Quantum-assisted quantum compiling

Sumeet Khatri,1,2,∗ Ryan LaRose,1,3,∗ Alexander Poremba,1,∗ Lukasz Cincio,1 Andrew T. Sornborger,4 and Patrick J. Coles1

1Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.
2Hearne Institute for Theoretical Physics, Department of Physics and Astronomy, Louisiana State University, Baton Rouge, Louisiana 70803, USA.
3Department of Computational Mathematics, Science, and Engineering & Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48823, USA.
4Information Sciences, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.

Compiling quantum algorithms for near-term quantum computers (accounting for connectivity and native gate alphabets) is a major challenge that has received significant attention both by industry and academia. Avoiding the exponential overhead of classical simulation of quantum dynamics will allow compilation of larger algorithms, and a strategy for this is to evaluate an algorithm’s cost on a quantum computer. To this end, we propose quantum-assisted quantum compiling (QAQC). In QAQC, we use the Hilbert-Schmidt inner product between a target unitary $U$ and a trainable unitary $V$ as the cost function to be evaluated on the quantum computer. We introduce two circuits for evaluating this cost. One circuit computes $\langle Tr(V^*U) \rangle$, and we use this circuit for gradient-free compiling. Our other circuit gives $\text{Tr}(V^*U)$ and is a generalization of the Power of One Qubit that we call the Power of Two Qubits. We use this circuit for gradient-based compiling. As a demonstration of QAQC, we compile various one-qubit gates on IBM’s and Rigetti’s quantum computers into their respective native gate alphabets. Future applications of QAQC include algorithm depth compression, black-box compiling, noise mitigation, and benchmarking.

I. INTRODUCTION

Factoring [1], approximate optimization [2], and simulation of quantum systems [3] are some of the applications for which quantum computers have been predicted to provide speedups over classical computers. Consequently, the prospect of large-scale quantum computers has generated interest from various sectors, such as the financial and pharmaceutical industries. Currently available quantum computers are not large-scale but rather have been called noisy intermediate-scale quantum (NISQ) computers [4]. A proof-of-principle demonstration of quantum supremacy with a NISQ device may be coming soon [5, 6]. Nevertheless, demonstrating the practical utility of NISQ computers appears to be a more difficult task.

While improvements to NISQ hardware are continuously being made by experimentalists, quantum computing theorists can contribute to the utility of NISQ devices by developing software. This software would aim to adapt textbook quantum algorithms (e.g., for factoring or quantum simulation) to NISQ constraints. NISQ constraints include: (1) limited numbers of qubits, (2) limited connectivity between qubits, (3) restricted (hardware-specific) gate alphabets, and (4) limited circuit depth due to noise. Algorithms adapted to these constraints will likely look dramatically different from their textbook counterparts.

These constraints have increased the importance of the field of quantum compiling. In classical computing, a compiler is a program that converts instructions into assembly language so that they can be read and executed by a computer. Similarly, a quantum compiler would take a high-level algorithm and convert it into a lower-level form that could be executed on a NISQ device. Already, a large body of literature exists on classical approaches for quantum compiling, e.g., using temporal planning [7, 8], machine learning [9], and other techniques [10–17].

A recent exciting idea is to use quantum computers themselves to train parametrized quantum circuits, as proposed in Refs. [2, 18–22]. The cost function to be minimized essentially defines the application. For example, in the variational quantum eigensolver (VQE) [18] and the quantum approximate optimization algorithm (QAOA) [2], the application is ground state preparation, and hence the cost is the expectation value of the associated Hamiltonian. Another example is training error-correcting codes [19], where the cost is the average code fidelity. In light of these works, it is natural to ask: what is the relevant cost function for the application of quantum compiling?

In this work, we introduce quantum-assisted quantum compiling (QAQC, pronounced “Quack”). The goal of QAQC is to compile a (possibly unknown) target unitary to a trainable quantum gate sequence. The cost function we use is given by the Hilbert-Schmidt inner product between the target unitary and the trainable gate sequence. A key feature of QAQC is the fact that the cost is computed directly on the quantum computer. This leads to an exponential speedup (in the number of qubits involved in the gate sequence) over classical methods to compute the cost, since classical simulation of quantum dynamics is exponentially slower than quantum simulation. Conse-

* The first three authors contributed equally to this work.
quently, one should be able to optimally compile large-
scale gate sequences using QAQC, whereas classical ap-
proaches to optimal quantum compiling will be limited
to smaller gate sequences [23].

The circuit used to evaluate the cost in QAQC can
itself be viewed as a quantum algorithm. This cost-
evaluation circuit should ideally have the shortest pos-
sible depth, since noise in NISQ hardware will lead to in-
accurate cost values for deep circuits. Our key technical
contribution is to present two novel short-depth circuits
for cost evaluation. We propose using one of these cir-
cuits for gradient-free QAQC (where the circuit evaluates
the cost only) and the other for gradient-based QAQC
(where the circuit evaluates the cost and the gradient of
the cost).

Both of our circuits compute the Hilbert-Schmidt in-
eral product between a target unitary $U$ and a trainable
unitary $V$,

$$
\langle V, U \rangle = \text{Tr}(V^\dagger U).
$$

Our first circuit, which we call the Hilbert-Schmidt Test,
computes the magnitude of this inner product, $|\langle V, U \rangle|$. It
achieves short depth by avoiding implementing con-
trolled versions of $U$ and $V$, and by implementing $U$ and
$V$ in parallel.

Our second circuit computes the real and imaginary
parts of $\langle V, U \rangle$. It is closely related to the Power of One
Qubit (POOQ) [24], a circuit that computes the trace of a
unitary $U$ by implementing the controlled-$U$ gate with a
single ancilla. In fact, it directly generalizes the POOQ,
using two ancillas, and hence we call it the Power of Two
Qubits (POTQ). Although the POTQ requires controlled
versions of $U$ and $V$, a key feature that keeps the depth
short is that these controlled gates are performed in par-
allel. Interestingly, setting $V$ to the identity in POTQ
recovers the original POOQ.

When incorporated into QAQC, both of our circuits
provide an exponential speedup over quantum compiling
on a classical computer. Below we give detailed descrip-
tions of our gradient-free and gradient-based methods for
QAQC, which, respectively, employ our Hilbert-Schmidt
Test and our POTQ circuits. Our QAQC method for
training a gate sequence $V$ involves optimizing over both
the structure of the gate sequence (types of gates and
their locations) as well as the continuous internal param-
eters inside the gates. Hence, QAQC involves solving a
hybrid discrete-continuous optimization problem.

As a proof-of-principle, we implement our gradient-free
QAQC on both IBM’s and Rigetti’s quantum computers,
and we compile various one-qubit gates to the native gate
alphabets used by these hardwares. To our knowledge,
this is the first compilation of a target unitary with cost
evaluation on actual NISQ hardware. In addition, we
compile multi-qubit gates using a simulator with both
our gradient-free and gradient-based methods. Although
the noise level of current NISQ hardware prevents us from
compiling multi-qubit gates with our QAQC method, we
are optimistic that slight improvements in the noise will
enable this application.

In what follows, we first discuss several applications of
interest for QAQC. Section III presents our main results:
our short-depth circuits for cost evaluation on a quan-
tum computer. Section IV outlines our gradient-free and
gradient-based methods for QAQC. Finally, in Sec. V
and VI, respectively, we discuss the implementations of
our compiling methods on quantum hardware and on a
simulator.
II. APPLICATIONS OF QAQC

Figure 1 illustrates four potential applications of QAQC. Suppose that there exists a quantum algorithm to perform some task, but its associated gate sequence is longer than desired. As shown in Fig. 1(a), it is possible to use QAQC to shorten the gate sequence by accounting for the NISQ constraints of the specific computer. This depth compression goes beyond the capabilities of classical compilers.

