^{-i\tau H_{\text{int}}}$ is a diagonal unitary. Therefore, we simply need to count the number of particles at each site, apply the corresponding phase shift $e^{-iJ_{\text{int}}\tau n_j(n_j-1)}$ for every $n_j > 1$, and uncompute the count. For arbitrary τ , this process exactly implements $e^{-i\tau H_{\text{int}}}$.

In first quantization, the particle number counting procedure can be implemented by the quantum sorting algorithm and the duplicate detection algorithm introduced in Sec. IV B, a subroutine of the NSIL symmetrization algorithm. Recall that for any $l \in \text{NSIL}$, the duplicate detection algorithm gives $\text{dup}(l)$, an array of n integers. For every element that is repeated more than once in l , there is one corresponding element of $\text{dup}(l)$ recording the number of repetitions, while all other elements of $\text{dup}(l)$ are 0. Here we explain our algorithm by presenting an example where the input is the symmetrized state of some NSIL l , without normalization. We use two ancilla registers, each containing n integers of $\log m$ bits.

We first apply the reversible sorting algorithm and the duplicate detection algorithm to implement

$$\begin{aligned} & \sum_{\sigma \in R_l} |\sigma(l)\rangle_1 |00\dots 0\rangle_2 |00\dots 0\rangle_3 \\ & \mapsto |l\rangle_1 |\text{dup}(l)\rangle_2 \sum_{\sigma \in R_l} |\text{rec}(\sigma)\rangle_3. \end{aligned} \quad (88)$$

Next, we observe that

$$\sum_{j=0}^{m-1} n_j(n_j-1) = \sum_{i=1}^n \text{dup}(l)_i \cdot (\text{dup}(l)_i - 1). \quad (89)$$

Therefore, the global phase shift corresponding to the interaction part can be implemented by applying a phase shift of $-i\tau J_{\text{int}} \text{dup}(l)_i [\text{dup}(l)_i - 1]$ for all i in parallel:

$$\begin{aligned} & |l\rangle_1 |\text{dup}(l)\rangle_2 \sum_{\sigma' \in R_l} |\text{rec}(\sigma')\rangle_3 \\ & \mapsto |l\rangle_1 \bigotimes_{i=1}^n e^{-i\tau J_{\text{int}} \text{dup}(l)_i [\text{dup}(l)_i - 1]} |\text{dup}(l)_i\rangle_2 \sum_{\sigma \in R_l} |\text{rec}(\sigma)\rangle_3 \\ & = e^{-iJ_{\text{int}}\tau \sum_{j=0}^{m-1} n_j(n_j-1)} |l\rangle_1 |\text{dup}(l)\rangle_2 \sum_{\sigma \in R_l} |\text{rec}(\sigma)\rangle_3. \end{aligned} \quad (90)$$

Finally, by applying the inverse of the duplicate detection algorithm and UNSORT_{1,3}, one can implement the overall transformation:

$$\sum_{\sigma \in R_l} |\sigma(l)\rangle \mapsto e^{-iJ_{\text{int}}\tau \sum_{j=0}^{m-1} n_j(n_j-1)} \sum_{\sigma \in R_l} |\sigma(l)\rangle. \quad (91)$$

Since the whole process is unitary, it can be applied to arbitrary superpositions of l . Note that all subroutines involved are parallelized, so the total gate count is $O(n \text{poly}(\log m))$, and the circuit depth is only $\text{poly}(\log n, \log m)$. This algorithm has no error and always succeeds.

However, combining efficient simulations of H_{int} and H_{hop} does not necessarily give an efficient simulation of their sum H_2 . Indeed, if we simply use the first-order product formula to simulate $e^{-iH_2 t} \approx (e^{-iH_{\text{int}} t/r} e^{-iH_{\text{hop}} t/r})^r$, the overall gate complexity will be $\text{poly}(n, m, t)$ due to the commutator scaling of Trotter error [37]. Therefore, we give a different strategy to simulate the Bose-Hubbard model with cost $\text{poly}(n, \log m, t)$.

C. Full-QSP simulation with optimal m scaling

In this section, we describe how to simulate Bose-Hubbard models on bounded-degree graphs with $\text{poly}(n, \log m)$ overall gate complexity.

We use the QSP framework [38] to simulate the whole first-quantized Hamiltonian H_1 . Observe that the sparsity of H_1 is $nd_s + 1$, where nd_s nonzero entries come from the hopping term because H_{hop} is the sum of n single-particle hopping terms, and each particle has at most d_s allowed hopping transitions. The remaining nonzero entry comes from the diagonal interaction term. However, H_{int} contributes significantly to $\|H_1\|_{\text{max}}$, since its eigenvalue is $\propto \sum_j n_j(n_j-1)$, which is maximized when all particles are in the same mode. Therefore,

$$\|H\|_{\text{max}} = \max[|J_{\text{hop}}|, |J_{\text{int}}|n(n-1)], \quad (92)$$

which is independent of m and dominated by n^2 for large numbers of particles. Now, the query complexity of the full-QSP algorithm is

$$O\left(t|J_{\text{int}}|n^3 d_s + \frac{\log(1/\epsilon)}{\log \log(1/\epsilon)}\right) \quad (93)$$

for arbitrary time duration t , error ϵ , and success probability $1 - 2\epsilon$, with $O(n \log m)$ additional gates.

