Bosehedral: Compiler Optimization for Bosonic Quantum Computing

Junyu Zhou*, Yuhao Liu*, Yunong Shi†, Ali Javadi-Abhari‡ and Gushu Li*
*Department of Computer and Information Science, University of Pennsylvania, Philadelphia, USA
Email: {junyuzh,liuyuhao,gushuli}@seas.upenn.edu
†AWS Quantum Technology, New York, USA, Email: shiyunon@amazon.com
‡IBM Quantum, New York, USA, Email: Ali.Javadi@ibm.com

***Abstract*—Bosonic quantum computing, based on the infinite-dimensional qumodes, has shown promise for various practical applications that are classically hard. However, the lack of compiler optimizations has hindered its full potential. This paper introduces Bosehedral, an efficient compiler optimization framework for (Gaussian) Boson sampling on Bosonic quantum hardware. Bosehedral overcomes the challenge of handling infinite-dimensional qumode gate matrices by performing all its program analysis and optimizations at a higher algorithmic level, using a compact unitary matrix representation. It optimizes qumode gate decomposition and logical-to-physical qumode mapping, and introduces a tunable probabilistic gate dropout method. Overall, Bosehedral significantly improves the performance by accurately approximating the original program with much fewer gates. Our evaluation shows that Bosehedral can largely reduce the program size but still maintain a high approximation fidelity, which can translate to significant end-to-end application performance improvement.**

***Index Terms*—Bosonic Quantum Computing, Gaussian Boson Sampling, Compiler Optimization**

I. Introduction

Bosonic quantum computing, also known as continuous-variable quantum computing [31], is built upon Bosonic modes. Contrary to qubit-based discrete variable quantum computing, the basic information processing unit in Bosonic quantum computing, the qumode, by itself has an infinite-dimensional state space (shown in Fig. 1). Bosonic quantum computing (QC) is attractive for multiple reasons: long lifetimes of qumodes (e.g., superconducting cavities [49]), the ability to transduce between stationary and flying (e.g., photonic) information [37], etc. In particular, Bosonic QC has strong built-in information encoding and processing capability to naturally and efficiently encode certain computations, such as Boson sampling [3] and Gaussian Boson Sampling [20], with various practical applications [11] (e.g., graph clique [8], graph similarity [42], point process [22], and molecule vibrational spectra simulation [21]) that are hard for classical computing. Recently, the quantum advantage [34, 57, 58] has been experimentally demonstrated on several Bosonic QC platforms. Several startups are pursuing the commercialization of Bosonic QC with various technologies [1, 24, 35, 45].

Despite the great hardware progress, the development of software and compiler optimizations for Bosonic QC is far behind. Early efforts on programming and compilation for Bosonic QC, including Strawberry Fields [24], Bosonic Qiskit [44], and Perceval [35], provide basic programming interfaces and operation decomposition functions [15, 41] but miss program optimizations. To summarize, compilation for Bosonic QC is in its infancy. In contrast to qubit-based QC, rich libraries of compilation passes [39, 43] are non-existent, hindering the full exploitation of these computing platforms.

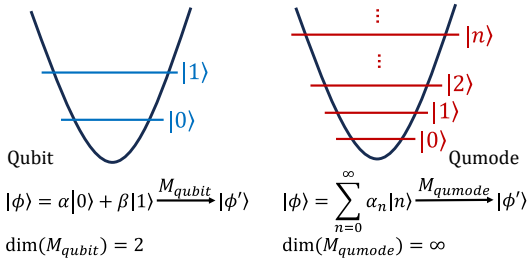

Fig. 1. Qubit vs Qumode

The complexity of devising compiler optimizations for Bosonic QC arises from the inherent infinite-dimensional state space of each individual qumode [31]. All the gates manipulating the state of qumodes, even for a single qumode, have infinite-dimensional gate matrices. Thus, it is highly non-trivial for the quantum compiler that runs on a classical computer to derive equivalent program transformations for optimization. Traditional techniques employed for quantum program transformations on qubit-based devices, which rely on gate matrices, encounter substantial obstacles when confronting the infinite-dimensional qumode gate matrices.

In this paper, we tackle this grand challenge by investigating the compiler optimization in the **high-level semantics** of Bosonic QC. Instead of directly handling the infinite-dimensional gate matrices applied on the qumode state, the unique high-level semantics of Bosonic QC allow us to analyze and optimize the program components efficiently and effectively in a compact data structure. In particular, we focus on the linear interferometer, the pivotal component containing most of the gates in a (Gaussian) Boson sampling program (a widely used Bosonic QC paradigm illustrated in Fig. 2). A linear interferometer can be considered as a unitary matrix of size $N \times N$ for an N-qumode program without losing any algorithmic information. All the program transformation and optimization techniques proposed in this paper can be reasoned about in the high-level unitary matrix.

To this end, we propose Bosehedral, an effective and efficient compiler optimization framework for (Gaussian) Boson

We thank the anonymous reviewers and shepherd for their constructive feedback and guidance. We thank Nathan Wiebe and Steven Girvin for insightful discussions. This work was partially supported by NSF CAREER Award 2338773, and the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under contract number DE-SC0012704.

sampling on Bosonic quantum hardware. In contrast to peephole approximations performed by qubit-based compilers [39, 43], Bosehedral can approximate and simplify circuits at a large scale by exploiting the global program semantics. **First**, Bosehedral optimizes the *qumode gate decomposition* for linear interferometers, which can be considered as adjusting the elimination patterns for different input high-level unitary matrices to maximize the occurrence of two-qumode gates with very small rotation angles. These small-angle gates are functionally akin to the identity and can be safely disregarded to reduce the gate error with minimal effect on the overall program semantics. **Second**, Bosehedral modifies the *logical-to-physical qumode mapping*, which can be considered as applying row and column permutations on the high-level unitary. Bosehedral can find better qumode mapping to further reduce rotation angles in the compiled two-qumode gates. The remapping is implemented via re-labeling the physical qumodes without any execution overhead. **Third**, Bosehedral comes with a *tunable probabilistic gate dropout* method to approximate linear interferometers. Gates with exceedingly small rotation angles will very likely be dropped, while gates with rotation angles near the threshold will be dropped probabilistically to average over the algorithmic approximation errors. As a result, Bosehedral can approximate (Gaussian) Boson sampling accurately using considerably fewer gates and thus substantially enhances the overall performance by largely mitigating the hardware error effects.

Overall approximation effect reasoning One key advantage of Bosehedral is that it can easily reason about the overall effect of the approximation during compilation time. After the gate decomposition and dropout, the global high-level semantics of the approximated program can be reconstructed by reversing the elimination process on the high-level unitary matrix. This is hard to achieve in the qubit-based approximation compilations because they usually rely on the unscalable low-level gate semantics and the gate matrices are exponentially large as the number of qubits increases.

Our experimental results show that Bosehedral can reduce gate by $\sim 25\%$ to 40% but maintain program fidelity of $\sim 98\%$ to 99.9% for various benchmarks and underlying architectures. Compared with the baseline Strawberry Fields [24], the divergence between the sampled output distribution in noisy simulation and the ideal output distribution is reduced by 26.1% on average, which translates to significant end-to-end application performance improvement as demonstrated in our detailed application studies.

Our major contributions can be summarized as follows:

1) We proposed Bosehedral, the first efficient and effective compiler optimization framework for (Gaussian) Boson sampling in Bosonic quantum computing.
2) Bosehedral overcomes the challenges of infinite-dimensional gate matrices by performing program analysis and compiler optimization at a high-level representation.
3) We proposed several compiler optimization algorithms for qumode gate decomposition, logical-to-physical qumode mapping, and probabilistic gate dropout for program simplification.
4) Our evaluation shows that Bosehedral can outperforms baseline Bosonic quantum compilers by significantly improving the execution fidelity and the end-to-end application performance for various benchmarks and architectures.

II. Bosonic Quantum Computing

This section introduces the necessary background on Bosonic quantum computing. We recommend [31, 50] for more comprehensive introductions.

A. Infinite-Dimensional Quantum Information Processing

In discrete variable quantum computing, the basic information processing unit is a qubit, whose state lies in a two-dimensional Hilbert space spanned by two basis states $|0\rangle$ and $|1\rangle$. In contrast, in Bosonic quantum computing, also known as the continuous-variable quantum computing, the basic information processing unit is a quantum mode or qumode. Its state lies in infinite-dimensional Hilbert space spanned by the Fock basis: $\{|n\rangle\}_{n=0}^{\infty}$. In literature, this infinite-level system is usually called a Bosonic mode and the corresponding quantum computing paradigm is called the Bosonic quantum computing [31, 50].

To understand how the states of Bosonic qumodes are manipulated, we first introduce the ladder operators, including the annihilation operator and creation operator, in the system state Hilbert space. For the k-th qumode, the annihilation operator $\hat{a}_k$ and the creation operator $\hat{a}_k^\dagger$ are defined as follows:

$$\begin{aligned} \hat{a}_k|n\rangle_k &= \sqrt{n}|n-1\rangle_k \qquad \text{for } n \geq 1 \\ \hat{a}_k^\dagger|n\rangle_k &= \sqrt{n+1}|n+1\rangle_k \qquad \text{for } n \geq 0 \end{aligned}$$

and $\hat{a}_k|0\rangle = 0$. For an N-qumode system, we have the following operator vectors:

$$\hat{a} = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_N)^\intercal,\ \hat{a}^\dagger = (\hat{a}_1^\dagger, \hat{a}_2^\dagger, \ldots, \hat{a}_N^\dagger)^\intercal$$

Qumode gates are usually defined with the $\hat{a}_k$'s and $\hat{a}_k^\dagger$'s. Some common qumode gates used in this paper are listed in Fig. 2 and introduced in the following. **Squeezing Gate** is a single-qumode gate denoted by 'S'. A squeezing gate applied on the k-th qumode is defined as: $S(\alpha) = exp\left(\frac{1}{2}(\alpha^*\hat{a}_k^2 - \alpha\hat{a}_k^{\dagger 2})\right)$, where $\alpha \in \mathbb{C}$. **Phase Shifter** is a single-qumode gate denoted by 'R'. A phase shifter applied on the k-th qumode is defined as: $R(\phi) = exp(i\phi\hat{a}_k^\dagger\hat{a}_k)$, where $\phi \in \mathbb{R}$. **Beamsplitter** is a two-qumode gate denoted by 'BS'. A Beamsplitter applied on the k-th and l-th qumodes is defined as: $BS(\theta, \phi) = exp\left(\theta(e^{i\phi}\hat{a}_k\hat{a}_l^\dagger - e^{-i\phi}\hat{a}_k^\dagger\hat{a}_l)\right)$, where ϕ and $\theta \in \mathbb{R}$. **Displacement** is a single-qumode gate denoted by 'D'. A displacement gate applied on the k-th qumode is defined as: $D(\alpha) = exp(\alpha\hat{a}_k^\dagger - \alpha^*\hat{a}_k)$, where $\alpha \in \mathbb{C}$. Note that all these gates have *infinitely-large gate matrices* as all $\hat{a}_k$s and $\hat{a}_k^\dagger$s are infinite-dimensional.