As a simple example, consider the problem of simulating the dynamics of a given quantum system with an unknown Hamiltonian \( \mathcal{H} \) (via \( e^{-i\mathcal{H}t} \)) on a quantum computer. We call this problem black-box uploading because by simulating the black-box, i.e., the unitary \( e^{-i\mathcal{H}t} \), we can “upload” the unitary onto the quantum computer. This scenario is depicted in Fig. 1(b). QAQC could be used to convert an analog black-box unitary into a gate sequence on a digital quantum computer.

We emphasize that the unitary \( V^* \) is implemented on the quantum computer, not \( V \) itself.

Finally, we highlight two additional applications that are the opposites of each other. These two applications can be exploited when the quantum computer has some pristine qubits (qubits with low noise) and some noisy qubits.

Consider Fig. 1(c). Here, the goal is to implement a CNOT gate on two noisy qubits. Due to the noise, we need to implement a true CNOT, i.e., a CNOT surrounded by one-qubit unitaries. QAQC can be used to learn the parameters in these one-qubit unitaries. By choosing the target unitary \( U \) to be a CNOT on a pristine (i.e., noiseless) pair of qubits, it is possible to learn the CNOT and then apply the noisy qubits in order to effectively implement a CNOT. We call this application noise-tailored algorithms, since the learned algorithms are robust to the noise process on the noisy qubits.

We emphasize that the unitary \( V^* \) is implemented on the quantum computer, not \( V \) itself.

The third and final step is to measure in the Bell basis. This corresponds to undoing the unitaries (the CNOTs and Hadamards) used to prepare \( |\Phi^+\rangle \) and then measuring in the standard basis. At the end, we are only interested in estimating a single probability: the probability for the Bell-basis measurement to give the \( |\Phi^+\rangle \) outcome, which corresponds to the all-zeros outcome in the standard basis. The amplitude associated with this probability is

\[
\langle \Phi^+ | U \otimes V^* | \Phi^+ \rangle = \frac{1}{\sqrt{d}} \text{Tr}(V^\dagger U). \tag{5}
\]

To obtain the first equality we used the ricochet property:

\[
1 \otimes X |\Phi^+\rangle = X^T \otimes 1 |\Phi^+\rangle , \tag{6}
\]

which holds for any operator \( X \) acting on a \( d \)-dimensional space. The probability of the \( |\Phi^+\rangle \) outcome is then the absolute square of the amplitude, i.e., \( (1/d^2)|\text{Tr}(V^\dagger U)|^2 \).

We note that there are asymptotically \( n \) qubits, and the goal is to determine what the equivalent benchmarking. Here, the unitary

\[
|\Phi^+\rangle = \frac{1}{\sqrt{d}} \sum_j |j\rangle_A \otimes |j\rangle_B , \tag{2}
\]

where \( j = (j_1, j_2, \ldots, j_n) \) is a vector index where each component \( j_k \) is chosen from \{0, 1\}. The first two gates in Fig. 2—the Hadamard gates and the CNOT gates (which are performed in parallel when on distinct qubits) — create the \( |\Phi^+\rangle \) state.

The second step is to act with \( U \) on system \( A \) and with \( V^* \) on system \( B \). (\( V^* \) is the complex conjugate of \( V \), where the complex conjugate is taken in the standard basis.) Note that these two gates are performed in parallel. This gives the state

\[
(U \otimes V^*)|\Phi^+\rangle = \frac{1}{\sqrt{d}} \sum_j U|j\rangle_A \otimes V^*|j\rangle_B . \tag{3}
\]

III. OUR CIRCUITS

Finally, we highlight two additional applications that are the opposites of each other. These two applications can be exploited when the quantum computer has some pristine qubits (qubits with low noise) and some noisy qubits.

Consider Fig. 1(c). Here, the goal is to implement a CNOT gate on two noisy qubits. Due to the noise, we need to implement a true CNOT, i.e., a CNOT surrounded by one-qubit unitaries. QAQC can be used to learn the parameters in these one-qubit unitaries. By choosing the target unitary \( U \) to be a CNOT on a pristine (i.e., noiseless) pair of qubits, it is possible to learn the CNOT and then apply the noisy qubits in order to effectively implement a CNOT. We call this application noise-tailored algorithms, since the learned algorithms are robust to the noise process on the noisy qubits.

Figure 1(d) depicts the opposite process, which is benchmarking. Here, the unitary \( U \) acts on a noisy set of qubits, and the goal is to determine what the equivalent unitary \( V \) would be if it were implemented on a pristine set of qubits. This essentially corresponds to learning the noise model, i.e., benchmarking the noisy qubits.
Hilbert-Schmidt inner product between $U$ and $V$. We therefore call the circuit in Fig. 2 the Hilbert-Schmidt Test (HST).

Consider the depth of this circuit. Let $D(G)$ denote the depth of a gate sequence $G$ for a quantum computer whose native gate alphabet includes the CNOT gate and the set of all one-qubit gates. Then, for the HST, we have

$$D(\text{HST}) = 4 + \max\{D(U), D(V^\dagger)\}. \quad (7)$$

The first term of 4 is associated with the Hadamards and CNOTs in Fig. 2, and this term is negligible when the depth of $U$ or $V^\dagger$ is large. The second term results from the fact that $U$ and $V^\dagger$ are performed in parallel. Hence, whichever unitary, $U$ or $V^\dagger$, has the larger depth will determine the overall depth of the HST.

### B. The Power of Two Qubits

While the HST only gives the magnitude of $\langle V, U \rangle$, we now consider a circuit that gives the real and imaginary parts. Before we discuss this circuit, let us review the Power of One Qubit (POOQ) [24], shown in Fig. 3(a), which is a circuit for computing the trace of a $d$-dimensional unitary $U$. This circuit acts on a $d$-dimensional system $A$, initially in the maximally mixed state, $1/d$, and a single-qubit ancilla $Q$ initially in the $|0\rangle$ state. After applying a Hadamard gate to $Q$ and a controlled-$U$ gate to $QA$ (with $Q$ the control system), the reduced density matrix $\rho_Q$ has its off-diagonal elements proportional to $\Tr(U)$. Hence, one can measure $Q$ in the $X$ and $Y$ bases, respectively, to read off the real and imaginary parts of $\Tr(U)$.

Our circuit for computing $\langle V, U \rangle$ generalizes the POOQ and is called the Power of Two Qubits (POTQ), depicted in Fig. 3(b). As the name suggests, the POTQ employs two single-qubit ancillas, $Q$ and $Q'$, each initially in the $|0\rangle$ state. In addition, two $d$-dimensional systems, $A$ and $B$, are initially prepared in the Bell state $|\Phi^+\rangle$ defined in Eq. (2). (Although not shown in Fig. 3(b), this Bell state is prepared with a depth-two circuit, as shown in Fig. 2.)