Again, to understand the gate complexity, we investigate the implementation of the two data-access oracles. Note that the basis state $|j_1 j_2 \dots j_n\rangle$ is linked to itself (the diagonal part) and all $|j_1 j_2 \dots j_{k-1} j'_k j_{k+1} \dots j_n\rangle$ with $\{j_k, j'_k\}$ being an edge in G . Therefore, to implement $\hat{O}_F$, it suffices to find and output the l th allowed $|j_1 j_2 \dots j_{k-1} j'_k j_{k+1} \dots j_n\rangle$ basis state. Finding the state is easy because for a given k , all j'_k s can be efficiently found, and the number of j'_k s that should be ahead of j_k can also be efficiently computed. However, the gate complexity of implementing $\hat{O}_F$ is still $O(n \log m)$ because the cost of outputting $n \log m$ bits dominates that of finding the state.

The oracle $\hat{O}_H$ takes two basis states as input and outputs the corresponding element of H_1 . When the two input basis states differ in the mode of exactly one boson, and the modes of that boson in the two states are neighbors in G , we output J_{hop} ; if they are not neighbors, or if more than two bosons have different modes, we output 0. When the inputs are the same, we need to evaluate $\sum_j n_j(n_j - 1)$. Just as in Sec. VI B and Eq. (89), we can again employ the duplicate detection algorithm to compute $\text{dup}(l)$ and compute the sum of all $\text{dup}(l)_i[\text{dup}(l)_i - 1]$, which takes $\tilde{O}(n \log n \log m)$ gates. However, unlike in the simulation of just the interaction term, where the phase operations can be easily parallelized, here we compute the sum explicitly and store it in the output register. This involves a summation algorithm, such as parallel reduction, which uses $O(n)$ additions of $O(\log n)$ -bit integers. Therefore, the total gate count for $\hat{O}_H$ is $\tilde{O}(n \log n \log m)$.

Now, we have all building blocks to compute the overall gate complexity for the simulation of e^{-iHt} with success probability $1 - 2\epsilon$ and error ϵ using QSP:

$$\tilde{O} \left[\left(t |J_{\text{int}}| n^3 + \frac{\log(1/\epsilon)}{\log \log(1/\epsilon)} \right) n \log n \log m \right]. \quad (94)$$

In the first-quantization-friendly scenario ($m \gg n$), the scaling with n , m , t is $O(tn^4 \log n \log m)$. This shows that first quantization enables a simulation with overall gate complexity $\text{poly}(n, \log m)$.

VII. APPLICATION: QUANTUM INTERFEROMETRIC IMAGING SYSTEM

A. QFT-based single-photon interferometry

We review quantum telescope arrays [12,13] for the low-photon-rate scenario with the help of the schematic diagram in Fig. 7. For simplicity, we work in an idealized model of a two-dimensional world where all m telescopes are equally spaced in a line. Each telescope $T_0, T_1, \dots, T_{m-1}$ has a lens (or mirror) to collect photons and a well-isolated photon detector equipped with a bosonic quantum memory (since

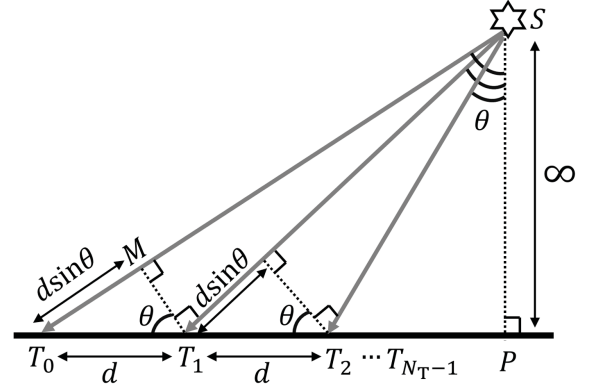


FIG. 7. In this idealized 2D model, since S is infinitely far away from the telescope array, $\theta = \angle T_0 T_1 M = \angle T_0 S P = \angle T_1 S P$. Therefore, the path difference between ST_j and ST_{j+1} is $d \sin \theta$.

photons are bosons). The distance between adjacent telescopes is d . The photon source (which may be a star, galaxy, etc.) is denoted by S and is assumed to be infinitely far away from the telescope array, so its position can be described using a single parameter θ , the angle between $T_0 S$ and SP where P is the nearest point to S on the telescope plane. Since both d and m are finite, the following angle relation can be derived from trigonometry:

$$\theta = \angle T_0 T_1 M = \angle T_0 S P = \angle T_1 S P = \dots = \angle T_{m-1} S P. \quad (95)$$