Compared with Finite-Dimensional Quantum Information Processing Qumodes are very different from three-

level qutrits or general d-level qudits because their infinite-dimensional nature leads to fundamentally different algorithm formulations. For qutrits and qudits, even if they have higher dimensions, the information is still encoded on the amplitudes of each discrete basis state. For example, we have the following qudit state:

$$|qudit\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle$$

The information is encoded in the α_i's. Thus the qutrits and qudits, together with qubits, still belong to the conventional discrete-variable quantum computing. While for qumodes, the corresponding algorithms are designed by the principle of *continuous-variable quantum computing*, a computing paradigm different from discrete-variable quantum computing. The information is encoded in a continuous function on another set of basis states, usually called the position/momentum basis. For example, a qumode state can be expressed either in the Fock basis or the position basis:

$$|qumode\rangle = \sum_{i=0}^{\infty} \alpha_i |i\rangle = \int_{-\infty}^{\infty} \psi(x) |x\rangle \, dx, i \in \mathbb{N}, x \in \mathbb{R}$$

The $\psi(x)$ encoding the algorithmic information is a continuous function where $x \in \mathbb{R}$ is a real number representing the position of the qumode's corresponding harmonic oscillator. The qumode gates are defined to manipulate the continuous functions $\psi(x)$'s rather than the amplitudes on the discrete basis states. This inherent difference makes it difficult to adapt the existing compilation approaches for photonic [56] or higher-dimensional systems like qutrits [19] which are designed for discrete-variable quantum computing.

B. Current State of Bosonic Quantum Computing

Bosonic quantum computing has attracted much attention recently due to its theoretical success and broad applications, as well as the high performance (such as low decoherence) of its processing unit: qumode. On the hardware side, Bosonic quantum computing is being pursued with mostly two different technologies, the superconducting Bosonic quantum processor by Quantum Inc. [1], and the photonics quantum processor by Xanadu [24], Quandela [35], and QuiX Quantum [45]. The errors of Bosonic quantum hardware are also different from those of qubit-based quantum hardware. The photon loss error is arguably the most significant error in Bosonic hardware [11, 27]. Other types of error include dephasing [27] and thermal noise [4]. Their error rates are usually only about 10% of the photon loss error rate on average [27]. Regarding the operation fidelity, the most recent quantum photonics device Ascella [35] containing 12 qumodes can achieve fidelity of 99.6% and 93.8% for single-qumode and two-qumode gates, respectively. The state-of-the-art measurement fidelity is 95% on Xanadu's X8 device. On the superconducting Bosonic quantum processor [49], error rates are reported as around 1% for beamsplitters, 1% for squeezing gates, 0.02% for displacement gates, $1\% \sim 0.1\%$ for measurement. In summary, the errors from two-qumode gates and measurement are dominant for now.

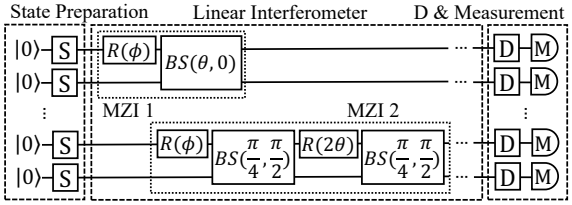

Fig. 2. Overview of a Gaussian Boson sampling (GBS) program

Along with the hardware development, initial efforts in programming and compilation for Bosonic Quantum Computing, such as Strawberry Fields [24], Bosonic Qiskit [44], and Perceval [35], have established basic programming interfaces, allowing users to construct Bosonic quantum programs with basic gates. Yet the compiler optimization development is much less developed.

III. Gaussian Boson Sampling

Gaussian Boson sampling (GBS) is a computation paradigm based on Bosonic QC. In this section, we provide an overview of GBS, as well as how to program GBS device. For interested readers, we recommend [11, 20] for more details.

A. GBS Program

A typical GBS program is depicted in the circuit diagram Figure 2. Similar to the quantum circuit for qubit-based quantum computing, each horizontal line represents one basic information processing unit: qumode. The blocks with different letters represented different qumode gates introduced in the previous Section II-A. The blocks are placed on different horizontal lines, representing those gates applied on different qumodes. The blocks with the 'M' symbol at the end represent measuring the qumodes. In GBS, the measurement is usually on the Fock basis and will return an integer $\{0, 1, 2, \dots\}$. The number represents the energy level of the current state and is usually called the number of photons in Physics literature. So that such measurement is also called the photon count measurement.

A GBS program typically has three major steps: state preparation, linear interferometer, and measurement. Usually, all qumodes are initialized to the vacuum state $|0\rangle$ with photon count 0. (Note that in Bosonic QC literature, many terms such as photons, interferometers, etc. are shared with photonics. However, in this paper, they represent purely software concepts, and can be implemented via diverse hardware platforms.) State preparation adds some photons to the system to prepare an initial state, which is done by applying squeezing gates in GBS. The prepared state will then be fed into a linear interferometer to perform a special calculation where the **high-level semantics** is a unitary matrix $\mathbf{U}$ applied on the vector of annihilation operators of the system and denoted as:

$$\hat{a} \to \hat{a}' = \mathbf{U}\hat{a}, \text{ where } \hat{a} = (\hat{a}_1, \hat{a}_2, \dots, \hat{a}_N)^\intercal.$$

The output of the linear interferometer is a multi-qumode Gaussian state. Finally, qumodes are measured. Sometimes

displacement gates are applied before the measurement, depending on the application.

B. Advantage of GBS

The GBS program above can demonstrate quantum advantage because it can compute the Hafnian of a matrix [9] efficiently while the Hafnian calculation is hard on classical computers. By carefully tuning the parameters in the qumode gates, we can obtain a final system state where the covariance matrix for the states of N qumodes is $\Sigma_{2N\times 2N}$. Then the Hafnian of the covariance matrix $\Sigma_{2N\times 2N}$ can be obtained by sampling from the final state with the photon count measurement [11, 20, 25] . The probability $Pr(S)$ of obtaining the measurement outcome $S = (s_1, s_2, \ldots, s_N)$ (s_i stands for observing s_i photons in qumode i) is given by:

$$Pr\left(S\right) = \frac{1}{\sqrt{\mathbf{det}\left(\mathbf{Q}\right)}} \frac{Haf\left(\mathcal{A}_S\right)}{s_1!s_2!\ldots s_N!} \tag{1}$$

here $\mathbf{Q}$ and $\mathcal{A}$ are completely determine by $\Sigma_{2N\times 2N}$:

$$\mathbf{Q} = \Sigma_{2N\times 2N} + \mathbb{I}_{2N}/2$$
$$\mathcal{A} = \begin{pmatrix} 0 & \mathbb{I}_N \\ \mathbb{I}_N & 0 \end{pmatrix} \left(\mathbb{I}_{2N} - \mathbf{Q}^{-1}\right)$$

and $\mathcal{A}_S$ is submatrix of $\mathcal{A}$ by selecting its rows and columns according to sample pattern S.

The matrix function $Haf(\cdot)$ shown in (1) is the Hafnian [9], and calculating the matrix Hafnian is a $\#P$-hard problem [48] ($\#P$-hard is at least as hard as NP-complete), thus it will be inefficient to obtain on classical computers.

C. Application of GBS

Consequently, GBS is suitable for applications that can benefit from fast matrix Hafnian calculation. We briefly introduce the following two categories:

(1) Graph Problem Associated with Hafnian: Many NP-hard graph problems, such as dense subgraph [6], graph clique [8], graph similarity [42], point process [22], can benefit from GBS since GBS can efficiently calculate the hafnian of graph's adjacency matrix. Although GBS cannot directly solve these problems in polynomial time, theoretical study shows that GBS can enhance any stochastic algorithm (e.g., random search, simulated annealing, and greedy algorithms) for such problems with provable speedup/success probability increase [7].

(2) Simulating Bosonic System: For example, molecule vibrational spectra simulation [21] is modeling Bosons which have infinite-dimension state space. Bosonic QC has natural advantages here as the Bosons can be directly encoded in qumodes and the Boson operations are also usually natively supported on Bosonic QC hardware.

These applications of different purposes can be programmed onto GBS devices by changing the qumode gate's parameters and the linear interferometer's unitary matrix. The three-step overall GBS program structure remains unchanged. How to turn the application into its corresponding program parameter setup can be found in [11]. The techniques in this paper do not rely on any application-specific information.

D. Limitation of GBS

On the hardware side, executing GBS programs requires Bosonics quantum processors with native hardware qumodes. Similar to qubit-based quantum computing hardware, they also suffer from noise, especially for the two-qumode gate, as introduced in Section II-B. Also, we cannot reach a state with infinite photons in practice. The reported numbers of photons measured from a physical qumode in experiments are 15 and 18, on superconducting Bosonic quantum processor [49] and photonic quantum processor [5], respectively. Fortunately, the amplitude of a high photon count basis state is usually exponentially small [5, 24].

On the theory side, although Bosonic quantum computing is universal, GBS is not a universal computing paradigm as it is mostly focused on accelerating the matrix Hafnian calculation. And due to the hardware noise, we can only execute approximate GBS in realistic hardware. Theoretical studies [3, 20, 25] show that if approximate Hafnian estimation can be efficiently solved on classical computers, it will imply polynomial hierarchy collapse [3, 52], which is believed to be unlikely to happen. Consequently, a common conjecture is that approximate GBS is also in $\#P$ [25].

E. GBS Program Compilation

Programming GBS device is to decide the parameters for each quantum gate. Typically, the parameters of squeezing gate and displacement gate are easy to calculate according to the applications. However, the linear interferometer, which is the largest building block in GBS, is given by a high-level unitary matrix $\mathbf{U}$ and needs to be decomposed into basic single- and two-qumode gates to be executed on hardware.

An $N \times N$ unitary matrix $\mathbf{U}$ for N qumode system has the following decomposition formula [41]:

$$\mathbf{U} = \Lambda \cdot \left(\prod_{(m,n)\in\mathcal{S}} \mathbf{T}_{m,n}\left(\theta, \phi\right) \right) \tag{2}$$

Here $\mathbf{T}_{m,n}(\theta, \phi)$ is a rotation in two-dimensional subspace, and $\mathcal{S}$ is a set containing pairs that describe the information of the dimension where the rotation is acting on. We also call the parameter θ as the rotation angle. Λ is the diagonal matrix whose diagonal entries λ_{ii} have modules $|\lambda_{ii}| = 1$. The matrix representation of $\mathbf{T}_{m,n}(\theta, \phi)$ is:

$$\mathbf{T}_{m,n}\left(\theta, \phi\right) = \begin{array}{c} \\ \text{row } m \\ \\ \text{row } n \\ \end{array} \begin{array}{c} \begin{array}{ccccc} \text{column } m & & & \text{column } n & \end{array} \\ \begin{pmatrix} \mathbb{I} & 0 & \cdots & 0 & 0 \\ 0 & e^{i\phi}\cos\left(\theta\right) & \cdots & -\sin\left(\theta\right) & 0 \\ \vdots & \vdots & \mathbb{I} & \vdots & \vdots \\ 0 & e^{i\phi}\sin\left(\theta\right) & \cdots & \cos\left(\theta\right) & 0 \\ 0 & 0 & \cdots & 0 & \mathbb{I} \end{pmatrix} \end{array}$$

Each $\mathbf{T}_{m,n}\left(\theta, \phi\right)$ can be implemented by an Mach-Zehnder interferometer (MZI) [40] circuit block applied on the m-th and n-th qumode [41]. This is the most expensive part of the computation due to noise. The error rate of a Beamsplitter

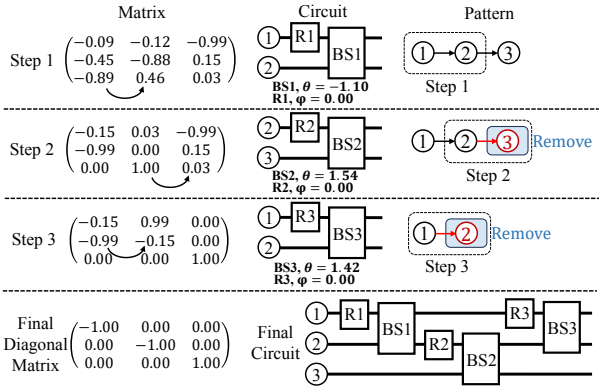

Fig. 3. Numerical example of the elimination process

can be over $10\times$ higher than that of other single-qumode gates [13, 49]. One MZI block can be realized using a Phase Shifter $R(\phi)$ on qumode m and a Beamsplitter $BS(\theta, 0)$ on qumodes m and n ('MZI 1' in Fig. 2). Some Bosonic quantum hardware platforms only natively support fixed $50:50$ Beamsplitters $BS(\pi/4, \pi/2)$ [12]. In this way, one MZI block can be implemented with two Phase Shifters and two Beamsplitters ('MZI 2' in Fig. 2). The discussion in this paper assumes the first implementation, but our proposed techniques are equally effective if using the second implementation.