The first step in the POTQ is to prepare the two-qubit maximally entangled state $\frac{1}{\sqrt{2}}(|0\rangle|0\rangle + |1\rangle|1\rangle)$ between $Q$ and $Q'$, using the Hadamard and CNOT gates as shown in Fig. 3(b). The second step is to apply a controlled-$U$ gate between $Q$ and $A$ (with $Q$ the control system). In parallel with this gate, the anticontrolled-$V^T$ gate is applied to $Q'B$, with $Q'$ the control system, where anti-controlled means that the roles of the $|0\rangle$ and $|1\rangle$ states on the control system are reversed. This results in the state:

$$\frac{1}{\sqrt{2}}(|0\rangle_Q'|0\rangle_Q' (U \otimes 1_B)|\Phi^+\rangle$$
$$+ |1\rangle_Q'|1\rangle_Q' (1_A \otimes V^T)|\Phi^+\rangle)$$
$$= \frac{1}{\sqrt{2}}(|0\rangle_Q'|0\rangle_Q' (U \otimes 1_B)|\Phi^+\rangle$$
$$+ |1\rangle_Q'|1\rangle_Q' (V \otimes 1_B)|\Phi^+\rangle), \quad (8)$$

where to obtain the equality we used the ricochet property in Eq. (6). As in the HST, note that $V$ itself is not implemented. In this case, its transpose is implemented.

Finally, a CNOT gate is applied to $QQ'$, with $Q$ the control system. This results in the reduced state on $Q$ being

$$\rho_Q = \frac{1}{2} |0\rangle\langle 0| + \Tr(V^T U)|0\rangle\langle 1|$$
$$+ \Tr(U^T V)|1\rangle\langle 0| + |1\rangle\langle 1|. \quad (9)$$

By inspection of $\rho_Q$, one can see that measuring $Q$ in the $X$ and $Y$ bases, respectively, gives the real and imaginary parts of $\Tr(V^T U)$.

Interestingly, if we set $V$ to the identity in the POTQ, then since the CNOT gate commutes with the controlled-$U$ gate and the reduced state of $|\Phi^+\rangle$ is the maximally mixed state $1/d$, we recover the POOQ. The POTQ is therefore a generalization of the POOQ.

Note that while the POOQ can also be used to determine $\Tr(V^T U)$, the POTQ has the advantage that the controlled gates for $U$ and $V$ can be executed in parallel, while in the POOQ they would have to be executed in series. This makes the POTQ better suited for NISQ devices, where short depth is crucial. Consider the depth of the POTQ. Denoting the controlled-$U$ and the anticontrolled-$V^T$ as $C_U$ and $C_{V^T}$ respectively, the overall depth is

$$D(\text{POTQ}) = 4 + \max\{D(C_U), D(C_{V^T})\} \quad (10)$$

Note the similarity here to Eq. (7). The overall depth is essentially determined by whichever controlled gate has the largest depth.
IV. OUR OPTIMIZATION METHOD

In this section, we give a detailed description of our optimization method for QAQC. First, we discuss how we parameterize the trainable unitary $V$ in terms of both discrete and continuous parameters. Then we present our two different approaches to optimizing over the continuous parameters: a gradient-free approach (which employs the circuit in Fig. 2) and a gradient-based approach (which employs the circuit in Fig. 3).

A. Discrete and continuous parameters

Consider an alphabet $\mathcal{A} = \{G_k(\alpha)\}_k$ of gates $G_k(\alpha)$ that are native to the quantum computer of interest. Here, $\alpha \in \mathbb{R}$ is a continuous parameter, and $k$ is a discrete parameter that identifies the type of gate and which qubits it acts on. For a given quantum computer, the problem of compiling $U$ to a gate sequence of length $L$ is to determine

$$ (\alpha_{\text{opt}}, k_{\text{opt}}) := \arg\min_{(\alpha, k)} C(U, V_k(\alpha)), $$

(11)

where

$$ V_k(\alpha) = G_{k_L}(\alpha_L)G_{k_{L-1}}(\alpha_{L-1}) \cdots G_k(\alpha_1), $$

(12)

is the trainable unitary. Here, $V_k(\alpha)$ is a function of the sequence $k = (k_1, \ldots, k_L)$ of parameters describing which gates from the native gate set are used and of the continuous parameters $\alpha = (\alpha_1, \ldots, \alpha_L)$ associated with each gate. The function $C(U, V_k(\alpha))$ is the cost, which quantifies how close the trained unitary is to the target unitary.

The optimization in (11) contains two parts: discrete optimization over the finite set of gate structures parameterized by $k$, and continuous optimization over the parameters $\alpha$ characterizing the gates within the structure. Our quantum-classical hybrid strategy to perform the optimization in (11) is illustrated in Fig. 4. As the set of gate structures grows exponentially with the number of gates $L$, a brute force search over all gate structures in order to obtain the best one is intractable in general. To efficiently search through this exponentially large space, we adopt an approach based on simulated annealing. (An alternative approach is genetic optimization, which has been implemented previously to classically optimize quantum gate sequences [28].)

Our simulated annealing approach starts with a random gate structure, then performs continuous optimization over the parameters $\alpha$ that characterize the gates in order to minimize the cost function. We then perform a structure update that involves randomly replacing a subset of gates in the sequence with new gates and re-optimizing the cost function over the continuous parameters $\alpha$. If this structure change produces a lower cost,
then we accept the change. If the cost increases, then we accept the change with probability decreasing exponentially in the magnitude of the cost difference. We iterate this procedure until the cost is (sufficiently close to) zero or until a maximum number of iterations is reached.

To perform the continuous optimization over the gate parameters, we use both a gradient-free approach and a gradient-based approach. Our gradient-free approach is based on a cost function defined by the Hilbert-Schmidt Test, while our gradient-based approach uses a cost function defined by the Power of Two Qubits.

### B. Gradient-free optimization

In the gradient-free approach to optimizing over the continuous gate parameters $\alpha$, we take as our cost function

$$C_{GF}(U, V_k(\alpha)) := 1 - \frac{1}{d} |\text{Tr}(V_k(\alpha) U)|,$$

(13)
and we compute this using the Hilbert-Schmidt Test as illustrated in Fig. 2 and described in Sec. IIIA. Note that for any two unitaries $U$ and $V$, $C_{GF}(U, V) = 0$ if and only if $U$ and $V$ differ by a global phase factor, i.e., $V = e^{i\varphi}U$ for some $\varphi \in \mathbb{R}$. By minimizing $C_{GF}$, we thus learn an equivalent unitary $V$ up to global phase.

---

**Algorithm 1: Gradient-free Continuous Optimization for QAQC**

**Input:** Unitary $U$ to be compiled; trainable unitary $V_k(\alpha)$ of a given structure; error tolerance $\text{tol} \in (0, 1)$; maximum number of iterations $N$; sample precision $\epsilon > 0$.

**Output:** Parameters $\alpha_{\text{opt}}$ such that at best

$$C_{GF}(U, V_k(\alpha_{\text{opt}})) \leq \text{tol}.$$

**Init:** $\alpha_{\text{opt}} \leftarrow 0$; $\text{cost} \leftarrow 1$

1. repeat

   2. choose an initial parameter $\alpha^{(0)}$ at random;

   3. run $\text{gp} \_ \text{minimize}$ with $\alpha^{(0)}$ as input and $\alpha_{\text{min}}$ as output; whenever the cost is called upon for some $\alpha$, run the HST on $V_k(\alpha)^*$ and $U$ approximately $1/e^2$ times to estimate the cost $C_{GF}(U, V_k(\alpha))$

   4. if $\text{cost} \geq C_{GF}(U, V_k(\alpha_{\text{min}}))$ then

   5. $\text{cost} \leftarrow C_{GF}(U, V_k(\alpha_{\text{min}})); \alpha_{\text{opt}} \leftarrow \alpha_{\text{min}}$

6. until $\text{cost} \leq \text{tol}$, at most $N$ times.