A photon emitted by S may finally be received by one of the m detectors, but the path length gone through by the photon varies for different detectors. For two adjacent detectors in the array, the path difference is $d \sin \theta$ by trigonometry, as shown in Fig. 7. The path difference leads to a phase difference, $2\pi d \sin \theta / \lambda$. If we let the phase at detector T_0 be 0, then the phase at detector T_j is $2\pi d j \sin \theta / \lambda$. Since the photon may arrive at any detector in the array, we can define the creation operator of the array $a^\dagger(\theta)$ with angle θ in terms of the creation operator of the memory of the j th detector, $b_j^\dagger$, for all $j \in \{0, 1, \dots, m-1\}$:

$$a^\dagger(\theta) = \sum_{j=0}^{m-1} \exp\left(i \frac{2\pi d j \sin \theta}{\lambda}\right) b_j^\dagger. \quad (96)$$

Note that $[b_i^\dagger, b_j^\dagger] = 0$, $[b_i, b_j^\dagger] = \delta_{ij}$, and $b_i^\dagger |v_i\rangle_i = \sqrt{v_i + 1} |v_i + 1\rangle_i$ since we are using a bosonic memory.

The initial state of the quantum memories is $|0\rangle_0 |0\rangle_1 \dots |0\rangle_{m-1}$, also written as $|00\dots 0\rangle$ for short. The state after a single photon detection is

$$a^\dagger(\theta) |00\dots 0\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} e^{i \frac{2\pi d j \sin \theta}{\lambda}} |0\rangle \otimes^{j-1} |1\rangle_j |0\rangle^{m-j}, \quad (97)$$

where the state is represented using second quantization. This is a natural representation because the photon detectors and their corresponding quantum memories are far from each other and the occupation numbers can be stored locally.

For most optical-wavelength astronomical observations, the number of photons received per time step is much less than 1. In this case, we need only be able to process the above state. Note that the only information we need from this state is the angle θ , since an image is essentially a photon distribution over angles. We notice that there is a periodicity in the phases depending on the θ , so the QFT is a natural tool to extract the information. To perform a QFT, we can transform the state into a first-quantized binary representation:

$$|0\rangle^{\otimes j-1}|1\rangle_j|0\rangle^{m-j} \mapsto |j\rangle, \quad (98)$$

where j is represented by up to $\log(m)$ qubits. To do so, we first bring the distributed quantum memories together, which can be done by either physically moving the qubits or using quantum repeaters. Next, we run a standard unary-to-binary encoder in $\log(m)$ depth. The transformed state is simply

$$|\psi(\theta)\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \exp\left(i \frac{2\pi d j \sin \theta}{\lambda}\right) |j\rangle. \quad (99)$$

We can now perform the QFT on this state to obtain

$$\frac{1}{m} \sum_{j=0}^{m-1} \sum_{k=0}^{m-1} \exp\left[2\pi i j \left(\frac{d \sin \theta}{\lambda} - \frac{k}{m}\right)\right] |k\rangle. \quad (100)$$

Observe that if

$$\frac{d \sin \theta}{\lambda} - \frac{k}{m} = v, \quad (101)$$

where v can be any integer, then the Fourier-transformed state is simply $|k\rangle$, where

$$k \left(\frac{\lambda}{md} \right) = \sin \theta - \frac{v\lambda}{d}. \quad (102)$$

The physical meaning of k is now straightforward: if the telescope array only receives photons from $\sin \theta = [(v\lambda)/d]$ to $\sin \theta = [(v+1)\lambda/d]$ (which can be enforced by the lens or mirror in front of the photon detector), then k is the difference between $\sin \theta$ and $v\lambda/d$ in terms of λ/md . Here md is the baseline, λ/d is the field of view, and λ/md is the angular resolution, which saturates the Rayleigh criterion. Since it is not necessary to physically bring photons together for interference, this method can be applied to longer baselines and shorter wavelengths

(infrared or optical) than classical interferometry, potentially leading to micro-arcsecond angular resolution. Furthermore, since this telescope array can handle single photon-events, for photon rate ϵ , it has sensitivity $O(\epsilon)$, much higher than the $O(\epsilon^2)$ sensitivity of intensity interferometers.

Note that in the discussions of quantum telescope arrays in this paper, we assume there are no atmospheric fluctuations that may produce local phase differences. This may be reasonable if the telescope array is space based, but for ground-based telescopes, such phases are inevitable and may affect image quality, since the phase difference between the i th and the j th telescopes is no longer strictly $2\pi d(j-i) \sin \theta / \lambda$. These fluctuations can potentially be corrected using classical methods such as adaptive optics [40], as mentioned in Ref. [41].