Fig. 3 shows a numerical example of decomposing a 3-dimensional unitary matrix as well as the corresponding circuit. For step 1, we use qumode 2 to eliminate qumode 1, resulting in a circuit on qumode 1 and 2 shown in the middle. Such elimination can also be denoted in an elimination pattern. Each node in this pattern represents a qumode. We use the arrow from qumode 1 to 2 to represent this elimination. Then in step 2, we eliminate qumode 2 with qumode 3. This finishes the elimination in one row and the last qumode is removed from the pattern. In step 3, we use qumode 2 to eliminate qumode 1 and finally generate a diagonal matrix. The final constructed circuit is at the bottom. The elimination pattern is also used later in Section V-A.

IV. Problem Formulation

In this section we introduce the high-level algorithmic optimization opportunity to overcome the challenge of infinite-dimensional qumode gate matrices and formulate our Bosonic compiler optimization problem.

A. Opportunities from Decomposition Flexibility

The optimization opportunities come from the high-level algorithmic property of the $\mathbf{T}_{m,n}(\theta, \phi)$ transformations. This transformation can be considered as using one entry in column n to eliminate the entry in column m in the unitary $\mathbf{U}$, similar to a Gaussian elimination process. Overall, the decomposition of $\mathbf{U}$ is to find a series of $\mathbf{T}_{m,n}(\theta, \phi)$ transformations to convert the unitary $\mathbf{U}$ into a diagonal matrix. Note that the original decomposition formula in (2) does not require any specific elimination order. Thus, the decomposition can be very flexible. Existing decomposition methods [15, 41] decompose the unitary into some special $\mathbf{T}_{m,n}(\theta, \phi)$'s where

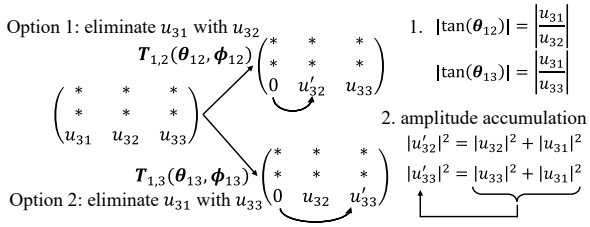

Fig. 4. Example of flexible decomposition

$n = m + 1$ is always fixed. The optimization opportunities from the flexible decomposition were missed.

We first illustrate the flexible decomposition of the linear interferometer unitary with the example in Fig. 4. There are two decomposition options in Fig. 4 to perform elimination on the row (u_{31}, u_{32}, u_{33}). The first option in the upper half of Fig. 4 is to eliminate u_{31} with u_{32} using an MZI block $\mathbf{T}_{1,2}(\theta_{12}, \phi_{12})$ on qumode 1 and 2. The parameters θ_{12} and ϕ_{12} are determined by u_{31} and u_{32} with following equation:

$$u_{31} e^{-i\phi_{12}} \cos(\theta_{12}) - u_{32} \sin(\theta_{12}) = 0 \tag{3}$$

We observe that there are two key properties of this elimination process that can help with the follow-up optimizations.

First, the rotation angle of the Beamsplitter in the generated MZI block satisfies: $|\tan(\theta_{12})| = |u_{31}/u_{32}|$. And we can notice that the parameter θ_{12} will be very small when $|u_{32}|$ is much larger than $|u_{31}|$. In this case, the Beamsplitter in the generated MZI block will be very close to an identity. Theoretically, the distance between a Beamsplitter with a small rotation angle θ and the identity is bounded by $\sim \theta^2/N$.

Second, all the matrices we used in decomposition are unitary, and the norm in every column and every row is always preserved. Therefore the amplitudes will be accumulated during the elimination. For example, in the first decomposition option in Fig. 4, the amplitude of the entry u_{31} will be accumulated into the new amplitude u'_{32} after the elimination with $|u'_{32}|^2 = |u_{31}|^2 + |u_{32}|^2$.

Another option shown in Fig. 4 is to eliminate u_{31} with u_{33}. This time the MZI block will be applied on qumode 1 and 3. The parameters and the amplitude accumulation will be different. This is a three-qumode example and the flexibility can be much larger with more qumodes.

B. Compilation Problem Formulation

This paper also considers the hardware's characteristics. Similar to qubit-based quantum hardware, Bosonic quantum hardware suffers from noise, and it is desirable to reduce the number of gates, especially the expensive two-qumode Beamsplitters. We also consider the qumode connectivity constraints and only allow MZI blocks to be applied on physically-adjacent qumode pairs with native Beamsplitter support.

Here we formulate the compilation problem of this paper. Given the input GBS program and the underlying hardware constraints, Bosehedral needs to find 1) the logical-to-physical qumode mapping and 2) the decomposition of the linear interferometer unitary into hardware-supported MZI blocks. Moreover, to mitigate the hardware noise effects, Bosehedral

aims to exploit the flexibility in decomposition such that the rotation angles in the generated Beamsplitters can be very small. These Beamsplitters are very close to the identity and can thus be dropped to mitigate their high gate error without significantly affecting the overall GBS program. Importantly, our method does not merely approximate the program, but it does so at the point of decomposition and mapping, and the compiler output's accuracy can be easily and scalably controlled by evaluating the fidelity of the reconstructed high-level unitary.

The compiler optimization in Bosehedral can be applied beyond optimizing GBS because its optimization target, the quantum linear interferometer, is a key component in many other applications [47], including but not limited to non-Gaussian Boson Sampling [3], quantum simulator [26], quantum metrology [18], quantum repeater network [46], etc.

V. Elimination Pattern Finding

We will first introduce the unitary decomposition optimization in Bosehedral. Our objective is to maximize the generation of MZI blocks with small Beamsplitter rotation angles. Our decomposition approach will only generate MZI blocks that are compatible with the qumode without any remapping in the middle. We will assume a trivial logical-to-physical qumode mapping in this section and discuss the related further optimization in the next section. Our design is based on the widely used two-dimensional lattice hardware coupling structure. But our idea and design flow can be generalized to other layouts like triangular or hexagonal arrays.

A. Elimination Pattern Template

We define an *elimination pattern template* to represent how the entries in the linear interferometer unitary $\mathbf{U}$ are eliminated. In this template, each node represents a qumode. When we eliminate one entry of one qumode i using the entry of another qumode j, we use a directed edge to connect the two corresponding qumodes from i to j.

Baseline elimination For example, the template in the upper part of Fig. 5 is the elimination template of existing decomposition methods [15, 41]. It has a chain structure. u_1 is first eliminated with u_2 so there is an edge from qumode 1 to 2. Then u_2 is eliminated with u_3, u_3 is eliminated with u_4, ..., and finally u_{N-1} is eliminated with u_N. After the elimination of the last row finishes, $|u_N|$ should be 1 due to amplitude accumulation. Then for the next row $N-1$, the last node N in the elimination pattern is removed. The elimination of row $N-1$ will follow the same pattern from u_1 but terminate at u_{N-1}. The node $N-1$ is then removed and the elimination of row $N-2$ will begin. Such process repeats from row N to row 1 and all the nodes in the elimination pattern are removed.

Bosehedral leverages the elimination flexibility mentioned in Section IV-A and redesigns a new elimination pattern. We first discuss the requirements and desired properties of such an elimination pattern graph.

First, the template must be a tree with all the directed edges from child nodes to parent nodes. As discussed in

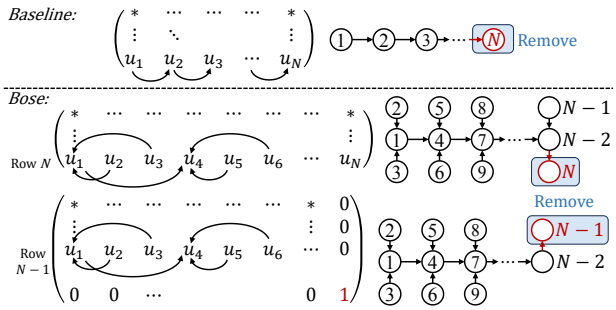

Fig. 5. Elimination pattern template: baseline vs Bosehedral

Section IV-A, the amplitudes of the entries are accumulated during the elimination process. The directed edges in the graph then naturally represent the flow of the amplitude accumulation. To complete the elimination of one row in the unitary, all the amplitudes must be finally accumulated onto one entry and all other entries must be zeroed. A tree with all the directed edges from child nodes to parent nodes can naturally represent how the amplitudes are accumulated from the leaf nodes to the root node. The dependency is that the entry of a parent node can only eliminate the entry of one child node after the entry of this child node has eliminated the entries of all its child nodes. For example, in the middle of Fig. 5, we must first eliminate qumode 2 and 3 with qumode 1 before we can further eliminate qumode 1 with qumode 4.

Second, recall that we hope to generate MZI blocks with small Beamsplitter rotation angles for further optimization and approximation. This can be realized by eliminating a small entry with a large entry (recall the first property of elimination in Section IV-A and Fig. 4). Meanwhile, the entries will become larger and larger as the elimination process gets close to the root due to the amplitude accumulation (the second property of elimination). If we can attach small entries to these entries with accumulated large amplitudes, we can create more large-small eliminations.

Finally, we also consider that this elimination pattern graph will later be physically realized in a two-dimensional lattice coupling structure. Although we have not considered the actual mapping onto hardware, we require that each node in this graph can have at most four neighboring nodes, so that it can be later mapped onto the hardware without introducing complicated transformations.

With all these considerations, our elimination pattern template graph design is shown in the lower part of Fig. 5. Basically, we have a *main path* in this elimination pattern reflecting the main flow of amplitude accumulation (the chain of nodes 1, 4, 7, ...). We can expect the amplitudes of previous nodes will be accumulated to the node throughout this main path, and they will be relatively large when they are used to eliminate other nodes. Thus for each node in the main path, we attach some leaf nodes as the *branches*. These branch nodes do not eliminate any other nodes so their amplitudes are expected to be small and large-small eliminations are created. Considering that each node has at most four neighbors, we attach two leaf nodes to each main path node, except the first and last node in the main path.