7. return $\alpha_{\text{opt}}, \text{cost}$.

---

For a given set of gate structure parameters $k$, the calculation of the cost on a quantum computer (as well as on a simulator) is affected by the fact that, due to finite sampling, the Hilbert-Schmidt Test allows us to obtain only an estimate of the magnitude of the Hilbert-Schmidt inner product. Noise within the quantum computer itself also affects the calculation of the cost. Therefore, in order to perform gradient-free optimization over the continuous internal gate parameters $\alpha$, we make use of stochastic optimization techniques that are designed to optimize noisy functions. Specifically, we make use of the $\text{gp} \_ \text{minimize}$ routine in the scikit-optimize Python library [29], which is a gradient-free optimization routine that performs Bayesian optimization using Gaussian processes [30, 31]. See Algorithm 1 for a general overview of the optimization procedure.

In Appendix A, Algorithm 3, we propose an alternative algorithm for gradient-free optimization that, on average, significantly reduces the number of calls to the quantum computer. As a result, it is more suitable for cloud computing under a queue submission system (e.g., IBM’s Quantum Experience). This algorithm performs a “multi-scale bisection” of the parameter space based on simulated annealing.

We emphasize that our approach to gradient-free optimization avoids the exponential overhead of evaluating the cost function classically, yet at the same time makes use of fast and efficient classical heuristics for optimization. In fact, using the Hilbert-Schmidt Test, Algorithm 1 requires only $O(1/e^2)$ calls to the quantum computer in order to evaluate the cost, where $\epsilon = 1/\sqrt{n_{\text{shots}}}$ is the sample precision, which is related to the number of samples $n_{\text{shots}}$ taken from the device.

As mentioned in Sec. IIIA, a subtle point about gradient-free compiling via the Hilbert-Schmidt Test is that, to compute $\text{Tr}(V_k(\alpha)^* U)$ in order to obtain the cost $C_{GF}(U, V_k(\alpha))$, the complex conjugate $V_k(\alpha)^*$ must be executed on the quantum computer, not $V_k(\alpha)$ itself. The complex conjugate of a unitary corresponding to a gate sequence can be obtained by taking the complex conjugate of each unitary in the gate sequence. However, if each gate in the sequence comes from a gate alphabet $\mathcal{A}$, it is possible that the complex conjugate of a gate in the sequence is not contained in the alphabet; for example, if $A = \{R_x(\pi/2), R_y(\theta)\}$, then the complex conjugate of $R_x(\pi/2)$, which is $R_x(-\pi/2)$, is not contained in $\mathcal{A}$. But the unitary $R_x(\pi)R_y(\pi/2)R_z(\pi)$ is equal (up to a global phase) to $R_z(-\pi/2)$. There are thus two ways to proceed when performing the compilation procedure: during the optimization over the continuous parameters, directly run the gate sequence corresponding to $V_k(\alpha)$, expressing it in terms of the native gate alphabet of the quantum computer, then at the end establish the complex conjugate of the optimal unitary as the unitary to which $U$ has been compiled. The latter would involve translating the complex conjugate of each gate in the optimal sequence into the native gate alphabet of the quantum computer. An alternative is to first take the complex conjugate $V_k(\alpha)^*$ by translating the complex conjugate of each gate in the sequence into the native gate alphabet, then execute the resulting sequence on the quantum computer. In each case, we allow for a small-scale classical compiler that can perform the simple translation of the complex conjugate of a gate sequence into the native gate alphabet of the quantum computer. Note that this small-scale classical compiler does not come with exponential overhead because it is only compiling one- and two-qubit gates.
Also, observe that if a gate alphabet is not closed under complex conjugation, then the depth of a gate sequence from that alphabet can increase by taking its complex conjugate. This is true for the example given above, in which the complex conjugate $R_x(-\pi/2)$ of $R_x(\pi/2)$ has a depth of three under the alphabet $A = \{R_x(\pi/2), R_z(\theta)\}$, while the original gate has a depth of only one. However, in general, note that the final depth increases by at most a constant factor relative to the original depth.

C. Gradient-based optimization

In this section, we introduce a gradient-based method for continuous optimization over the gate parameters $\alpha$ based on the POTQ algorithm for computing $\text{Tr}(V^\dagger U)$. While recent work on gradient descent continuous optimization has shown vast quantum speedups over classical variants [32–34], the majority of proposals still appear to be out of reach for implementations on NISQ devices, mainly due to their use of techniques, such as quantum state representation, the quantum Fourier transform, and the Grover search algorithm, which have high resource requirements. Instead, we focus on continuous optimization procedures that are feasible on current quantum computers and leave improvements to our algorithms as an open problem.

We define our cost function as the normalized Hilbert-Schmidt distance,

$$C_{GB}(U, V_k(\alpha)) := \frac{1}{2d} \| U - V_k(\alpha) \|^2_{HS} = 1 - \frac{1}{d} \text{Re}[\text{Tr}(V_k(\alpha)\dagger U)].$$

(14)

Note that for any two unitaries $U$ and $V$, the cost $C_{GB}(U, V)$ is zero if and only if $U = V$. Contrary to the gradient-free cost function $C_{GF}$, this cost function does not vanish if $U$ and $V$ differ only by a global phase. Indeed, if $V = e^{i\varphi}U$, then $C_{GB}(U, V) = 1 - \cos(\varphi)$.

In order to avoid exponential overhead, it is crucial for our gradient-based method to evaluate the gradient of the cost function on a quantum computer. Notice that the POTQ allows us to compute $\text{Re}[\text{Tr}(V_k(\alpha)\dagger U)]$ on a quantum computer. Similarly, since the gradient of the cost function $C_{GB}$ is given by

$$\nabla_{\alpha} C_{GB}(U, V_k(\alpha)) = -\frac{1}{d} \text{Re}[\text{Tr}(\nabla_{\alpha} V_k(\alpha)\dagger U)],$$

(15)

the POTQ can also be used to evaluate the gradient of the cost function on a quantum computer. Here, the gradient of $V_k(\alpha)$ is defined by

$$\nabla_{\alpha} V_k(\alpha) = \left( \frac{\partial V_k(\alpha)}{\partial \alpha_1}, \ldots, \frac{\partial V_k(\alpha)}{\partial \alpha_L} \right),$$

(16)

where the $(i, j)$ matrix element of the $\ell$-th component is

$$\left( \frac{\partial V_k(\alpha)}{\partial \alpha_{\ell}} \right)_{i,j} = \frac{\partial V_k(\alpha)}{\partial \alpha_{\ell}} |_{i,j}.$$

(17)

FIG. 5. (a) Any single-qubit gate $U$ can be decomposed into three elementary rotations (up to a global phase). Given appropriate parameters $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, $U$ can be written as $V(\alpha) = e^{-i\alpha_3 \sigma_z/2} e^{-i\alpha_2 \sigma_y/2} e^{-i\alpha_1 \sigma_x/2}$. (b) Any two-qubit gate $U_{AB}$ can be decomposed into three CNOT gates as well as 15 elementary single-qubit gates, where each unitary $U(\alpha(\ell))$ can be written as in (a). This decomposition is known to be optimal [35], i.e., it uses the least number of continuous parameters and CNOT gates. General universal quantum circuits for $n$-qubit gates are discussed in [36].

