Learning the quantum algorithm for state overlap

Lukasz Cincio,[1] Yiğit Subaşı,[1] Andrew T. Sornborger,[2] and Patrick J. Coles[1]

[1] *Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.*
[2] *Information Sciences, Los Alamos National Laboratory, Los Alamos, NM 87545, USA.*

Short-depth algorithms are crucial for reducing computational error on near-term quantum computers, for which decoherence and gate infidelity remain important issues. Here we present a machine-learning approach for discovering such algorithms. We apply our method to a ubiquitous primitive: computing the overlap $\mathrm{Tr}(\rho\sigma)$ between two quantum states ρ and σ. The standard algorithm for this task, known as the Swap Test, is used in many applications such as quantum support vector machines, and, when specialized to $\rho = \sigma$, quantifies the Renyi entanglement. Here, we find algorithms that have shorter depths than the Swap Test, including one that has constant depth (independent of problem size). Furthermore, we apply our approach to the hardware-specific connectivity and gate alphabets used by Rigetti's and IBM's quantum computers and demonstrate that the shorter algorithms that we derive significantly reduce the error - compared to the Swap Test - on these computers.

I. INTRODUCTION

Quantum supremacy [1] may be coming soon [2]. While it is an exciting time for quantum computing, decoherence and gate fidelity continue to be important issues [3]. Ultimately these issues limit the depth of algorithms that can be implemented on near-term quantum computers (NTQCs) and increase the computational error for short-depth algorithms. Furthermore, NTQCs do not currently have enough qubits to fully leverage the benefit of quantum error-correcting codes [4, 5]. This highlights the need for general methods to reduce the depth of quantum algorithms in order to avoid the accumulation of errors.

Analytical efforts to find short-depth algorithms face several challenges. First, quantum algorithms are fairly non-intuitive to classically trained minds. Second, actual NTQCs may not be fully connected. Third, different NTQCs use different fundamental gate sets. It may not be obvious how to optimize algorithms for a given connectivity and a given gate set. This motivates the idea of an automated approach for discovering and optimizing quantum algorithms [6–8].

In this work, we take a machine-learning approach to evolving quantum algorithms, see Fig. 1. Our approach can be applied either to ideal hardware to derive fundamental algorithms or to a non-fully connected hardware with a non-ideal gate set to derive hardware-specific algorithms. We conceptually divide a quantum computation into the available resources, consisting of input qubits (data qubits and ancilla qubits) and output measurements, and the algorithm, consisting of a quantum gate sequence and classical post-processing of the measurement results (see Fig. 1). Fixing the resources as hyperparameters, we optimize the algorithm in a task-oriented manner, i.e., by minimizing a cost function that quantifies the discrepancy between the algorithm's output and the desired output, for a training data set that exemplifies the desired computation. We emphasize that our work goes beyond quantum compiling, which has received recent attention [9–13]. Although our method can be used to optimally compile known algorithms, our main goal is to discover novel algorithms via task-oriented programming.

We apply our approach to a ubiquitous task: computing the overlap between two quantum states. This computation yields $|\langle\psi|\phi\rangle|^2$ for two pure states $|\psi\rangle$ and $|\phi\rangle$, and more generally gives $\mathrm{Tr}(\rho\sigma)$ for two (possibly mixed) states ρ and σ. Furthermore, when specialized to the case $\rho = \sigma$, it computes the purity $\mathrm{Tr}(\rho^2)$ of a given state ρ.

There is a well-known algorithm for this task called the Swap Test [14, 15]. In quantum optics the Swap Test has a simple physical implementation [16–18]. However, for gate-based quantum computers (e.g., IBM's, Google's, and Rigetti's superconducting quantum computers and IonQ's trapped-ion quantum computer), the optimal implementation of the Swap Test is not obvious, and for single qubit states it can involve on the order of 15 to 30 gates, depending on the connectivity (See Fig. 2). For example, the Swap Test was experimentally implemented by IonQ [19] to quantify entanglement, with an algorithm employing 7 two-qubit gates and 11 one-qubit gates. Figure 2(B) and (C) respectively show decompositions of the Swap Test for IBM's and Rigetti's quantum computers, highlighting the non-trivial nature of implementing this algorithm.