B. Multiple-photon interferometry

So far, we have only discussed the case where the detected state has the form $a^\dagger(\theta)|00\dots 0\rangle$. More generally, it is natural to consider multiple-excitation states of the form

$$|\psi(\theta_1, \dots, \theta_n)\rangle = \prod_{j=1}^n a^\dagger(\theta_j) |00\dots 0\rangle, \quad (103)$$

where n is the number of photons received by the telescope array in one time step, and θ_j is the angle of the j th photon. Here we assume the use of bosonic quantum memories to store multiple photons at the same time in a distributed telescope array coherently. Although multiphoton storage is more difficult than storing a single photon, it has been demonstrated on several physical platforms, including atomic quantum memory [42] and all-optical quantum memory [43]. Furthermore, controlled release of multiple photons has been demonstrated in a microwave cavity [44].

This multiple-photon scenario is not relevant to the most common situations in optical or infrared interferometry in astronomy. This is mainly because the signal from a distant astronomical object is very weak when it arrives at the telescope array. It is also related to the observed wavelengths. Typically, astronomical sources emit photons of all wavelengths. At radio or microwave wavelengths, since the energy per photon (hc/λ) is low, the number of photons per mode can be more than 1. However, very long baseline interferometry has been realized in these wavelengths, making quantum interferometry unnecessary. The more astronomically interesting regime is optical and infrared wavelengths, for which we design quantum telescopes arrays to perform interferometry. However, the photon rate in this case is usually much lower than 1 per mode due to the much higher energy per photon.

However, we claim that the photon rate can be much higher when one or more of the following conditions hold:

- (1) the number of detectors m is large;

- (2) the area of each aperture is large;
- (3) the field of view is large (which can be achieved when d is very small);
- (4) the photon wavelength is long; or
- (5) the source is bright.

These conditions might be met in more general imaging tasks beyond observational astronomy, so we call a quantum “telescope” array working under these conditions an interferometric imaging system. Note that even under these conditions, we could be in a regime where the number of photons is much less than the number of modes, making the first-quantized representation advantageous.

Now, we outline how our NSIL symmetrization algorithm makes it possible to extract all θ_j from $|\psi(\theta_1, \dots, \theta_n)\rangle$ using only one copy of the state using Result 2. Let

$$\phi_{i,j} := \frac{2\pi d j \sin \theta_i}{\lambda}. \quad (104)$$

The explicit multiple-photon state is

$$\begin{aligned} & \prod_{j=1}^n a^\dagger(\theta_j) |00\dots 0\rangle \\ & \propto \sum_{r \in [m]^n} \exp\left(i \sum_{i=1}^n \phi_{i,r_i}\right) b_{r_1}^\dagger b_{r_2}^\dagger \dots b_{r_n}^\dagger |00\dots 0\rangle, \end{aligned} \quad (105)$$

where r is a list of integers whose i th element r_i represents at which telescope the i th photon is detected. Now, letting n_i be the number of photons detected at telescope i in r ,

$$b_{r_1}^\dagger b_{r_2}^\dagger \dots b_{r_n}^\dagger |00\dots 0\rangle = \sqrt{\prod_{i=0}^{m-1} n_i!} |n_0 n_1 \dots n_{m-1}\rangle \quad (106)$$

due to the bosonic nature of the quantum memory. This is the second-quantized representation of the photon state.

As with the single-photon case, we convert to the first-quantized representation in order to apply the quantum Fourier transform. To do so, as described in Sec. V, we first transform $|n_0 n_1 \dots n_{m-1}\rangle$ into a list l of telescope indices in increasing order. This transformation circuit has $\tilde{O}(\log^2 n)$ depth and uses $O(n \log m)$ ancilla qubits. As a simple example, if $m = 3$, $n = 4$, and $r = 2101$, then the state in Eq. (106) is $|121\rangle$, which will be transformed to $|l(r)\rangle = |0112\rangle$. Here $l(r) = 0112$ is an NSIL as it is the sorted version of r . We also notice that the factor $\sqrt{\prod_{i=0}^{m-1} n_i!}$ is just $\sqrt{|H_{l(r)}|}$, square root of the order of the subgroup of S_n preserving $l(r)$. We can thus write the transformed un-normalized state as

$$\sum_{r \in [m]^n} \exp\left(i \sum_{i=1}^n \phi_{i,r_i}\right) \sqrt{|H_{l(r)}|} |l(r)\rangle. \quad (107)$$

Next, we can apply the NSIL symmetrization algorithm in Result 2 to obtain the un-normalized first-quantized state

$$\begin{aligned} & |\psi_S(\theta_1, \dots, \theta_n)\rangle \\ & = \sum_{r \in [m]^n} \exp\left(i \sum_{i=1}^n \phi_{i,r_i}\right) \sqrt{\frac{|H_{l(r)}|}{|S_{l(r)}|}} \sum_{\sigma' \in S_{l(r)}} |\sigma'(l(r))\rangle \\ & \propto \sum_{r \in [m]^n} \exp\left(i \sum_{i=1}^n \phi_{i,r_i}\right) |H_{l(r)}| \sum_{\sigma' \in S_{l(r)}} |\sigma'(l(r))\rangle, \end{aligned} \quad (108)$$

where we use $|S_n| = |S_{l(r)}| \cdot |H_{l(r)}|$ in the last line.