Evaluating the gradient on a quantum computer is possible due to the fact that any unitary gate can be decomposed into circuits in which only the single-qubit gates are parametrized. This is illustrated in Fig. 5. For example, consider the rotation gate $R_z(\alpha) = e^{-i\alpha \sigma_z/2}$, which is parametrized by the angle $\alpha$. Then, the derivative with respect to $\alpha$ can be written as

$$\partial_\alpha R_z(\alpha) = \frac{i}{2} \sigma_z R_z(\alpha),$$

(18)

which follows from the Taylor expansion of the exponent. This means that

$$\partial_\alpha C_{GB}(U, R_z(\alpha)) = -\frac{1}{d} \text{Re}[\text{Tr}(\partial_\alpha R_z(\alpha)\dagger U)] = -\frac{1}{2d} \text{Re}[\text{Tr}(\sigma_z R_z(\alpha)\dagger U)].$$

(19)

(20)

In general, then, to compute the derivative of the cost function with respect to the continuous parameters, we simply add the appropriate local Pauli gate as part of the POTQ. Once the gradient has been evaluated, it can be supplied to a classical gradient descent optimization routine.

Our gradient-based optimization procedure is outlined in Algorithm 2. Given an arbitrary unitary $U$ as input, Algorithm 2 compiles $U$ to a unitary $V_k(\alpha)_{\text{opt}}$ of a given structure $k$ that minimizes the cost $C_{GB}$. The gradient is evaluated with the POTQ circuit as a subroutine within a classical gradient-descent algorithm. The overall query complexity of Algorithm 2 is $O(NTL/\epsilon^2)$, where $\epsilon = 1/\sqrt{\text{shots}}$ is the sample precision, $N$ is the maximum number of repetitions over random initial parameters $\alpha^0$, $L$ is the dimension of the continuous parameter space of $\alpha$, and $T$ is the number of gradient descent iterations for a suitable learning rate $\eta > 0$. In order to improve convergence, it may also be useful to supply the quantum subroutines for computing the cost function and the
Algorithm 2: Gradient-based Continuous Optimization for QAQC

Input: Unitary $U$ to be compiled; a trainable circuit $V_k(\alpha)$ of a given structure, where $\alpha$ is a continuous circuit parameter of dimension $L$; maximum number of iterations $N$; error tolerance $\text{tol} \in (0, 1)$; learning rate $\eta > 0$; sample precision $\epsilon > 0$.

Output: Parameters $\alpha_{\text{opt}}$ such that at best $C_{\text{GF}}(U, V_k(\alpha_{\text{opt}})) \leq \text{tol}$.

1. $\alpha_{\text{opt}} \leftarrow 0; \text{cost} \leftarrow 1$
2. repeat
3.   for $\tau = 1, 2, \ldots, T$ do
4.     for $i = 1, 2, \ldots, L$ do
5.       run the POTQ on $\partial_{\alpha_i} V_k(\alpha^{(r-1)})^T$ and $U$ approximately $1/\epsilon^2$ times to estimate $\text{Re} \left( \text{Tr} \left[ \partial_{\alpha_i} V_k(\alpha^{(r-1)}) U \right] \right)$
6.     update $\alpha^{(r)} \leftarrow \alpha^{(r-1)} - \eta \nabla_{\alpha} C_{\text{GF}}(U, V_k(\alpha^{(r-1)}))$
7.     run the POTQ on $V_k(\alpha^{(r)})^T$ and $U$ approximately $1/\epsilon^2$ times to estimate the cost $C_{\text{GF}}(U, V_k(\alpha^{(r)}))$
8.     if $\text{cost} > C_{\text{GF}}(U, V_k(\alpha^{(r)}))$ then
9.       $\text{cost} \leftarrow C_{\text{GF}}(U, V_k(\alpha^{(r)})); \alpha_{\text{opt}} \leftarrow \alpha^{(r)}$
10. until $\text{cost} \leq \text{tol}$, at most $N$ times
11. return $\alpha_{\text{opt}}, \text{cost}$

In essence, our method discretizes the continuous parameter space of angles $\theta$ to perform the continuous optimization. These angles are selected uniformly over the unit circle and the grid spacing between them decreases in the number of iterations. See Appendix A for full details. If the cost of the new sequence is less than the cost of the previous sequence, then we accept the change. Otherwise, we accept the change with a probability that decreases exponentially in the magnitude of the difference in cost. This change in cost defines one iteration as shown in Fig. 6.

In Fig. 6(a), we show results for compiling single qubit gates on IBMQX4. All gates ($I, T, X, \text{and } H$) converge to a cost below 0.1, but no gate achieves a cost below our

V. IMPLEMENTATION ON QUANTUM HARDWARE

In this section, we present the results of executing our QAQC procedure described in Sec. IV on IBM’s and Rigetti’s quantum computers. In each case, we performed gradient-free continuous parameter optimization in order to minimize the cost function $C_{\text{GF}}$ in (13), evaluating this cost function on the quantum computer. In what follows, the depth of a gate sequence is defined relative to the native gate alphabet of the quantum computer used.

A. IBM’s quantum computers

Here we consider the 5-qubit IBMQX4 and the 16-qubit IBMQX5. For these quantum computers, the native gate set is

$$A_{\text{IBM}} = \{R_z(\pi/2), R_z(\theta), \text{CNOT}_{i,j}\}$$

where the single-qubit gates $R_z(\pi/2)$ and $R_z(\theta)$ can be performed on any qubit and the two-qubit CNOT gate can be performed between any two qubits allowed in the topology; see [39] for the topology of IBMQX4 and [40] for the topology of IBMQX5.

To compile a given unitary $U$, we use the general procedure outlined in Sec. IV. Specifically, our initial gate structure, given by $V_k(\alpha)$, is selected at random from the gate alphabet in (21). We then calculate the cost $C_{\text{GF}}(U, V_k(\alpha))$ by executing the Hilbert-Schmidt Test shown in Fig. 2 on the quantum computer. To perform the continuous parameter optimization over the angles $\theta$ of the $R_z$ gates, instead of employing the optimization procedure outlined in Sec. IVB, we make use of Algorithm 3 outlined in Appendix A. This method is designed to limit the number of objective function calls to the quantum computer, which is an important consideration when using queue-based quantum computers like IBMQX4 and IBMQX5 since these can entail a significant amount of idle time in the queue.

In essence, our method discretizes the continuous parameter space of angles $\theta$ to perform the continuous optimization. These angles are selected uniformly over the unit circle and the grid spacing between them decreases in the number of iterations. See Appendix A for full details. If the cost of the new sequence is less than the cost of the previous sequence, then we accept the change. Otherwise, we accept the change with a probability that decreases exponentially in the magnitude of the difference in cost. This change in cost defines one iteration as shown in Fig. 6.
achieve the lowest cost in the cost vs. depth plot in the optimal decompositions. The same decompositions also internal gate parameters.

We define an iteration to be one accepted update in gate results are shown in Fig. 6(c). As described in Algorithm 1, we define an iteration to be one accepted update in gate structure followed by a continuous optimization over the parameters \( \theta \) of the \( R_z \) gates using the gradient-free stochastic optimization technique described in Sec. IVB, Algorithm 1. We take the cost error tolerance (the parameter \( \text{tol} \) in Algorithm 1) to be \( 10^{-2} \), and for each run of the Hilbert-Schmidt Test, we let \( n_{\text{shots}} = 10,000 \) in order to estimate the cost. Our results are shown in Fig. 6(c). As described in Algorithm 1, we define an iteration to be one accepted update in gate structure followed by a continuous optimization over the internal gate parameters.

The gates compiled in Fig. 6(c) have the following optimal decompositions. The same decompositions also achieve the lowest cost in the cost vs. depth plot in the inset.