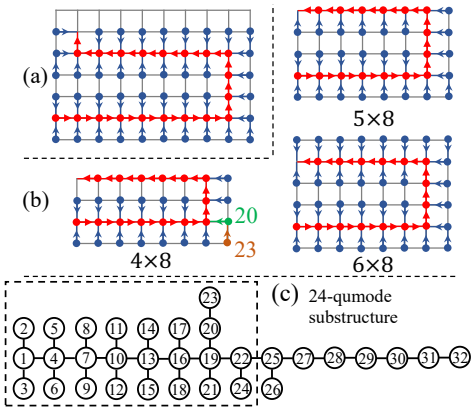

Fig. 6. Example of elimination pattern embedding

The elimination process is executed row by row from bottom to top. We start from row N and eliminate u_2 and u_3 with u_1 first. Then u_1 is eliminated with u_4. The elimination will follow the pattern on the right and finishes after accumulating all the amplitudes in row N to u_N and we have $|u_N| = 1$. All other entries in the column N will also become zero due to the column normalization of unitary matrices. Then we eliminate row $N-1$ and the node N is removed from the elimination pattern since there is no amplitude in the last entry of row $N-1$. We also need a small modification at the end of the pattern, flipping the edge direction from $N-1$ to $N-2$. This will make the elimination terminate at the $N-1$ node, which is a leaf node in the tree, and then this node $N-1$ is removed when eliminating the next row $N-2$. Such an elimination process repeats from row N to 1. The MZI blocks in the decomposed circuit can be generated using the parameters of the $\mathbf{T}_{m,n}(\theta, \phi)$'s obtained in the elimination.

B. Zigzag Elimination Pattern Embedding

The next step is to embed the elimination pattern into the actual hardware coupling graph, a two-dimensional lattice. We propose a Zigzag pattern embedding to maintain the overall structure in the original template. A general procedure is shown in Fig. 6 (a). We start the embedding by aligning the main path with the longer edge of the two-dimensional lattice. Here we start from the bottom left of the lattice and embed the pattern template in the first three rows. The start node of the main path is denoted as the 'start point'. When the embedding reaches the right side, the main path will turn up and align with the column edge. Around the turning point, we may be unable to attach two branch nodes to each main path node. If one node cannot be directly connected to the main path, we can attach it to one branch node (e.g., nodes 23 and 20 in Fig. 6 (b)). After we reach the next three rows, our main path will turn to the left and follow similar patterns to add branches. When we reach the left side, the main path will turn up again and finally formulate a Zigzag pattern. Depending on the result of the number of rows modulo 3, we may need to drop some branch nodes at the end. Fig. 6 (b) shows the three different cases. Finally, the main path will end on either the left or right edge at a point denoted as the 'end point'.

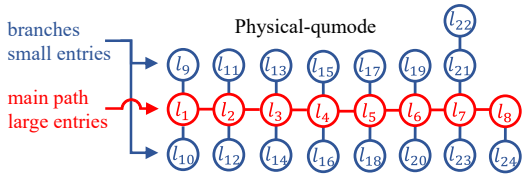

Fig. 7. Example of logical-to-physical mapping

C. Sub-Pattern Selection

The last step is to select some of the physical qumodes for the follow-up computation when the total number of qumodes on the device exceeds the total number of logical qumodes in the program. Recall that the purpose of our template pattern is to use the entries on the main path to eliminate entries on the branches to produce small rotation angles. Therefore, we will select the qumodes on the main path connecting to more branch qumodes and disregard those main path qumodes with fewer branch qumodes.

Our physical qumode selection will first label all the physical qumodes based on a breadth-first-search starting with the first qumode ('start point') in the template pattern. Fig. 6 (c) shows an embedding structure that comes from a 4×8 device. As shown in Fig. 6 (b), the qumodes that are far away from the start point will have fewer branches due to the edge of the two-dimensional lattice. We will choose the qumodes from the lower label to the higher label until the number of total qumodes is satisfied to implement the problem we are considering. An example of selecting a 24-qumode substructure from a 32-qumode 4×8 device is in Fig. 6 (c).

VI. Qumodes Mapping Optimization

In the previous section, we select a hardware-compatible elimination pattern based on the trivial mapping. In this section, we will introduce how the logical-to-physical qumode mapping can be optimized to further improve the yield of small Beamsplitter rotation angles via permutation operations that come with no execution overhead.

A. Motivating Example

We introduce our qumode mapping algorithm with a motivating example considering the unitary decomposition in one row. Recall that the small rotation angles are expected to happen when we use a qumode in the main path of the pattern to eliminate another qumode on the branches. If we can map the qumodes with large entries in the unitary on the main path and those with small entries on the branches at the very beginning, the angles we produced during the elimination will be further reduced. Suppose we have a 24-dimensional vector:

$$(l_1, l_2, l_3, \ldots, l_{24})$$

and we assume that their amplitudes satisfy $|l_1| \geq |l_2| \geq \ldots \geq |l_{24}|$ without loss of generality. In this example, we will perform elimination on this row using the 24-qumode elimination pattern from Fig. 6 (c).

A desired logical-to-physical qumode mapping of this example is depicted in Fig. 7. The 8-qumode row in the middle is

our main path, with its branches on two sides. Since we want large entries to appear in the main path, $l_1, l_2, \ldots, l_8$ will be mapped to the main path from the start point because they are the largest one. After mapping the main path, we deal with the remaining branches, and the large entries remaining should be sent to the branch near the start point, thus l_9, l_{10} are branches to l_1, other mappings are similar. In this way, the accumulated amplitude will be even larger when the elimination process along the main path is approaching the end point, because the branch nodes with large amplitudes are attached close to the start point. The branch nodes attached to the main path nodes near the end point will have the smallest amplitudes.

A special case is that the qumode on the main path has 3 branch qumodes, like qumode contains l_7 shown in Fig. 7. We will first map the larger one to the long branch, and the smaller one to the short branch. In Fig. 7, we put l_{21} and l_{22} to long side and l_{23} to the short side.

B. Mapping via Permutation

The qumode mapping of GBS in this paper is highly different from its counterpart in discrete-variable quantum computing, the qubit mapping problem. The qubit mapping usually involves determining the initial logical-to-physical qubit layout and injecting SWAPs in the middle to resolve the dependencies for each two-qubit gate. In contrast, our mapping optimization will directly encode the mapping transition into the unitary, the high-level algorithmic representation of the linear interferometer. We identify that the qumode mapping in GBS can be considered as adding permutation matrices before and after the unitary. And the permutation operations can be implemented without any additional gates.

We first show that performing the row permutation and column permutation to the unitary encoded in the GBS interferometer will not affect the final sampling result.

A matrix permutation can be expressed as:

$$\mathbf{U_{per}} = P_r \mathbf{U} P_c$$

where $\mathbf{U}$ is the original matrix, $\mathbf{U_{per}}$ is the matrix after permutation, P_r and P_c are row permutation and column permutation, respectively. We can rewrite the matrix $\mathbf{U}$ as:

$$\mathbf{U} = P_r^T \mathbf{U_{per}} P_c^T$$

In this case, if we want to encode unitary matrix $\mathbf{U}$ into the device, we can first encode the permutation P_c^T, then encode the unitary $\mathbf{U_{per}}$, and lastly encode another permutation P_r^T, as shown in Fig. 8.

Since the state preparation and measurement in GBS usually do not involve multi-qumode operations, P_c^T can be done by changing the initial logical-to-physical qumode mapping. Suppose the P_c^T is given by the following permutation:

$$i \to \pi_c(i)$$

this relationship means we transfer the qumode i into qumode $\pi_c(i)$, thus the physical qumode $\pi_c(i)$ after P_c^T contains the qumode i before this permutation. As a result, to implement the same transformation we can omit the permutation P_c^T and

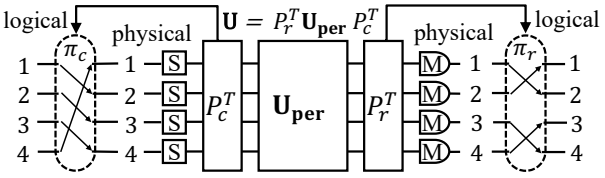

Fig. 8. Mapping via Permutation

map logical qumode i into the physical qumode $\pi_c(i)$ directly, as shown on the left of Fig. 8.

Similarly for the permutation P_r^T, suppose we get $output_i$ in the physical qumode i after the unitary transformation $\mathbf{U_{per}}$, this output should lie in the physical qumode $\pi_r(i)$ if we execute permutation P_r^T, where $\pi_r(i)$ stands for the permutation given by P_r^T. Thus, instead of implementing the P_r^T in the circuit, we can directly obtain the output of logical qumode $\pi_r(i)$ from the output of physical qumode i.

In summary, to obtain the GBS execution results with $\mathbf{U}$ as the linear interferometer from the GBS experiments using the permuted unitary $\mathbf{U_{per}}$, we just need to relabel the qumodes before and after the GBS program based on the permutation matrices P_c^T and P_r^T.

C. Two Properties of Elimination

We first introduce two mathematical properties of the elimination process. These two properties will guide the design of our qumode mapping algorithm. We use the example in Fig. 9. Suppose a and c have large amplitudes while b and d are much smaller. We use c to eliminate d in the last row.

First, the elimination will not change the sum of the squares of the amplitudes in the region of a row containing all the entries that have changed in the elimination. In Fig. 9, we highlight the red region in the second row and the blue region in the last row. In the example, changes only happen in the first and third columns. The two highlighted regions contain all the entries that have changed values in the transformation. Note that:

$$\begin{cases} |a|^2 + |b|^2 &= |\widetilde{a}|^2 + |\widetilde{b}|^2 \\ |c|^2 + |d|^2 &= |\widetilde{c}|^2 \end{cases}$$

Thus the sum of the squares of the amplitudes in the highlighted region will not change.

This property allows us to focus on the amplitude in the region as a whole instead of each entry's specific amplitude in this region. For example, in Fig. 9, when we decompose the last row, we may accumulate the amplitude of the first entry and second entry into the third entry. This process happens in the blue region, and then the third entry and the last entry can form a large-small pair to produce a small rotation angle. Because the sum of the squares of the amplitudes in the blue region is fixed, we can expect their amplitude accumulated as a whole to be large without checking each individual entry.

Second, the elimination for one row does not change the relative order of amplitudes of another row if the entries in both two rows are in decreasing order and the generated Beamsplitter rotation angle is small. That is, after the elimination, the amplitudes of $\widetilde{a}$ and $\widetilde{c}$ are still larger than that of $\widetilde{b}$. This

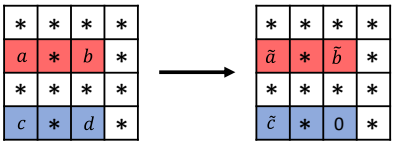

Fig. 9. Property of elimination

property can be understood with the formula that represents the elimination process:

$$\begin{cases} \widetilde{a} = ae^{-i\phi}\cos(\theta) - b\sin(\theta), \ \widetilde{b} = ae^{-i\phi}\sin(\theta) + b\cos(\theta) \\ \widetilde{c} = ce^{-i\phi}\cos(\theta) - d\sin(\theta), \ 0 = ce^{-i\phi}\sin(\theta) + d\cos(\theta) \end{cases}$$

The last equation gives the relationship that:

$$|\tan(\theta)| = \left|\frac{d}{c}\right|$$

which indicates θ is small since we assume d is small and c is large. From the first equation, we have the following:

$$|\frac{\widetilde{a}}{a}| \geq |\cos(\theta)| - |\frac{b}{a}||\sin(\theta)|$$

since θ and $|\frac{b}{a}|$ are small, the amplitude of $\widetilde{a}$ remains large. Similarly, the amplitude of $\widetilde{c}$ also remains large.