To make it easier to analyze this state, we rewrite it as a superposition over $r \in [m]^n$, i.e.,

$$|\psi_S(\theta_1, \dots, \theta_n)\rangle \propto \sum_{r \in [m]^n} \beta_r |r\rangle, \quad (109)$$

where β_r is the coefficient corresponding to $|r\rangle$. We notice that if r' satisfies $l(r') = l(r)$, then its phase in Eq. (108) contributes to β_r , and it can be related to r' by $r' = \sigma'(r)$, where $\sigma' \in S_r$. Therefore,

$$\begin{aligned} \beta_r & = |H_{l(r)}| \sum_{\sigma \in S_r} \exp\left(i \sum_{i=1}^n \phi_{i,\sigma'(r)_i}\right) \\ & = \sum_{\sigma \in S_n} \exp\left(i \sum_{i=1}^n \phi_{i,\sigma(r)_i}\right), \end{aligned} \quad (110)$$

where in the second line we replace the summation over S_r by that over S_n and use the degeneracy in S_n when acting on r , i.e., $|H_{l(r)}| \cdot |S_r| = |S_n|$. We also observe that the sum is preserved under a permutation of photon indices, so

$$\begin{aligned} \beta_r & = \sum_{\sigma \in S_n} \exp\left(i \sum_{i=1}^n \phi_{\sigma^{-1}(12\dots n)_i, \sigma^{-1} \circ \sigma(r)_i}\right) \\ & = \sum_{\sigma \in S_n} \exp\left(i \sum_{i=1}^n \phi_{\sigma(12\dots n)_i, r_i}\right), \end{aligned} \quad (111)$$

where in the last line we replace $\phi_{\sigma^{-1}(12\dots n)_i, r_i}$ by $\phi_{\sigma(12\dots n)_i, r_i}$ because σ^{-1} is unique for each σ and $\sigma^{-1} \in S_n$.

Finally, we rewrite

$$\begin{aligned} |\psi_S(\theta_1, \dots, \theta_n)\rangle & \propto \sum_{r \in [m]^n} \sum_{\sigma \in S_n} \exp\left(i \sum_{i=1}^n \phi_{\sigma(12\dots n)_i, r_i}\right) |r\rangle \\ & = \sum_{\sigma \in S_n} \sum_{r \in [m]^n} \exp\left(i \sum_{i=1}^n \phi_{\sigma(12\dots n)_i, r_i}\right) |r\rangle \\ & = \sum_{\sigma \in S_n} \bigotimes_{i=1}^n \left(\sum_{r_i \in [m]} \exp(i \phi_{\sigma(12\dots n)_i, r_i}) |r_i\rangle \right). \end{aligned} \quad (112)$$

All θ_i can be extracted by running the QFT on all n registers in parallel and measuring in the computational basis:

$$\text{QFT}^n |\psi_S(\theta_1, \dots, \theta_n)\rangle = \frac{1}{\sqrt{n!}} \sum_{\sigma \in S_n} \bigotimes_{i=1}^n |k_{\sigma(12\dots n)_i}\rangle, \quad (113)$$

where $k_{\sigma(12\dots n)_i} = [(md)/\lambda] \{\sin \theta_{\sigma(12\dots n)_i} - [(v\lambda)/d]\}$ with $v \in \mathbb{Z}$ determined by the direction of the telescope, as in Eq. (102).

Thus, with the help of the NSIL symmetrization algorithm, a single-photon quantum telescope array can be upgraded to an interferometric imaging system capable of processing multiple photons at a time. The quantum computation for n photons on m detectors has depth $\text{poly}(\log n)$ and uses $O(m \log n + n \log m)$ qubits. Capturing multiple photons may allow imaging systems to work for more general purposes, including high-resolution observation of the sun and astronomical surveys. Our scheme may also be particularly useful in some specific wavelengths, such as the far infrared, where the photon rate is close to 1 but classical very long baseline interferometry is not practical.

Another potential application in the far future is photography in dark conditions. Note that to achieve field of view λ/d with angular resolution $\lambda/(Nd)$, it suffices to have an $N \times N$ array of low-resolution lenses or mirrors with aperture radii much shorter than d , instead of a single lens or mirror with aperture Nd . Manufacturing large-aperture lenses is difficult and expensive even for ordinary photography, which can be avoided with our imaging system at the cost of a quantum information processor and well-isolated photon detectors.

It is also worth mentioning that instead of directly producing images, quantum telescope arrays can offer better accuracy in estimating specific parameters, such as the distance between binary stars, as introduced in Ref. [45]. Our multiple-photon interferometric imaging system can potentially extend the list of high-precision parameter estimations.