1. \( \mathbb{1} \) gate: \( R_z(\theta) \), with \( \theta \approx 0.01\pi \).
2. \( T \) gate: \( R_z(\theta) \), with \( \theta \approx 0.30\pi \).
3. \( X \) gate: \( R_x(\pi/2)R_z(\pi/2) \).
4. \( H \) gate: \( R_z(\theta_1)R_x(\pi/2)R_z(\theta_2) \), with \( \theta_1 = \theta_2 = 0.50\pi \).

In particular, note that \( R_x(\pi/2)R_z(\pi/2) \) is a textbook decomposition of the \( X \) gate, yet the cost is not zero.

Similarly, Fig. 6(b) shows results for compiling the same single-qubit gates as above on IBMQX5. The gate sequences have the same structure as listed above for IBMQX4. The optimal angles achieved are \( \theta = -0.03\pi \) for the \( \mathbb{1} \) gate and \( \theta = 0.23\pi \) for the \( T \) gate. The \( X \) gate compiles to \( R_z(\pi/2)R_x(\pi/2) \), and the Hadamard \( H \) compiles to \( R_x(\pi/2)R_z(\pi/2)R_x(\pi/2) \).

In our data collection, we performed on the order of 10 independent optimization runs for each target gate above. The standard deviations of the \( \theta \) angles were on the order of 0.05\( \pi \), and this can be viewed as the error bars on the average values quoted above.

**B. Rigetti’s quantum computer**

The native gate set of Rigetti’s 8Q-Agave 8-qubit quantum computer is

\[
A_{\text{Rigetti}} = \{ R_x(\pm \pi/2), R_z(\theta), \text{CZ}_{i,j} \} \tag{22}
\]

where the single-qubit gates \( R_x(\pm \pi/2) \) and \( R_z(\theta) \) can be performed on any qubit and the two-qubit CZ gate can be performed between any two qubits allowed in the topology; see [41] for the topology of the 8Q-Agave quantum computer.

As with the implementation on IBM’s quantum computers, for the implementation on Rigetti’s quantum computer we make use of the general procedure outlined in Sec. IV. Specifically, we perform random updates to the gate structure followed by continuous optimization over the parameters \( \theta \) of the \( R_z \) gates using the gradient-free stochastic optimization technique described in Sec. IVB, Algorithm 1. We take the cost error tolerance (the parameter \( \text{tol} \) in Algorithm 1) to be \( 10^{-2} \), and for each run of the Hilbert-Schmidt Test, we let \( n_{\text{shots}} = 10,000 \) to evaluate the cost for each run of the Hilbert-Schmidt Test. (b) Compiling on the IBMQX5 quantum computer, in which we again took \( n_{\text{shots}} = 8,000 \) to evaluate the cost for each run of the Hilbert-Schmidt Test. (c) Compiling on Rigetti’s 8Q-Agave quantum computer. For each run of the Hilbert-Schmidt Test to determine the cost, we took \( n_{\text{shots}} = 10,000 \).

1. \( \mathbb{1} \) gate: \( R_z(\theta) \), with \( \theta \approx 0 \).
2. \( T \) gate: \( R_z(\theta) \), with \( \theta \approx 0.342\pi \).
3. \( X \) gate: \( R_x(-\pi/2)R_z(-\pi/2) \).
4. \( H \) gate: \( R_z(\theta_1)R_x(\pi/2)R_z(\theta_2) \), with \( \theta_1 \approx 0.50\pi \) and \( \theta_2 \approx 0.49\pi \).

As with the results on IBM’s quantum computers, none of the gates achieve a cost less than \( 10^{-2} \), due to factors such as finite sampling, gate fidelity, decoherence, and readout error. In addition, similar to the IBM results, the standard deviations of the \( \theta \) angles here were on the
order of 0.05π, which can be viewed as the error bars on the average values (over 10 independent runs) quoted above.

C. Discussion

On both IBM’s and Rigetti’s hardware, we are able to successfully compile one-qubit gates with no a priori assumptions about gate structure or parameters. In principle, our approach extends to compiling larger unitaries $U$ with an exponential speedup over classical methods. In practice, limitations arise in this approach due to decoherence, gate infidelity, and readout error on NISQ computers.

These limitations are more pronounced as circuit depth increases. We therefore encounter significant performance loss for controlled unitaries as required in the POTQ. Consequently, we did not implement our gradient-based optimization method on current quantum devices, but we speculate that improvements to quantum hardware will enable this application.

Our gradient-free optimization method employs a shorter-depth circuit (the HST in Fig. 2), and hence we were able to implement it on NISQ hardware. Nevertheless, the imperfections in NISQ hardware do affect the accuracy of estimating the cost. A qualitative noise analysis of the HST is as follows. To compile a unitary $U$ acting on $n$ qubits, a circuit with $2n$ qubits is needed. Preparing the maximally-entangled state $|Φ^+\rangle$ in the first portion of the circuit requires $n$ CNOT gates, which are significantly noisier than one-qubit gates and propagate errors to other qubits through entanglement. In principle, all Hadamard and CNOT gates can be implemented in parallel, but on near-term devices this may not be the case. Additionally, due to limited connectivity of NISQ devices, it is generally not possible to directly implement CNOTs between arbitrary qubits. Instead, the CNOTs need to be “chained” between qubits that are connected, a procedure that can significantly increase the depth of the circuit.

The next level of the circuit involves implementing $U$ in the top $n$-qubit register and $V^*$ in the bottom $n$-qubit register. Here, the noise of the computer on $V^*$ is not necessarily undesirable since it could allow us to compile noise-tailored algorithms that counteract the noise of the specific computer, as described in Sec. II. Nevertheless, the depth of $V^*$ and/or $U$ essentially determines the overall circuit depth as noted in (7), and quantum coherence decays exponentially with the circuit depth. Hence, compiling larger gate sequences involves additional loss of coherence on NISQ computers.

The final level of the circuit involves making a Bell measurement on all qubits and is the reverse of the first part of the circuit. As such, the same noise analysis of the first portion of the circuit applies here. Readout errors can be significant on NISQ devices [42], and our circuit involves a number of measurements that scales linearly in the number of qubits. Hence, compiling larger unitaries can increase overall readout error.

For all of these reasons, we limit implementation on today’s quantum devices to one-qubit gates. This establishes a proof-of-principle demonstration of QAQC, and our approach will only improve as the engineering of quantum hardware improves. To demonstrate the applicability of QAQC to future devices, we now show examples for compiling two-qubit unitaries by running the Hilbert-Schmidt Test and the POTQ on simulators.

We also note that the performance of any cost function optimization method is likely to decrease when compiling unitaries over large numbers of qubits, particularly when using random initial guesses for the circuit structure. Recent results on barren plateaus in gradient-based optimization [43, 44] suggest that the probability of observing non-zero gradients tends to become exponentially small as a function of the number of qubits. Similar issues have been noted in classical deep learning [45]. While this is not an issue for the small circuits considered in this paper, as the number of qubits increases this becomes an increasingly important consideration. There may be ways around these problems, as suggested in [45]. For instance, recent work [46] shows that gradient descent with momentum (GDM) using an adaptive (multiplicative) integration step update, called resilient backpropagation (rProp), can help with convergence. But, this is an active research area and will likely be important to the success of NISQ devices and more generally to the success of quantum machine learning.