As for $\widetilde{b}$, we can derive the inequality that:

$$|\frac{\widetilde{b}}{a}| \leq |\sin(\theta)| + |\frac{b}{a}||\cos(\theta)|$$

Since θ and $|\frac{b}{a}|$ are small, the amplitude of $\widetilde{b}$ remains small compared with a.

This property is useful if we have a matrix in which the large entries appear in the beginning, and the small entries occur in the end for every row. Eliminating one of its rows won't change the order of absolute value in the remaining rows. As a result, if we find a good mapping for one of its rows (similar to the motivating example in Section VI-A), the elimination of other rows can still benefit from this mapping after the elimination of this row.

D. Finding the Permutations

We now describe our mapping method based on the properties and observations above. We explain it using the 24-qumode elimination pattern in Fig. 7. In the elimination pattern, there are 8 qumodes on the main path and 16 qumodes on the branches. Our objective is to map the large entries to the main path as much as possible. The first property allows us to consider the amplitudes in a region with multiple columns. So our first step is to move large entries to the left side via the column permutations. As depicted on the left of Fig. 10, we vertically divide the unitary into multiple regions. The first region is for the main path with 8 columns. The following regions are for the branches and each region corresponds to one branch. They usually have one or two columns because the branches have one or two qumodes.

After the column partition, we will calculate the sum of the squares of the amplitudes in the main path region in each row as shown on the right of Fig. 10. There summations are denoted as $\{\alpha_1, \ldots, \alpha_{24}\}$. We denote the K-th largest one in

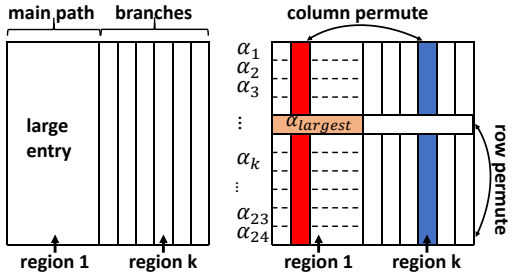

Fig. 10. Column and row permutations

this array as an indicator (K can be around the size of half of the unitary dimension). In practice, we select the value of K that can generate more small Beamsplitter rotation angles ($\theta < 0.1$).This indicator generally represents the amplitudes of the main path region entries of the largest K rows. Then we try to exchange the columns in the main path region 1 with the columns of other regions. If we find that one exchange can increase the indicator, we will accept this exchange. Such a process will merge the large entries to the main path and the first few branches. With the recorded column exchanges, we will generate the overall column permutation.

We then find the row permutation. Since we decompose the unitary from the bottom row, we hope that rows with good numerical properties (i.e., those with large entries in the main path) are placed at the bottom. With those good rows at the bottom, we can take advantage of the second property of the elimination process mentioned in Section VI-C. These bottom rows are first executed and the elimination of these rows will not affect the good numerical properties of other rows. Overall, our row permutation is generated by reordering the rows based on the sum of the squares of the amplitudes in the main path region of each row.

VII. Probabilistic Gate Dropout

The optimizations above have increased the occurrence of small rotation angles in the Beamsplitters. In this section, we introduce the probabilistic gate dropout method that will select a sequence of rotation angles in the decomposition of unitary. The purpose of this probabilistic dropout method is to drop those Beamsplitters with very small rotation angles with high probabilities while those with rotation angles near the threshold will be dropped more randomly to average over the algorithmic errors incurred by the approximation.

Reconstructing High-Level Semantics One key advantage of Bosehedral is that it can easily know the overall approximation effect after some gate dropout by reconstructing the high-level semantics from the decomposed gates. Recall the unitary decomposition formula in (2). Once we drop some Beamsplitters, we can reuse this formula to calculate the approximated unitary by setting the θ's in the corresponding MZI blocks to be 0. For example, if we drop the Beamplitter in the second and the third MZI blocks. We can just use the following formula to obtain the approximated unitary $\mathbf{U_{app}}$ by setting $\theta_2 = \theta_3 = 0$:

$$\mathbf{U_{app}} = \Lambda\left(\mathbf{T}\left(\theta_1, \phi_1\right)\mathbf{T}\left(\theta_2 = 0, \phi_2\right)\mathbf{T}\left(\theta_3 = 0, \phi_3\right)\cdots\right)$$

Note that all the matrices in this formula have size $N \times N$ so that the overall approximated unitary can be calculated efficiently. This allows us to easily know and tune how much approximation we will have during compilation time.

We now introduce our gate dropout method which monitors the overall approximation during dropout gate selection. After the decomposition, we will have $N(N-1)/2$ MZI blocks with their Beamsplitter rotation angles $\{\theta_1, \theta_2, \ldots, \theta_{N(N-1)/2}\}$. These angles will be selected using the following procedure.

1) We will select an accuracy threshold τ. Then we find the angle threshold $|\Theta|$ such that if we omit the rotations in which the angel's absolute value is less than $|\Theta|$, we can get the approximation unitary whose accuracy is just above the accuracy threshold τ. Suppose there are M angles kept at this step.
2) All the angles are divided by $|\Theta|$. We will select a positive integer K and raise the absolute values of the angles in the list to the K-th power.

$$\{|\theta_1/\Theta|^K, |\theta_2/\Theta|^K, \ldots, |\theta_{N(N-1)/2}/\Theta|^K\}$$

3) We normalize these new angles to construct a probability distribution:

$$p_i = \frac{|\theta_i/\Theta|^K}{\sum_{j=1}^{N(N-1)/2} |\theta_j/\Theta|^K}$$

This distribution represents how likely an angle θ_i will be picked into matrix reconstruction. There are two special cases. If $K = 1$, we are randomly sampling the Beamsplitters by their rotation angle amplitudes. If K goes to infinite (usually 100 is enough), we simply drop the angles smaller than the angel threshold $|\Theta|$.
4) We select L as the number of iterations. We select M angles by the probability distribution for each iteration and reconstruct the approximation unitary matrix. We denote τ_K as the average fidelity of the L iterations.
5) We find the positive integer K such that this process can maximize τ_K. In this case, we are able to maximize the approximation accuracy with M MZI blocks.

After M, $|\Theta|$, and K are determined, Bosehedral will generate the GBS circuit for each sample. One GBS program may require over thousands of repeated executions to obtain the final distribution. In each execution, we will use the probability distribution above to select M rotations and generate the GBS circuit with their associated MZI blocks.

VIII. Evaluation

In this section, we evaluate Bosehedral by comparing with state-of-the-art baselines, analyze the effects of each optimization step, and study the end-to-end application performance.

A. Experiments Setup

Experiment Configurations: To illustrate the effect of each optimization step, we design four experiment configurations. 1. 'Baseline' is to use the vanilla linear interferometer unitary decomposition [15] without any optimization. 2. 'Rot-Cut' is to directly drop gates under the baseline decomposition to reach a given unitary approximation rate. 3. 'Decomp-Opt' is to only use our optimized decomposition pattern with unitary approximation without qumode mapping optimization. 4. 'Full-Opt' is to apply all Bosehedral optimizations.

TABLE I
Benchmark Information

Benchmark	Qumode#	Squeezing	Displacement	Phase Shifter	Beamsplitter
DS	24	24	0	300	276
MC	24	24	0	300	276
GS	24	24	0	300	276
VS	24	48	24	0	552

Hardware Configuration: Similar to the qubit-based superconducting quantum architectures, two-dimensional lattice coupling is widely adopted in both recent experimental progress [49] and schematic design of superconducting Bosonic processors [10]. We select three different 2D lattices: 6×6, 5×7, and 3×8.

Benchmarks: We select four different typical GBS applications, Dense Subgraph (DS), Maximum Clique (MC), Graph Similarity (GS), and Molecule Vibration Spectra Simulation (VS), with four program instances for each benchmark. For DS, MC, and GS, we generate four random graphs of 24 nodes for each application and add edges between each pair of nodes with a probability of 0.7 to 0.9. The numbers of different types of gates for the benchmarks are listed in Table I. Note that for GBS programs, the numbers of gates mostly depend on the number of qumodes. For VS, we select the molecule Pyrrole using the data from Strawberry Fields [24]. We simulate its vibrational spectra at four temperatures (1000K, 750K, 500K, and 250K). All the programs in our benchmarks have 24 qumodes. This scale is limited by the classical simulator capability. One 24-qumode GBS experiment simulation requires a few CPU hours, and we report the simulation results of over 1000 GBS experiments in this paper.

Metrics: We use the Jensen-Shannon Divergence (JSD) [2] between the output distribution of the different experiment configurations and the standard output distribution as an application-independent metric to evaluate the improvement of Bosehedral. The standard distribution of each benchmark is generated by noise-free simulation of the original GBS program. The compilation effect is indicated by the fidelity of the approximated unitary matrix of the linear interferometer and the number of gates. The fidelity is defined as $tr(\mathbf{U}_{app} \cdot \mathbf{U}^\dagger)/N$ for N-qumode programs where $\mathbf{U}$ is the original $N \times N$ unitary and $\mathbf{U}_{app}$ is the approximated unitary. We also adopt application-specific metrics at the end to provide a more intuitive understanding of the end-to-end benefit of Bosehedral.

Implementation: We implemented Bosehedral in Python and leveraged basic infrastructure in Strawberry Fields [24]. Our noisy GBS simulation experiments are executed on the 'Gaussian' simulator backend in Strawberry Fields [24]. To the best of our knowledge, this is already the most advanced simulator available which can allow us to accurately simulate 24-qumode GBS experiments with noise. We simulate the gate photon loss error, which is the most significant error in

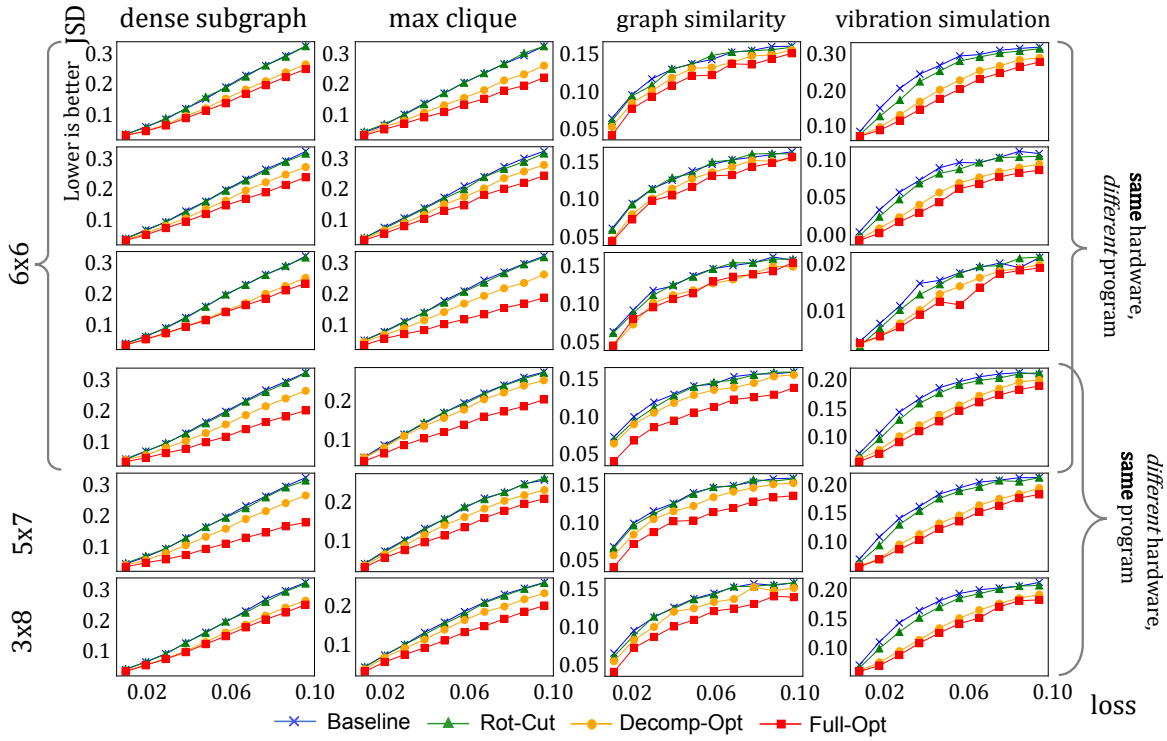

Fig. 11. Overall GBS execution quality improvement with Bosehedral optimizations

TABLE II
BEAMSPLITTER REDUCTION AND APPROXIMATED UNITARY FIDELITY

Benchmark & Fidelity	Rot-Cut	Decomp-Opt	Full-Opt (Avg. Beamsplitter #)
DS, 99.90%	4.3%	16.8%	28.8% (197)
MC, 99.96%	5.0%	18.4%	24.1% (210)
GS, 99.90%	3.4%	18.8%	26.0% (204)
VS, 98.00%	11.8%	34.7%	39.6% (333)