VIII. SUMMARY AND DISCUSSION

In this paper, we studied the general quantum symmetrization problem. We identified subtleties with repetitions in the input list and pointed out issues with previous SIL symmetrization algorithms when acting on NSILs. Using sorting networks, we then proposed the first logarithmic-depth quantum algorithms for symmetrizing NSILs. These algorithms leverage the coset decomposition of the symmetric group, use an ancilla register to mediate multiplications of permutations elements, and adapt SIL symmetrization algorithms as subroutines. The single-input symmetrization algorithm features $\tilde{O}(\log n)$ depth and uses $O(n \log n + \log m)$ ancilla qubits for NSILs of n integers with value up to m , while the superposed-input

symmetrization algorithm uses $\tilde{O}(\log^3 n)$ depth and $O(n \log n)$ ancilla qubits. We also proposed an $\tilde{O}(\log^2 n)$ -depth algorithm to transform between second-quantized states and NSILs, which allows for efficient conversion between second-quantized states and first-quantized states.

Our algorithms finally enable first-quantized simulation of bosons by giving a mechanism for preparing symmetrized initial states. As an example, we introduced quantum algorithms for simulating Bose-Hubbard models using $\text{poly}(n, \log m)$ qubits and gates. The NSIL symmetrization algorithm can be used to prepare (superpositions of) Dicke states of any Hamming weight in logarithmic depth using a reasonable number of ancilla qubits. The efficient conversion algorithm between first quantization and second quantization provides a mechanism for processing multiple photons at a time in the quantum telescope array proposal.

We distinguish between the scenario in our paper and in other algorithmic applications of symmetrization. It is well known that the graph isomorphism problem can be solved by creating a superposition over all permutations of a given graph [46]. However, our results cannot be used in that context since there is no increasing input guarantee in the setting of graph isomorphism. Indeed, if the vertices of a graph could be ordered in a canonical way, this would solve graph isomorphism.

Our algorithm requires all-to-all connectivity due to the sorting networks used, just as in Ref. [4]. A natural open problem is to find low-depth sorting networks with restricted connectivity, which could make our algorithm much more favorable for near-term devices. It might also be possible to reduce the circuit depth of the superposed-input symmetrization algorithm in Result 2 or increase the success probability of the algorithm in Result 1. In addition, we observe that any comparison-based sorting algorithm can fit into our symmetrization framework. Therefore, in practice, if n is finite and the overhead of implementing parallelized all-to-all sorting networks dominates the circuit depth, one can also employ bubble sort or odd-even transposition network [47] to perform symmetrization. Both of them have $O(n)$ circuit depth (which is optimal because swapping items at the beginning and the end of the list requires $n - 1$ steps), but requires only 1D connectivity and still have $\text{poly}(n)$ gate complexity. Another possible architecture of n registers is a $\sqrt{n} \times \sqrt{n}$ 2D square lattice where comparators are only available between nearest-neighbor registers. In this case, mesh sorting networks [48,49] enable instantiating our symmetrization algorithms in $\tilde{O}(\sqrt{n})$ depth (which is also provably optimal [50]).

Another potential follow-up direction is finding more examples in which first-quantized simulation of bosons has asymptotically favorable resource scaling. This includes designing new algorithms and discovering new applications suitable for the first-quantized encoding. Also, the algorithm for Bose-Hubbard models proposed in this paper

has n^4 scaling with the number of particles, which is unlikely optimal and might be further improved while preserving the $\text{poly}(\log m)$ scaling with the number of modes.

ACKNOWLEDGMENTS

We thank Alexey Gorshkov, Emil Khabiboulline, Cheng-Ju Lin, and Yuan Su for helpful discussions. We acknowledge support from the National Science Foundation (QLCI Grant No. OMA-2120757).

DATA AVAILABILITY

There are no publicly available research data or software supporting this manuscript. Requests for further information or data should be sent to the authors.