VI. IMPLEMENTATION ON SIMULATOR

In this section, we present our results on finding short-depth algorithms for single-qubit and two-qubit gates using a simulator. We use the gate alphabet

$$A = \{R_x(\pi/2), R_z(\theta), \text{CNOT}_{i,j}\},$$

which is the gate alphabet defined in Eq. (21) except with full connectivity between the qubits. We adopt the methods from Sec. IV B and IV C to perform the continuous-parameter optimization. The simulations are performed assuming perfect connectivity between the qubits, no gate errors, and no decoherence. Although we do not obtain the exponential advantages of running our algorithms on a quantum computer, these assumptions allow us to compile larger unitaries.

a. Gradient-free optimization. Using Rigetti’s quantum virtual machine [25], we compile the controlled-Hadamard (CH) gate, the CZ gate, the SWAP gate, and the two-qubit Fourier transform QFT$_2$ by adopting the continuous optimization procedure in Algorithm 1. We also compile the single-qubit gates $X$ and $H$. For each run of the Hilbert-Schmidt Test to determine the cost, we took $n_{\text{shots}} = 20,000$. Our results are shown in Fig. 7. For the SWAP gate, we find that circuits of depth one and two cannot achieve zero cost, but
There exists a circuit with depth three for which the cost vanishes. The circuit achieving this zero cost is the well-known decomposition of the SWAP gate into three CNOT gates. While our compilation procedure reproduces the known decomposition of the SWAP gate, it discovers a decomposition of both the CZ and the QFT₂ gates that differs from their conventional “textbook” decompositions, as shown in Fig. 7(c). In particular, these decompositions have shorter depths than the conventional decompositions when written in terms of the gate alphabet in (23).

b. Gradient-based optimization. We use IBM’s simulator [26] to compile a selection of single-qubit and two-qubit gates by performing the gradient-based optimization procedure in Algorithm 2. In order to improve convergence, we additionally supplied the gradient, as well as the cost function, to a stochastic minimizer from the Python library SciPy. For the single-qubit gates, we assume a fixed structure for the gate sequence according to the decomposition in Fig. 5(a), while for the two-qubit gates we assume a fixed structure for the gate sequence according to the decomposition in Fig. 5(b). We compile the T gate, X gate, Hadamard (H) gate, as well as the CNOT and CZ gates, all with \( n_{\text{shots}} = 10,000 \). The results are shown in Fig. 8. We note that increasing \( n_{\text{shots}} \) to higher orders of magnitude significantly reduces the sampling error and results in more stable convergence at the cost of an increase in runtime.

VII. CONCLUSIONS

Quantum compiling is crucial in the era of NISQ devices, where constraints on NISQ computers (such as limited connectivity, limited circuit depth, etc.) place severe restrictions on the quantum algorithms that can be implemented in practice. In this work, we presented a methodology for quantum compilation called quantum-assisted quantum compiling (QAQC), whereby a quantum computer provides an exponential speedup in evaluating the cost of a gate sequence, i.e., how well the gate sequence matches the target. In principle, QAQC should allow for the compiling of larger algorithms than standard classical methods for quantum compiling, due to this exponential speedup. As a proof-of-principle, we implemented QAQC on IBM’s and Rigetti’s quantum computers to compile various one-qubit gates to their native gate alphabets. To our knowledge, this is the first time NISQ hardware has been used to compile a target unitary. Current noise levels in the hardware prevented us from compiling multi-qubit gates, although we are optimistic that future noise reduction could enable larger scale QAQC.
Our main technical results were two short-depth circuits for computing the Hilbert-Schmidt inner product between a target unitary $U$ and a trainable unitary $V$. One of these circuits, which we call the Hilbert-Schmidt Test and describe in Sec. III A, computes the inner product magnitude, and we incorporated this circuit into our gradient-free QAQC method in Sec. IV B. The other circuit, which we call the Power of Two Qubits and describe in Sec. III B, computes the real and imaginary parts of the inner product, and we exploited this circuit for gradient-based QAQC in Sec. IV C. The latter circuit generalizes the famous Power of One Qubit [24] and hence is likely of interest to a broader community.

QAQC is a novel variational algorithm, similar to other well-known variational algorithms such as VQE [18] and QAOA [2]. Variational algorithms are likely to provide some of the first real applications of quantum computers in the NISQ era. In the case of QAQC, it is an algorithm that makes other algorithms more efficient to implement, via algorithm depth compression. The central application of our technique is thus to make quantum computers more useful.

ACKNOWLEDGMENTS

We thank IBM and Rigetti for providing access to their quantum computers. The views expressed in this paper are those of the authors and do not reflect those of IBM or Rigetti. SK, RL, and AP acknowledge support from the U.S. Department of Energy through a quantum computing program sponsored by the LANL Information Science & Technology Institute. RL acknowledges support from an Engineering Distinguished Fellowship through Michigan State University. LC was supported by the U.S. Department of Energy through the J. Robert Oppenheimer fellowship. ATS and PJC were supported by the LANL ASC Beyond Moore’s Law project.

[1] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[2] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. arXiv:1411.4028, 2012.

[3] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[4] John Preskill. Quantum computing in the NISQ era and beyond. arXiv:1801.00862, 2018.

[5] John Preskill. Quantum computing and the entanglement frontier. arXiv:1203.5813, 2012.

[6] C Neil, P Roushan, K Kechedzhi, S Boixo, SV Isakov, V Smelyanskiy, R Barends, B Burkett, Y Chen, Z Chen, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. arXiv:1709.06678, 2017.

[7] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.

[8] Kyle EC Booth, Minh Do, J. Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. arXiv:1803.06775, 2018.

[9] Lukasz Cincio, Yiğit Subaşı, Andrew T Sornborger, and Patrick J Coles. Learning the quantum algorithm for state overlap. arXiv:1803.04114, 2018.

[10] Dmitri Maslov, Gerhard W Dueck, D Michael Miller, and Camille Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, 2008.

[11] Austin G. Fowler. Constructing arbitrary Steane code single logical qubit fault-tolerant gates. *Quantum Information and Computation*, 11(9 & 10):867–873, 2011.

[12] Jeffrey Booth Jr. Quantum compiler optimizations. arXiv:1206.3348, 2012.

[13] Yunseong Nam, Neil J Ross, Yuan Su, Andrew Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. arXiv:1710.07345, 2017.

[14] Frederic T Chong, Diana Franklin, and Margaret Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180, 2017.

[15] Luke Heyron and Earl T Campbell. An efficient quantum compiler that reduces $T$ count. arXiv:1712.01557, 2017.

[16] Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 2018.

[17] Angelo Oddi and Riccardo Rasconi. Greedy randomized search for scalable compilation of quantum circuits. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 446–461. Springer, 2018.

[18] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alan Aspuru-Guzik, and Jeremy L. O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014.

[19] Peter D Johnson, Jonathan Romero, Jonathan Olson, Yudong Cao, and Alán Aspuru-Guzik. QVECTOR: an algorithm for device-tailored quantum error correction. arXiv:1711.02249, 2017.

[20] Marcello Benedetti, Delfina Garcia-Pintos, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. arXiv:1801.07688, 2018.

[21] Kostas Mitarai, Makoto Negoro, and Keisuke Kitagawa, Masahiro nd Fujii. Quantum circuit learning. arXiv:1803.00743, 2018.

[22] Guillaume Verdon, Jason Pye, and Michael Broughton. A Universal Training Algorithm for Quantum Deep Learning. arXiv:1806.09729, 2018.
23] We note that classical compilers may be applied to large-scale quantum algorithms, but they are limited to local compiling. We thus emphasize the distinction between translating the algorithm to the native alphabet with simple, local compiling and optimal compiling. Local compiling may reach partial optimization but in order to discover the shortest circuit one may need to use a holistic approach, where the entire algorithm is considered, which requires a quantum computer for compiling.