Bosonic hardware [11, 27] and currently the only error type supported in currently available simulator. Other types of error include dephasing [27] and thermal noise [4]. Their error rates are usually only about 10% of the photon loss error rate on average [27] while they are not covered by photon loss. We set the photon loss error rate from 0.01 to 0.1 based on recent experimental data [13, 32]. Each GBS experiment is sampled by 10000 times. In the probabilistic gate dropout, the accuracy threshold τ is set to be from 98.00% to 99.96%, then the Beamsplitter count M and angle threshold $|\Theta|$ are determined as described in Section VII. The power index K is decided by repeating select M Beamsplitters and calculating the average unitary approximation fidelity in 10000 samples, and its value ranges from 20 to 100. The experiments are executed on a server with 16 CPU cores and 128GB memory.

B. Overall Improvement

We first apply Bosehedral to approximate the linear interferometer to a certain fidelity for all the four benchmarks on the 6×6 architecture, and the results are the first four rows in Fig. 11. The X-axis is the loss rate and the Y-axis is the Jensen-Shannon Divergence (JSD) between the output distribution of the standard output and the distribution of the corresponding experimental configuration. Each column represents one application and has four program instances for each. Table II shows the fidelities of the approximated unitaries and the gate count reduction Bosehedral is able to achieve. A small JSD will indicate better performance. It can be observed that as the loss increases, the JSD of 'Full-Opt' grows much slower than that of 'Baseline'. On average, the JSD of 'Full-Opt' can be reduced by 31.6%, 33.8%, 12.6%, 26.4%, compared with 'Baseline' for the DS, MC, GS, and VS benchmarks, respectively. The great improvement comes from the fact that Bosehedral can approximate the linear interferometer accurately using much fewer gates. As shown in Table II, 'Full-Opt' can reduce 28.8%, 24.1%, 26.0%, 39.6% Beamsplitters but still maintain the fidelity of the linear interferometer unitary over 99.90%, 99.96%, 99.90%, 98.00%, respectively. The average remaining Beamsplitter count for 'Full-Opt' is also in Table II. Note that the single-qumode gates are not changed in Bosehedral. In summary, with a tiny algorithmic error introduced in the unitary approximation, Bosehedral can improve the overall performance by largely mitigating the hardware error.

Importance of new decomposition and mapping: It can be observed that directly dropping the gates can hardly provide any improvement as the JSDs of 'Baseline' and 'Rot-Cut' are very close (shown in Fig. 11). The Beamsplitter reduction is also very small (6.1% on average in Table II), indicating a large hardware error. These results suggest that the optimized decomposition pattern and qumode mapping is necessary to enable the optimizations in Bosehedral.

Effect of Each Step: The effect of the decomposition pattern optimization can be obtained by comparing 'Decomp-Opt' against 'Baseline'. Fig. 11 shows that 'Decomp-Opt' can reduce the JSDs by 21.2%, 16.3%, 7.2%, 19.5%. Table II

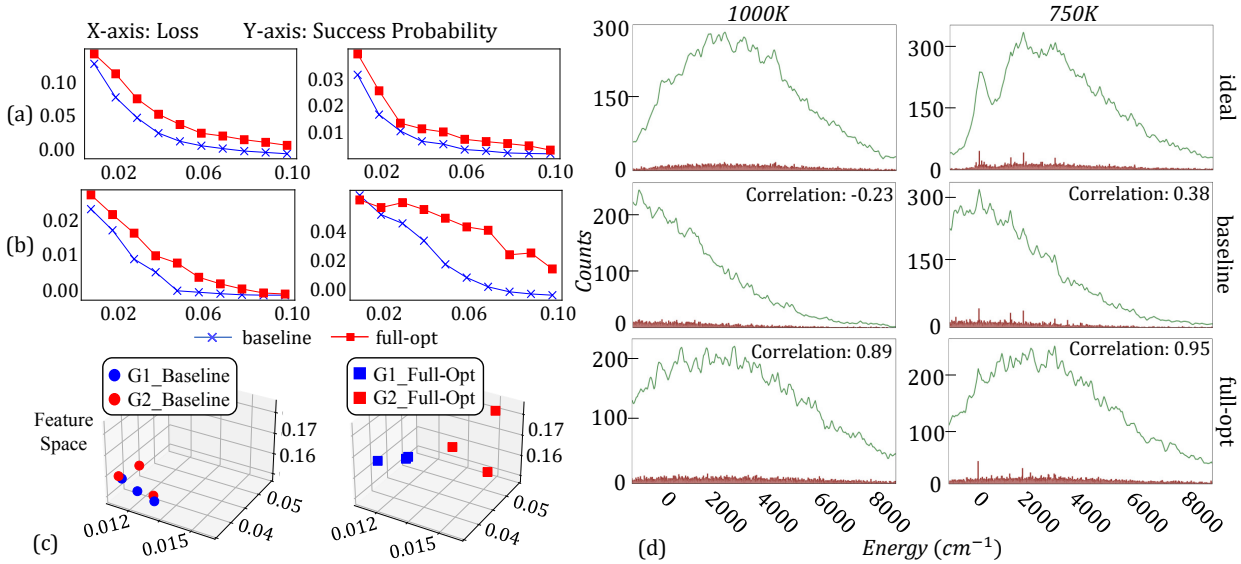

Fig. 12. End-to-end performance improvement. (a) Dense Subgraph, (b) Maximum Clique, (c) Graph Similarity, (d) Vibration Spectra

shows that 'Decomp-Opt' can reduce the gate count by 16.8%, 18.4%, 18.8%, 34.7%. The effect of qumode mapping can be observed when comparing 'Full-Opt and 'Decomp-Opt'. At the given approximated unitary fidelities, our logical-to-physical qumode mapping can further reduce the JSD by 10.4%, 17.5%, 5.4%, and 6.9% on average for the four benchmarks. Table II also shows that the Beamsplitter count reduction is increased by 12.0%, 5.7%, 7.2%, 4.9% (7.4% on average), respectively. The contribution breakdown between the decomposition optimization and the qumode mapping optimization is about 1.6 : 1 and 3.0 : 1 by comparing the JSD and Beamsplitter count reduction, respectively.

C. Hardware Structure Impact

We also studied the impact of different hardware coupling structures, and the results are in the last three rows in Fig. 11. Each column represents one application, and each row represents one hardware structure. We select one program instance for each application (one random graph for DS, MC, GS, and the 750K temperature simulation for VS). The results of other program instances of one application are similar. It can be observed that the improvement of Bosehedral is not affected by changing to a different 2D lattice structure. On the 5×7 and 3×8 structures, the JSDs are reduced by 36.6%, 25.1%, 16.1%, 28.9% for the four benchmarks, which is similar to the improvement on 6×6 structure.

D. End-to-End Application Performance Improvement

In order to better understand the impact of Bosehedral optimizations on the end-to-end application performance, we append the post-GBS data processing for all the benchmarks and evaluate them case-by-case. The details of the post-processing procedures for the selected benchmarks are out-of-scope, but they can be found in [11].

Dense Subgraph: The GBS output will directly indicate a subgraph, and we measure the probability of successfully finding the densest subgraph of which the number of nodes is greater or equal to 10 in two of our random graphs, and the results are in Fig. 12 (a). The end-to-end success probability is increased by 64.1% on average after Bosehedral optimizations.

Maximum Clique: The GBS output will serve as an initial trial in a follow-up clique finding subroutine. We measure the probability of successfully finding the clique whose nodes are greater or equal to 10 in two random graphs, and the results are in Fig. 12 (b). The end-to-end success probability is increased by 72.9% on average after using Bosehedral.

Graph Similarity: The sampled output distribution will be converted into graph features. We randomly generate two highly-different graphs as two seeds and then generate two sets of similar graphs by adding small modifications to the two different seeds. Fig. 12 (c) shows the feature vectors in the feature space sampled from the graphs in the two similarity sets G1 and G2. On the left are the feature vectors obtained from 'Baseline'. On the right are the feature vectors obtained with 'Full-Opt'. We can observe that the feature vectors are almost mixed for 'Baseline' as the significant photon loss error tends to lose information about the sampled graphs. But for the feature vectors of 'Full-Opt', they remain easily distinguishable for two clusters. We measure the distance between the two clusters' central positions in Fig. 12 (c). It shows that the distance is increased by 135% after using Bosehedral.

Vibration Spectra: We calculate the sampled molecule vibrational spectra using the GBS output. Fig. 12 (d) shows the simulated vibrational spectra at 1000K and 750K with loss at 0.02. The brown bars in the plot are the histogram of sampled energies and the green curve above is a Lorentzian broadening of the spectrum, which is a common practice in visualizing such a spectrum [11]. The first row is the standard vibrational spectra. The second row is the spectra obtained from the 'Baseline' configuration. The third row is the spectra obtained from 'Full-Opt'. Compared with 'Baseline', the results of 'Full-Opt' are much more similar to the ideal spectra for both

TABLE III
PERFORMANCE AT DIFFERENT PROBLEM SCALES (FIDELITY=0.95)

Qumode #	10	15	20	60	100	200	500
BS Gate # drop	29.33%	34.9%	31.6%	27.6%	27.5%	27.3%	27.1%
Decomp time	0.013s	0.033s	0.056s	0.67s	2.4s	21.4s	615.6s
Total time	0.016s	0.039s	0.067s	0.85s	3.3s	32.3s	1071.1s

temperature settings. Quantitatively, the Pearson correlation coefficients between the spectra generated by 'Baseline' and the standard spectra are -0.23, 0.38 at 1000K and 750K, and the results of 'Full-Opt' have much higher correlation coefficients 0.89 and 0.95. A correlation coefficient closer to 1 is better. The spectra of the 'Baseline' tend to shift to the low energy side due to the photon loss during the simulation.