Here, our machine-learning approach finds algorithms with a shorter depth than the Swap Test for computing the overlap. We first consider the same "quantum resources" as the Swap Test (access to a qubit ancilla and measurement on the ancilla), and our approach reduces the gate count to 4 controlled-NOTs (CNOTs) and 4 one-qubit gates. We call this our Ancilla-Based Algorithm (ABA). Then we allow for the additional resource of measuring all of the qubits, which gives an even shorter depth algorithm that essentially corresponds to a Bell-basis measurement with classical post-processing. We call this our Bell-Basis Algorithm (BBA). We also find short-depth algorithms for the specific hardware connectivity and gate alphabets used by IBM's and Rigetti's

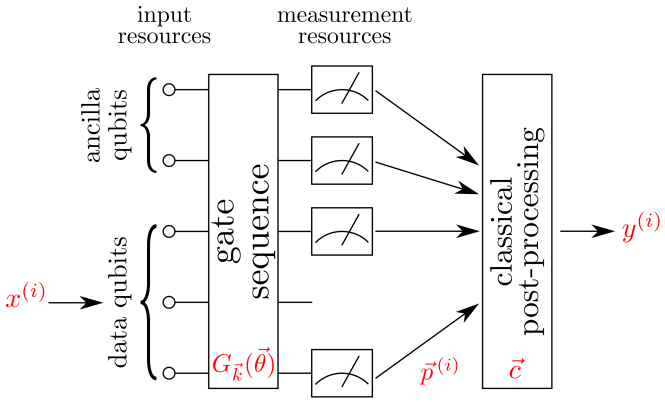

FIG. 1: Machine-learning approach to discovering and optimizing quantum algorithms. We optimize an algorithm for a given set of resources, which includes input resources (ancilla and data qubits) and measurement resources (i.e., which qubits can be measured). The algorithm is then determined by the quantum gate sequence and the classical post-processing of the measurement results. To find the algorithm that computes the function $x \to f(x)$, we minimize a cost function that quantifies the discrepancy between the desired output $f(x^{(i)})$ and the actual output $y^{(i)}$ for a set of training data inputs $\{x^{(i)}\}$. If the training data are sufficiently general, the algorithm that minimizes the cost should be a general algorithm that computes $f(x)$ for any input x.

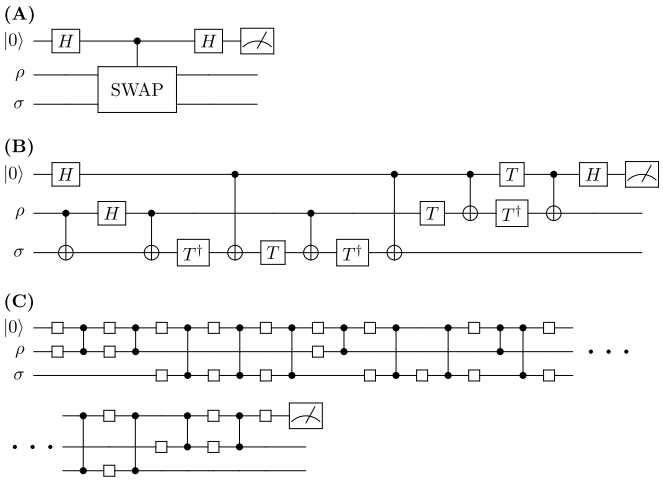

FIG. 2: Swap Test circuits. (A) The canonical Swap Test circuit. H indicates the Hadamard gate. (B) The Swap Test circuit adapted for IBM's 5-qubit quantum computer, constructed by decomposing controlled-swap into the Toffoli gate, via Refs. [25, 26], and then manually eliminating gates that had no effect on the output. T is the $\pi/8$ phase gate. (C) The structure of a Swap Test circuit, showing the locations of the one-qubit gates and controlled-Z gates, constructed automatically by Rigetti's compiler for their 19-qubit quantum computer.

quantum computers, which is crucial for reducing the computational error. Indeed, we found that our short-depth algorithms reduced the error (compared to the Swap Test) by 66% on IBM's 5-qubit computer and by 70% on Rigetti's 19-qubit computer.