[24] E. Knill and R. Laflamme. Power of one bit of quantum information. Physical Review Letters, 81:5672–5675, 1998.

[25] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture. arXiv:1608.03355, 2016.

[26] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. arXiv:1707.03429, 2017.

[27] Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.

[28] Adrian Gepp and Phil Stocks. A review of procedures to evolve quantum algorithms. Genetic Programming and Evolvable Machines, 10(2):181–228, June 2009.

[29] Scikit-optimize. https://github.com/scikit-optimize/scikit-optimize, 2018.

[30] J. Mockus. On Bayesian methods for seeking the extremum. In Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974, pages 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[31] Michael A. Osborne, Roman Garnett, and Stephen J. Roberts. Gaussian processes for global optimization. In 3rd International Conference on Learning and Intelligent Optimization (LION3) 2009, 2009.

[32] Patrick Rebentrost, Maria Schuld, Leonard Wossnig, and Seth Lloyd. Quantum gradient descent and newton’s method for constrained polynomial optimization. arXiv:1612.01789, 2016.

[33] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. arXiv:1704.04992, 2017.

[34] Andras Gilyen, Srinivasan Arunachalam, and Nathan Wiebe. Optimizing quantum optimization algorithms via faster quantum gradient computation. arXiv:1711.00165, 2017.

[35] Farrokh Vatan and Colin Williams. Optimal quantum circuits for general two-qubit gates. Physical Review A, 69:032315, Mar 2004.

[36] P. B. M Sousa and R. V. Ramos. Universal quantum circuit for n-qubit quantum gate: A programmable quantum gate. Quantum Information and Computation, 7(3):228–242, March 2007.

[37] Scipy optimization and root finding. https://docs.scipy.org/doc/scipy/reference/optimize.html, 2018.

[38] Xiao-Qi Zhou, Timothy C. Ralph, Pruet Kalasuwan, Mian Zhang, Alberto Peruzzo, Benjamin P. Lanyon, and Jeremy L. O’Brien. Adding control to arbitrary unknown quantum operations. Nature Communications, 2(413), 2011.

[39] IBM Q 5 Tenerife backend specification. https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/tenerife/V1, 2018.

[40] IBM Q 16 Rueschlikon backend specification. https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/rueschlikon/V1, 2018.

[41] Rigetti 8Q-Agave specification v.2.0.0.dev0. http://docs.rigetti.com/en/latest/qpu.html, 2018.

[42] Abbasin Kandala, Kristan Temme, Antonio D. Corcoles, Antonio Mezzacapo, Jerry M. Chow, and Jay M. Gambetta. Extending the computational reach of a noisy superconducting quantum processor. arXiv:1805.04492, 2018.

[43] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. arXiv:1803.11173, 2018.

[44] Alexandre G.R. Day, Marin Bukov, Phillip Weinberg, Pankaj Mehta, and Dries Sels. The glassy phase of optimal quantum control. arXiv:1803.10856, 2018.

[45] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. AISTATS, pages 249–256, 2010.

[46] M Benedetti, D Garcia-Pintos, Y S Nam, and A Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. arXiv:1801.07686, 2018.

Appendix A: Alternative method for gradient-free optimization

An alternative approach to performing the continuous parameter optimization described in Sec. IVB is outlined in Algorithm 3. We implement this method, which is designed to limit the number of times the objective function is evaluated, specifically for the hardware of IBM because the queue submission system can require a significant time to make many calls to the quantum computer. In the discrete optimization procedure in Algorithm 3, we start with four angles spread uniformly in the interval $[0,2\pi)$—namely $0, \pi/2$, and $\pi/2$. This significantly reduces the size of the search space and allows us to get close to, or find exactly, an optimal gate sequence. Once the optimal structure is reached from this step, we then bisect the angles for each frame change $R_2(\alpha)$ by evaluating the overlap with a new circuit containing $R_2(\alpha \pm \pi/2^t)$, where $t = 1, 2, \ldots, t_{\text{max}}$ is determined by the iteration in the procedure. Although we do not explore all angles in the interval, the runtime is logarithmically faster than a continuous search due to the bisection procedure. An additional advantage of this approach is that many gates have angles that are simple fractions of $\pi$, e.g., $T = R_2(\pi/4)$ and $H = R_2(-\pi/2)R_4(-\pi/2)R_2(-\pi/2)$. In a noiseless environment, the two steps above are sufficient. On actual devices, we implement a third step of stochastic optimization by evaluating the cost for the new circuit with each frame change $R_2(\alpha)$ replaced by $R_2(\alpha \pm \Delta(t))$ for some small value $\Delta(t) \ll 1$ decreasing monotonically with the iteration $t$. This allows us to compile for a given device by accounting for noise and gate errors. This can be thought of as a “fine-grained” angular optimization in contrast to the
Algorithm 3: Gradient-free Optimization using Bisection for QAQC

**Input:** Unitary $U$ to be compiled; trainable circuit $V_k$ of a given structure and gate alphabet $\mathcal{A}$; error tolerance $\text{tol} \in (0, 1)$; maximum number of iterations $N$; maximum number of bisections $t_{\text{max}}$ of the unit circle; sample precision $\epsilon > 0$.

**Output:** Parameters $\alpha_{\text{opt}}$ that minimize the cost $C_{\text{GF}}$ at best to within $\text{tol}$ of zero, i.e. where $C_{\text{GF}}(V_k(\alpha_{\text{opt}}), U) \leq \text{tol}$.

**Init:** Restrict all gates in $\mathcal{A}$ with continuous parameters to discrete angles in the set $\Omega_0 = \{0, \pi/2, \pi, 3\pi/2\}$; $\alpha_{\text{opt}} \leftarrow 0$; $\text{cost} \leftarrow 1$;

for $t = 1, 2, \ldots, t_{\text{max}}$ do
  repeat
    anneal over all possible bisected angles in the set $\Omega_t := \{\alpha \pm \pi/2^{t+1} | \alpha \in \Omega_0 \} \cup \Omega_{t-1}$;
    whenever the cost is called upon some $\alpha \in \Omega_t$, run the HST on $V_k(\alpha)^*$ and $U$ approximately $1/\epsilon^2$ times to estimate the cost $C_{\text{GF}}(V_k(\alpha), U)$;
    if $\text{cost} \geq C_{\text{GF}}(V_k(\alpha), U)$ then $\text{cost} \leftarrow C_{\text{GF}}(V_k(\alpha), U)$;
  until $\text{cost} \leq \text{tol}$ at most $N$ times.
repeat
  minimize the cost over all small continuous increments $\Delta(t) \ll 1$ within the set of bisected angles $\Omega_t$; whenever the cost is called on some $\alpha + \Delta(t)$, for $\alpha \in \Omega_t$, run the HST on $V_k(\alpha + \Delta(t))^*$ and $U$ approximately $1/\epsilon^2$ times to estimate the cost $C_{\text{GF}}(V_k(\alpha + \Delta(t)), U)$;
  if $\text{cost} \geq C_{\text{GF}}(V_k(\alpha + \Delta(t)), U)$ then $\text{cost} \leftarrow C_{\text{GF}}(V_k(\alpha + \Delta(t)), U)$; $\alpha_{\text{opt}} \leftarrow \alpha + \Delta(t)$;
  until $\text{cost} \leq \text{tol}$ at most $N$ times.
return $\alpha_{\text{opt}}$, $\text{cost}$.