E. Scalability Study

Although our GBS simulation is limited within 24 qumodes due to the complexity of classically simulating GBS, Bosehedral comes with great scalability and can be applied to much larger GBS programs. All the program analysis and compilation are performed on the high-level unitary representation of the linear interferometer, whose size grows linearly as the number of qumodes increases. The most time-consuming step is the matrix decomposition that has a complexity of $O(N^3)$ where N is the number of qumodes. Here we select seven numbers of qumodes from 10 to 500. For each qumode count N, we randomly generate five unitaries as the interferometer, and then apply Bosehedral to optimize it with a unitary approximation fidelity at 95% on a $3 \times \frac{N}{3}$ device. Table III shows the average results, including the gate count reduction, the decomposition time, and the total time, of applying full optimization on five random unitaries. Even for 500-qumode programs, Bosehedral can still reduce the number of gates by 27.1% and the overall compilation time is about 18 minutes.

F. Discussion on Approximated Compilation

To summarize, Bosehedral optimizes the qumode gate decomposition and logical-to-physical qumode mapping to generate more BS gates with small rotations. Then these gates with small rotation parameters can be pruned.

Result of Approximation. The result of the approximation technique with gate pruning is that the hardware errors associated with the pruned gates are eliminated. The semantics of the original GBS program have also changed since the unitary of the linear interferometer is approximated. But the semantics are only slightly changed because we only remove gates with small rotation parameters and these gates are very close to the identity. Overall, the benefit from gate error reduction is larger than the side effect of approximation, and the final execution fidelity can be improved, as reflected by the JSD decrease.

Tolerable Approximation Ratio. The approximation ratio that can be tolerated is mostly determined by the desired fidelity of the specific application. For DS and MC tasks, if a suboptimal sampled graph is acceptable then a higher approximation ratio can be tolerated. For GS and VS, the tolerable approximation ratio is determined by the desired robustness of the follow-up classifier and the required simulating accuracy, respectively.

Tolerable Gate Pruning. First, it is obvious that the tolerable gate pruning is directly related to the tolerable approximation ratio. A higher approximation ratio can naturally lead to more gate pruning. Second, the entry amplitude distribution of the linear interferometer unitary, usually determined by the application, will affect how many gates we can prune.

In addition to the effect of the applications, the hardware coupling structure will also affect the gate pruning because different hardware can support different elimination patterns. Recall that in the elimination template in Fig. 5, each node on *main path* has two *branches*, which means we can have two quantum gates with small rotation angles together with one quantum gate with big rotation angle. Then the upper bound of gate pruning ratio in this pattern template is around 2/3 because we always need to keep the gates in the main path. But if we have a fully connected qumode device, we can freely apply the BS gate on arbitrary two qumodes to create much more small rotation angles and the upper bound of gate pruning can be much higher (in our preliminary study we can reach over 0.9).

IX. RELATED WORK

Bosonic Quantum Software Frameworks There have been several early efforts on the software framework for programming and compilation of Bosonic QC, such as the Xanadu's Strawberry Fields [24], Quandela's Perceval [35], and Bosonic Qiskit [44]. These works provide basic programming infrastructures for Bosonic QC but with almost no optimizations to the best of our knowledge. In addition, [23] studied the low-level pulse compilation for individual qumode gates with analytical solutions for a single superconducting qubit-qumode pair. [14] designed a language to describe the linear optic quantum circuit. Unfortunately, none of them is able to simplify a Bosonic quantum program.

Linear Interferometer Implementation To help design the linear optics experiments, previous works [15, 41] have studied how to implement a linear interferometer with available optics instruments. Their solutions later serve as the linear interferometer implementation methods in Bosonic QC software frameworks like the Strawberry Fields [24] and Perceval [35]. As introduced in Section V-A (Fig. 5), they use a chain-structure elimination pattern with no remapping where it is hard to generate Beamsplitters with small rotations.

Approximated and Topology-Aware Quantum Compilation The approximated or topology-aware compilation has been widely explored in qubit-based quantum computing. Notable examples include a series of works [16, 38, 51, 53, 55] in the BQSkit project [54] as well as some other works [17, 28–30, 33, 36]. However, due to the difficulty in evaluating the gate matrices when the number of qubits is large, their approximation calculation and topology-aware resynthesis are usually limited to small-scale circuit blocks at each step or special types of circuits like Hamiltonian simulation. In summary, these approaches targeting qubit-based QC are not applicable in Bosonic QC because of the built-in infinite-dimensional state spaces.

APPENDIX

A. Abstract

This artifact contains the code of the core algorithm of Bosehedral compiler and related experiments to reproduce our key results and figures, including Tables II, III and all sub-figures in Figures 11 and 12. The artifact and experiments require an x86-64 Linux server with at least 32GB RAM and 50GB available disk space. Multiple machines are recommended to prepare to obtain all results in Figures 11 and 12 in a reasonable amount of time. We split sub-figures into different files for possible parallelization.

B. Artifact check-list (meta-information)

- **Algorithm:** Our key algorithm, the Bosehedral compiler framework, is in each notebook.
- **Program:** Our simulation programs for different experiments are prepared in each notebook.
- **Run-time environment:** Python 3.10, Jupyter Notebook
- **Hardware:** an x86-64 Linux server with at least 32GB RAM and 50GB disk space
- **Metrics:** Unitary approximation fidelity and Jensen-Shannon divergence
- **Experiments:** Tables II, III and Figures 11, 12
- **Disk space required:** 50GB
- **Time needed to prepare workflow:** 10 minutes
- **Time needed to complete experiments:** 2 weeks
- **Publicly available:** Yes
- **Code licenses:** MIT
- **Workflow framework:** Python, StrawberryFields, Jupyter notebook
- **Archived:** 10.5281/zenodo.10895187

C. Description

1) How to access: Our artifact is published on GitHub https://github.com/JunyuZhou2002/Compiler-Optimization-for-Bosonic-Quantum-Computing.git or through archive https://doi.org/10.5281/zenodo.10895187.

2) Hardware dependencies: A Linux server with a single CPU can execute our artifact. RAM should be 32GB or above, and available disk space should be 50GB or more. However, since most experiments are *independent* and to avoid long execution time, we strongly recommended preparing multiple machines with powerful CPUs to evaluate multiple experiments simultaneously.

3) Software dependencies: Python 3.10 is required to run our experiments. Linux is recommended since some Python packages may only work under Linux. All software dependencies are included in the `requirements.txt`.

D. Installation

After cloning or downloading our artifact, enter the folder. It is recommended to work inside a virtual environment to avoid conflicts. Use the following commands to install the necessary Python packages for our experiment:

```
python3 -m venv venv
source venv/bin/activate
pip3 install -r requirements.txt
```

E. Experiment workflow

After downloading our artifact and preparing the virtual environment following installation instructions, you can open the Jupyter Notebooks in the virtual environment to reproduce our results. Notebooks are carefully split, and the order does not matter. The correspondents between experiments and artifacts are in Tables IV and V:

TABLE IV
CORRESPONDENTS BETWEEN FIGURES AND ARTIFACTS

Figure	Sub-Figure	Artifacts
Figure 11	dense subgraph	`src/dense-subgraph/*`
	max clique	`src/max-clique/*`
	graph similarity	`src/graph-similarity/*`
	vibration simulation	`src/vibration-spectra/*`
Figure 12	(a) left	`src/dense-graph/4`
	(a) right	`src/dense-graph/2`
	(b) left	`src/max-clique/4`
	(b) right	`src/max-clique/3`
	(c) left	`src/graph-similarity/1&4`
	(c) right	`src/graph-similarity/1&4`
	(d) left	`src/vibration-spectra/1`
	(d) right	`src/vibration-spectra/4`

TABLE V
CORRESPONDENTS BETWEEN TABLES AND ARTIFACTS

Table	Artifacts
Table II	`src/fidelity/fidelity.ipynb`
Table III	`src/scalability/scale.ipynb`

F. Evaluation and expected results

Notebooks plot the exact sub-figures as in our paper, except for some styling differences. Tables II and III are directly printed.

G. Experiments Customization

Due to internal issues with OpenBLAS, we limited its thread number to 8 at the beginning of each notebook. However, you could set `default_n_threads` to a larger number for higher speedup if it does not cause errors.

H. Notes

It is *recommended* to *simultaneously run multiple experiments on multiple machines* since simulating GBS classically consumes vast amounts of time.

REFERENCES

[1] "Careers // quantum circuits, inc." https://quantumcircuits.com/careers, (Accessed on 11/21/2023).

[2] "Jensen–shannon divergence - wikipedia," https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence, (Accessed on 08/06/2023).

[3] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," in *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, ser. STOC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 333–342. [Online]. Available: https://doi.org/10.1145/1993636.1993682

[4] E. J. Anderson and B. A. Bash, "Fundamental limits of thermal-noise lossy bosonic multiple access channel," in *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2022, pp. 1–6.

[5] J. M. Arrazola, V. Bergholm, K. Brádler, T. R. Bromley, M. J. Collins, I. Dhand, A. Fumagalli, T. Gerrits, A. Goussev, L. G. Helt, J. Hundal, T. Isacsson, R. B. Israel, J. Izaac, S. Jahangiri, R. Janik, N. Killoran, S. P. Kumar, J. Lavoie, A. E. Lita, D. H. Mahler, M. Menotti, B. Morrison, S. W. Nam, L. Neuhaus, H. Y. Qi, N. Quesada, A. Repingon, K. K. Sabapathy, M. Schuld, D. Su, J. Swinarton, A. Száva, K. Tan, P. Tan, V. D. Vaidya, Z. Vernon, Z. Zabaneh, and Y. Zhang, "Quantum circuits with many photons on a programmable nanophotonic chip," *Nature*, vol. 591, no. 7848, pp. 54–60, Mar 2021. [Online]. Available: https://doi.org/10.1038/s41586-021-03202-1

[6] J. M. Arrazola and T. R. Bromley, "Using gaussian boson sampling to find dense subgraphs," *Physical review letters*, vol. 121, no. 3, p. 030503, 2018.

[7] J. M. Arrazola, T. R. Bromley, and P. Rebentrost, "Quantum approximate optimization with gaussian boson sampling," *Physical Review A*, vol. 98, no. 1, p. 012322, 2018.

[8] L. Banchi, M. Fingerhuth, T. Babej, C. Ing, and J. M. Arrazola, "Molecular docking with gaussian boson sampling," *Science Advances*, vol. 6, no. 23, p. eaax1950, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.aax1950

[9] A. Björklund, B. Gupt, and N. Quesada, "A faster hafnian formula for complex matrices and its benchmarking on a supercomputer," *Journal of Experimental Algorithmics (JEA)*, vol. 24, pp. 1–17, 2019.