-
- [1] R. P. Feynman, *Simulating physics with computers*, *Int. J. Theor. Phys.* **21**, 467 (1982).
 - [2] D. S. Abrams and S. Lloyd, *Simulation of many-body Fermi systems on a universal quantum computer*, *Phys. Rev. Lett.* **79**, 2586 (1997).
 - [3] N. J. Ward, I. Kassal, and A. Aspuru-Guzik, *Preparation of many-body states for quantum simulation*, *J. Chem. Phys.* **130** (2009).
 - [4] D. W. Berry, M. Kieferová, A. Scherer, Y. R. Sanders, G. H. Low, N. Wiebe, C. Gidney, and R. Babbush, *Improved techniques for preparing eigenstates of fermionic Hamiltonians*, *npj Quantum Inf.* **4**, 22 (2018).
 - [5] Y. Su, D. W. Berry, N. Wiebe, N. Rubin, and R. Babbush, *Fault-tolerant quantum simulations of chemistry in first quantization*, *PRX Quantum* **2**, 040332 (2021).
 - [6] H. H. S. Chan, R. Meister, T. Jones, D. P. Tew, and S. C. Benjamin, *Grid-based methods for chemistry simulations on a quantum computer*, *Sci. Adv.* **9**, eabo7484 (2023).
 - [7] T. Kosugi, H. Nishi, and Y.-i. Matsushita, *First-quantized eigensolver for ground and excited states of electrons under a uniform magnetic field*, *Jpn. J. Appl. Phys.* **62**, 062004 (2023).
 - [8] T. N. Georges, M. Bothe, C. Sünderhauf, B. K. Berntson, R. Izsák, and A. V. Ivanov, *Quantum simulations of chemistry in first quantization with any basis set*, *npj Quantum Inf.* **11**, 55 (2025).
 - [9] R. I. Nepomechie and D. Raveh, *Qudit Dicke state preparation*, *Quantum Inf. Comput.* **24**, 0037 (2024).
 - [10] We are aware of concurrent work that also gives an algorithm for preparing Dicke states with polylogarithmic depth but uses only polylogarithmically many ancillas [11]. The approaches are very different: while we use sorting networks, the other approach applies a simple sequence of collective rotations and parity measurements. Our approach solves a more general symmetrization problem, while the other approach is simpler, likely performs better in practice, and can perform better when the Hamming weight is lower.
 - [11] J. Yu, S. R. Muleady, Y.-X. Wang, N. Schine, A. V. Gorshkov, and A. M. Childs, *Efficient preparation of Dicke states*, *Phys. Rev. Lett.* **136**, 030601 (2026).
 - [12] D. Gottesman, T. Jennewein, and S. Croke, *Longer-baseline telescopes using quantum repeaters*, *Phys. Rev. Lett.* **109**, 070503 (2012).
 - [13] E. T. Khabiboulline, J. Borregaard, K. De Greve, and M. D. Lukin, *Quantum-assisted telescope arrays*, *Phys. Rev. A* **100**, 022316 (2019).
 - [14] L. Alonso and R. Schott, *A parallel algorithm for the generation of a permutation and applications*, *Theor. Comput. Sci.* **159**, 15 (1996).
 - [15] D. W. Berry, N. C. Rubin, A. O. Elnabawy, G. Ahlers, A. E. DePrince III, J. Lee, C. Gogolin, and R. Babbush, *Quantum simulation of realistic materials in first quantization using non-local pseudopotentials*, *npj Quantum Inf.* **10**, 130 (2024).
 - [16] P. Mukhopadhyay, T. F. Stetina, and N. Wiebe, *Quantum simulation of the first-quantized Pauli-Fierz Hamiltonian*, *PRX Quantum* **5**, 010345 (2024).
 - [17] Y. Tong, V. V. Albert, J. R. McClean, J. Preskill, and Y. Su, *Provably accurate simulation of gauge theories and bosonic systems*, *Quantum* **6**, 816 (2022).
 - [18] T. Kuwahara, T. V. Vu, and K. Saito, *Effective light cone and digital quantum simulation of interacting bosons*, *Nat. Commun.* **15**, 2520 (2024).
 - [19] J. D. Watson, J. Bringewatt, A. F. Shaw, A. M. Childs, A. V. Gorshkov, and Z. Davoudi, *Quantum algorithms for simulating nuclear effective field theories*, [arXiv:2312.05344](https://arxiv.org/abs/2312.05344).
 - [20] R. H. Dicke, *Coherence in spontaneous radiation processes*, *Phys. Rev.* **93**, 99 (1954).
 - [21] R. Prevedel, G. Cronenberg, M. S. Tame, M. Paternostro, P. Walther, M.-S. Kim, and A. Zeilinger, *Experimental realization of Dicke states of up to six qubits for multi-party quantum networking*, *Phys. Rev. Lett.* **103**, 020503 (2009).
 - [22] G. Tóth, *Multipartite entanglement and high-precision metrology*, *Phys. Rev. A* **85**, 022322 (2012).
 - [23] Y. Ouyang, *Permutation-invariant quantum codes*, *Phys. Rev. A* **90**, 062317 (2014).
 - [24] S. Hadfield, Z. Wang, B. O’Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, *From the quantum approximate optimization algorithm to a quantum alternating operator ansatz*, *Algorithms Mol. Biol.* **12**, 34 (2019).
 - [25] A. Bäertschi and S. Eidenbenz, *Deterministic preparation of Dicke states*, in *Proceedings of the 22nd International Symposium on Fundamentals of Computation Theory* (Springer, Cham, 2019), pp. 126–139, [10.1007/978-3-030-25027-0_9](https://doi.org/10.1007/978-3-030-25027-0_9).
 - [26] A. Bäertschi and S. Eidenbenz, *Short-depth circuits for Dicke state preparation*, in *Proceedings of the 2022 IEEE International Conference on Quantum Computing and Engineering* (IEEE, Piscataway, NJ, 2022), pp. 87–96, [10.1109/QCE53715.2022.00027](https://doi.org/10.1109/QCE53715.2022.00027).
 - [27] H. Buhrman, M. Folkertsma, B. Loff, and N. M. Neumann, *State preparation by shallow circuits using feed forward*, *Quantum* **8**, 1552 (2024).
 - [28] L. Piroli, G. Styliaris, and J. I. Cirac, *Approximating many-body quantum states with quantum circuits and measurements*, *Phys. Rev. Lett.* **133**, 230401 (2024).
 - [29] C.-J. Lin, Z.-W. Liu, V. V. Albert, and A. V. Gorshkov, *Covariant quantum error-correcting codes with metrological entanglement advantage*, *Phys. Rev. Lett.* **135**, 110801 (2025).