[10] A. Blais, A. L. Grimsmo, S. M. Girvin, and A. Wallraff, "Circuit quantum electrodynamics," *Rev. Mod. Phys.*, vol. 93, p. 025005, May 2021. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.93.025005

[11] T. R. Bromley, J. M. Arrazola, S. Jahangiri, J. Izaac, N. Quesada, A. D. Gran, M. Schuld, J. Swinarton, Z. Zabaneh, and N. Killoran, "Applications of near-term photonic quantum computers: software and algorithms," *Quantum Science and Technology*, vol. 5, no. 3, p. 034010, may 2020. [Online]. Available: https://dx.doi.org/10.1088/2058-9565/ab8504

[12] J. Carolan, C. Harrold, C. Sparrow, E. Martín-López, N. J. Russell, J. W. Silverstone, P. J. Shadbolt, N. Matsuda, M. Oguma, M. Itoh, G. D. Marshall, M. G. Thompson, J. C. F. Matthews, T. Hashimoto, J. L. O'Brien, and A. Laing, "Universal linear optics," *Science*, vol. 349, no. 6249, pp. 711–716, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aab3642

[13] B. J. Chapman, S. J. de Graaf, S. H. Xue, Y. Zhang, J. Teoh, J. C. Curtis, T. Tsunoda, A. Eickbusch, A. P. Read, A. Koottandavida, S. O. Mundhada, L. Frunzio, M. Devoret, S. Girvin, and R. Schoelkopf, "High-on-off-ratio beam-splitter interaction for gates on bosonically encoded qubits," *PRX Quantum*, vol. 4, p. 020355, Jun 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.4.020355

[14] A. Clément, N. Heurtel, S. Mansfield, S. Perdrix, and B. Valiron, "LO_v-Calculus: A Graphical Language for Linear Optical Quantum Circuits," in *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Szeider, R. Ganian, and A. Silva, Eds., vol. 241. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 35:1–35:16. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16833

[15] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "Optimal design for universal multiport interferometers," *Optica*, vol. 3, no. 12, pp. 1460–1465, Dec 2016. [Online]. Available: https://opg.optica.org/optica/abstract.cfm?URI=optica-3-12-1460

[16] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Towards optimal topology aware quantum circuit synthesis," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2020, pp. 223–234. [Online]. Available: https://doi.org/10.1109/QCE49297.2020.00036

[17] A. M.-v. de Griend and R. Duncan, "Architecture-aware synthesis of phase polynomials for nisq devices," *arXiv preprint arXiv:2004.06052*, 2020. [Online]. Available: https://arxiv.org/abs/2004.06052

[18] J. P. Dowling, "Quantum optical metrology–the lowdown on high-n00n states," *Contemporary physics*, vol. 49, no. 2, pp. 125–143, 2008.

[19] P. Gokhale, J. M. Baker, C. Duckering, N. C. Brown, K. R. Brown, and F. T. Chong, "Asymptotic improvements to quantum circuits via qutrits," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 554–566.

[20] C. S. Hamilton, R. Kruse, L. Sansoni, S. Barkhofen, C. Silberhorn, and I. Jex, "Gaussian boson sampling," *Phys. Rev. Lett.*, vol. 119, p. 170501, Oct 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.119.170501

[21] J. Huh, G. G. Guerreschi, B. Peropadre, J. R. McClean, and A. Aspuru-Guzik, "Boson sampling for molecular vibronic spectra," *Nature Photonics*, vol. 9, no. 9, pp. 615–620, aug 2015. [Online]. Available: https://doi.org/10.1038%2Fnphoton.2015.153

[22] S. Jahangiri, J. M. Arrazola, N. Quesada, and N. Killoran, "Point processes with gaussian boson sampling," *Phys. Rev. E*, vol. 101, p. 022134, Feb 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.101.022134

[23] C. Kang, M. B. Soley, E. Crane, S. Girvin, and N. Wiebe, "Leveraging hamiltonian simulation techniques to compile operations on bosonic devices," *arXiv preprint arXiv:2303.15542*, 2023. [Online]. Available: https://arxiv.org/abs/2303.15542

[24] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook, "Strawberry fields: A software platform for photonic quantum computing," *Quantum*, vol. 3, p. 129, mar 2019. [Online]. Available: https://doi.org/10.22331%2Fq-2019-03-11-129

[25] R. Kruse, C. S. Hamilton, L. Sansoni, S. Barkhofen, C. Silberhorn, and I. Jex, "Detailed study of gaussian boson sampling," *Physical Review A*, vol. 100, no. 3, p. 032326, 2019.

[26] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri *et al.*, "Towards quantum chemistry on a quantum computer," *Nature chemistry*, vol. 2, no. 2, pp. 106–111, 2010.

[27] P. Leviant, Q. Xu, L. Jiang, and S. Rosenblum, "Quantum capacity and codes for the bosonic loss-dephasing channel," *Quantum*, vol. 6, p. 821, 2022.

[28] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[29] G. Li, Y. Shi, and A. Javadi-Abhari, "Software-hardware co-optimization for computational chemistry on superconducting quantum processors," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. IEEE Press, 2021, p. 832–845. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00070

[30] G. Li, A. Wu, Y. Shi, A. Javadi-Abhari, Y. Ding, and Y. Xie, "Paulihedral: A generalized block-wise compiler optimization framework for quantum simulation kernels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 554–569. [Online]. Available: https://doi.org/10.1145/3503222.3507715

[31] S. Lloyd and S. L. Braunstein, "Quantum computation over continuous variables," *Phys. Rev. Lett.*, vol. 82, pp. 1784–1787, Feb 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.82.1784

[32] Y. Lu, A. Maiti, J. W. Garmon, S. Ganjam, Y. Zhang, J. Claes, L. Frunzio, S. Girvin, and R. J. Schoelkopf, "A high-fidelity microwave beamsplitter with a parity-protected converter," *arXiv preprint arXiv:2303.00959*, 2023. [Online]. Available: https://arxiv.org/abs/2303.00959

[33] L. Madden and A. Simonetto, "Best approximate quantum compiling problems," *ACM Transactions on Quantum Computing*, vol. 3, no. 2, mar 2022. [Online]. Available: https://doi.org/10.1145/3505181

[34] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, A. E. Lita, T. Gerrits, S. W. Nam, V. D. Vaidya, M. Menotti, I. Dhand, Z. Vernon, N. Quesada, and J. Lavoie, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, pp. 75–81, Jun 2022. [Online]. Available: https://doi.org/10.1038/s41586-022-04725-x

[35] N. Maring, A. Fyrillas, M. Pont, E. Ivanov, P. Stepanov, N. Margaria, W. Hease, A. Pishchagin, T. H. Au, S. Boissier, E. Bertasi, A. Baert, M. Valdivia, M. Billard, O. Acar, A. Brieussel, R. Mezher, S. C. Wein, A. Salavrakos, P. Sinnott, D. A. Fioretto, P.-E. Emeriau, N. Belabas, S. Mansfield, P. Senellart, J. Senellart, and N. Somaschi, "A general-purpose single-photon-based quantum computing platform," 2023. [Online]. Available: https://arxiv.org/abs/2306.00874

[36] S. Martiel and T. G. d. Brugière, "Architecture aware compilation of quantum circuits via lazy synthesis," *Quantum*, vol. 6, p. 729, Jun. 2022. [Online]. Available: https://doi.org/10.22331/q-2022-06-07-729

[37] A. Narla, S. Shankar, M. Hatridge, Z. Leghtas, K. M. Sliwa, E. Zalys-Geller, S. O. Mundhada, W. Pfaff, L. Frunzio, R. J. Schoelkopf, and M. H. Devoret, "Robust concurrent remote entanglement between two superconducting qubits," *Phys. Rev. X*, vol. 6, p. 031036, Sep 2016. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.6.031036

[38] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari, "Quest: Systematically approximating quantum circuits for higher output fidelity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 514–528. [Online]. Available: https://doi.org/10.1145/3503222.3507739

[39] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.2573505

[40] J. Rarity, P. Tapster, E. Jakeman, T. Larchuk, R. Campos, M. Teich, and B. Saleh, "Two-photon interference in a mach-zehnder interferometer," *Physical review letters*, vol. 65, no. 11, p. 1348, 1990.

[41] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, "Experimental realization of any discrete unitary operator," *Phys. Rev. Lett.*, vol. 73, pp. 58–61, Jul 1994. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.73.58

[42] M. Schuld, K. Brádler, R. Israel, D. Su, and B. Gupt, "Measuring the similarity of graphs with a gaussian boson sampler," *Physical Review A*, vol. 101, no. 3, mar 2020. [Online]. Available: https://doi.org/10.1103%2Fphysreva.101.032314

[43] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, nov 2020. [Online]. Available: https://dx.doi.org/10.1088/2058-9565/ab8e92

[44] T. J. Stavenger, E. Crane, K. C. Smith, C. T. Kang, S. M. Girvin, and N. Wiebe, "C2qa - bosonic qiskit," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, pp. 1–8. [Online]. Available: https://doi.org/10.1109/HPEC55821.2022.9926318

[45] C. Taballione, M. C. Anguita, M. de Goede, P. Venderbosch, B. Kassenberg, H. Snijders, N. Kannan, W. L. Vleeshouwers, D. Smith, J. P. Epping, R. van der Meer, P. W. H. Pinkse, H. van den Vlekkert, and J. J. Renema, "20-Mode Universal Quantum Photonic Processor," *Quantum*, vol. 7, p. 1071, Aug. 2023. [Online]. Available: https://doi.org/10.22331/q-2023-08-01-1071

[46] M. Takeoka, S. Guha, and M. M. Wilde, "Fundamental rate-loss tradeoff for optical quantum key distribution," *Nature communications*, vol. 5, no. 1, p. 5235, 2014.

[47] S.-H. Tan and P. P. Rohde, "The resurgence of the linear optics quantum interferometer — recent advances & applications," *Reviews in Physics*, vol. 4, p. 100030, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405428318300431

[48] L. G. Valiant, "The complexity of computing the permanent," *Theoretical computer science*, vol. 8, no. 2, pp. 189–201, 1979.

[49] C. S. Wang, J. C. Curtis, B. J. Lester, Y. Zhang, Y. Y. Gao, J. Freeze, V. S. Batista, P. H. Vaccaro, I. L. Chuang, L. Frunzio, L. Jiang, S. M. Girvin, and R. J. Schoelkopf, "Efficient multiphoton sampling of molecular vibronic spectra on a superconducting bosonic processor," *Phys. Rev. X*, vol. 10, p. 021060, Jun 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.10.021060

[50] C. Weedbrook, S. Pirandola, R. García-Patrón, N. J. Cerf, T. C. Ralph, J. H. Shapiro, and S. Lloyd, "Gaussian quantum information," *Rev. Mod. Phys.*, vol. 84, pp. 621–669, May 2012. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.84.621

[51] M. Weiden, J. Kalloor, J. Kubiatowicz, E. Younis, and C. Iancu, "Wide quantum circuit optimization with topology aware synthesis," in *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)*, 2022, pp. 1–11. [Online]. Available: https://doi.org/10.1109/QCS56647.2022.00006

[52] Wikipedia contributors, "Polynomial hierarchy — Wikipedia, the free encyclopedia," 2024, [Online; accessed 3-April-2024]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Polynomial_hierarchy&oldid=1196336821

[53] X.-C. Wu, M. G. Davis, F. T. Chong, and C. Iancu, "Reoptimization of quantum circuits via hierarchical synthesis," in *2021 International Conference on Rebooting Computing (ICRC)*, 2021, pp. 35–46. [Online]. Available: https://10.1109/ICRC53822.2021.00016

[54] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE, "Berkeley quantum synthesis toolkit (bqskit) v1," 4 2021. [Online]. Available: https://www.osti.gov//servlets/purl/1785933

[55] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2021, pp. 232–243. [Online]. Available: https://doi.org/10.1109/QCE52317.2021.00041

[56] H. Zhang, A. Wu, Y. Wang, G. Li, H. Shapourian, A. Shabani, and Y. Ding, "Oneq: A compilation framework for photonic one-way quantum computation," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[57] H.-S. Zhong, Y.-H. Deng, J. Qin, H. Wang, M.-C. Chen, L.-C. Peng, Y.-H. Luo, D. Wu, S.-Q. Gong, H. Su, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, J. J. Renema, C.-Y. Lu, and J.-W. Pan, "Phase-programmable gaussian boson sampling using stimulated squeezed light," *Phys. Rev. Lett.*, vol. 127, p. 180502, Oct 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.127.180502

[58] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, and J.-W. Pan, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.abe8770