- [30] K. E. Batchner, *Sorting networks and their applications*, in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (Association for Computing Machinery, New York, 1968), pp. 307–314, [10.1145/1468075.1468121](#).
- [31] H. S. Stone, *Parallel processing with the perfect shuffle*, *IEEE Trans. Comput.* **100**, 153 (1971).
- [32] M. Ajtai, J. Komlós, and E. Szemerédi, *An $o(n \log n)$ sorting network*, in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 1983), pp. 1–9, [10.1145/800061.808726](#).
- [33] W. D. Hillis and G. L. Steele Jr., *Data parallel algorithms*, *Commun. ACM* **29**, 1170 (1986).
- [34] A. M. Childs, D. Gosset, and Z. Webb, *The Bose-Hubbard model is QMA-complete*, in *International Colloquium on Automata, Languages, and Programming* (Springer, New York, 2014), pp. 308–319, [10.1007/978-3-662-43948-7_26](#).
- [35] V. Iyer, S. Jain, S. Jordan, and R. Somma, *Efficient quantum circuits for high-dimensional representations of $SU(n)$ and Ramanujan quantum expanders*, [arXiv:2602.15180](#).
- [36] S. Jain, V. Iyer, R. D. Somma, N. Bao, and S. Jordan, *Efficient quantum Hermite transform*, in *Proceedings of the 58th Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2026), pp. 541–552, [10.1145/3798129.3800772](#).
- [37] A. M. Childs, Y. Su, M. C. Tran, N. Wiebe, and S. Zhu, *Theory of Trotter error with commutator scaling*, *Phys. Rev. X* **11**, 011020 (2021).
- [38] G. H. Low and I. L. Chuang, *Optimal Hamiltonian simulation by quantum signal processing*, *Phys. Rev. Lett.* **118**, 010501 (2017).
- [39] G. H. Low, *Hamiltonian simulation with nearly optimal dependence on spectral norm*, in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2019), pp. 491–502, [10.1145/3313276.3316386](#).
- [40] J. M. Beckers, *Adaptive optics for astronomy-principles, performance, and applications*, *Annu. Rev. Astron. Astrophys.* **31**, 13 (1993).
- [41] E. T. Khabiboulline, J. Borregaard, K. De Greve, and M. D. Lukin, *Optical interferometry with quantum networks*, *Phys. Rev. Lett.* **123**, 070504 (2019).
- [42] W. Zhang, M.-X. Dong, D.-S. Ding, S. Shi, K. Wang, Z.-Y. Zhou, G.-C. Guo, and B.-S. Shi, *Interfacing a two-photon noon state with an atomic quantum memory*, *Phys. Rev. A* **98**, 063820 (2018).
- [43] M. Bouillard, G. Boucher, J. Ferrer Ortas, B. Pointard, and R. Tualle-Broui, *Quantum storage of single-photon and two-photon Fock states with an all-optical quantum memory*, *Phys. Rev. Lett.* **122**, 210501 (2019).
- [44] W. Pfaff, C. J. Axline, L. D. Burkhardt, U. Vool, P. Reinhold, L. Frunzio, L. Jiang, M. H. Devoret, and R. J. Schoelkopf, *Controlled release of multiphoton quantum states from a microwave cavity memory*, *Nat. Phys.* **13**, 882 (2017).
- [45] A. Sajjad, M. R. Grace, and S. Guha, *Quantum limits of parameter estimation in long-baseline imaging*, *Phys. Rev. Res.* **6**, 013212 (2024).
- [46] A. Ambainis, L. Magnin, M. Roetteler, and J. Roland, *Symmetry-assisted adversaries for quantum state generation*, in *Proceedings of the 26th Annual Conference on Computational Complexity* (IEEE Computer Society, Washington, DC, 2011), pp. 167–177, [10.1109/CCC.2011.24](#).
- [47] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching* (Addison-Wesley Professional, Reading, MA, 1998), Vol. 3.
- [48] C. D. Thompson and H. T. Kung, *Sorting on a mesh-connected parallel computer*, *Commun. ACM* **20**, 263 (1977).
- [49] I. D. Scherson, S. Sen, and Y. Ma, *Two nearly optimal sorting algorithms for mesh-connected processor arrays using shear-sort*, *J. Parallel Distrib. Comput.* **6**, 151 (1989).
- [50] Y. Ma, S. Sen, and I. D. Scherson, *The distance bound for sorting on mesh-connected processor arrays is tight*, in *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)* (IEEE, Washington, DC, 1986), pp. 255–263, [10.1109/SFCS.1986.54](#).