Due to the fundamental nature of state overlap, the Swap Test appears in many applications. In *quantum supervised learning* [20, 21], which subsumes *quantum support vector machines* [22], the Swap Test is used to assign each data vector to a cluster. The Swap Test allows one to *quantify entanglement*, particularly the Renyi order-2 entanglement, for many-body quantum states [19, 23]. The Swap Test is useful for *benchmarking* on a quantum computer, since it can quantify the purity $\mathrm{Tr}(\rho^2)$ and hence the amount of decoherence that has occurred. It also has application to *quantum Merlin-Arthur games* [24]. For all of the above applications, our shorter-depth algorithm can be directly substituted in place of the Swap Test.

Note that if ρ and σ represent states on n qubits, the difficulty for computing $\mathrm{Tr}(\rho\sigma)$ scales exponentially with n for a classical computer. In contrast, the Swap Test has a circuit depth that grows linearly in n, giving an exponential speedup. Our ABA also has this property of scaling linearly with n, and it achieves a constant factor reduction in circuit depth (relative to Swap Test). On the other hand, our BBA has the nice feature that its circuit depth is constant, independent of n (although the complexity of its classical post-processing grows linearly in n). Due to its constant circuit depth, the BBA seems to be the best algorithm for quantifying state overlap on near-term quantum computers.

In what follows, we first present our machine-learning approach for discovering quantum algorithms. This approach can be applied to other applications besides overlap and hence should be of independent interest. Next, we present our main results: short-depth circuits for computing state overlap on idealized hardware. Then, we present hardware-specific algorithms for computing overlap. Finally we discuss our implementation of these algorithms on Rigetti's and IBM's quantum computers, leading to a reduction in the computational error relative to the Swap Test.

II. MACHINE-LEARNING APPROACH

Our machine-learning approach is summarized in Fig. 1. The general structure is that we divide the parameters up into the hyperparameters (i.e., the "resources") and the optimization parameters (i.e., the "algorithm").

A. Resources

The hyperparameters are the quantum resources at the input and output of the quantum circuit. At the input, the resources are the number of ancilla qubits (each initialized in the $|0\rangle$ state) and data qubits (qubits that

store the input data for the computation). At the output, the resources are the number and locations of the measurements on the output qubits (see Fig. 1). As an example, in the Swap Test for single-qubit states, one is allowed access to one ancilla qubit in the $|0\rangle$ state and two data qubits (a single copy of each state, ρ and σ) at the input, and one is allowed to measure only the ancilla qubit at the output.

The input data may be classical or quantum, depending on the computation of interest. In the case of state overlap - the computation considered in this paper - the input data are quantum states and hence no encoding is necessary. However, for completeness, we note that our approach also applies to classical inputs, in which case the encoding (i.e., storing the classical data in the quantum state of the data qubits) can be thought of as a hyperparameter that one fixes while optimizing the algorithm.

B. Algorithm

Our approach searches for an optimal "algorithm", where we consider the algorithm to be a quantum gate sequence with associated classical post-processing. We parameterize (and hence optimize over) both the gate sequence and the post-processing.

Let us first consider the gate sequence. We define a gate alphabet $\mathcal{A} = \{A_j(\theta)\}$. Here, each gate A_j is either a one-qubit or two-qubit gate and may also have an internal continuous parameter θ. Hence, $\mathcal{A}$ is a discrete set, but each element of $\mathcal{A}$ may have a continuous parameter associated with it. The precise choice of $\mathcal{A}$ depends on which hardware one is considering. For example, the connectivity differs between IBM and Rigetti hardware, and the former employs CNOT gates while the latter employs controlled-Z gates. For IBM's 5-qubit computer "ibmqx4" we can write out the gate alphabet as

$$\begin{aligned}\mathcal{A}_{\text{ibmqx4}} = \{&\text{CNOT}^{10}, \text{CNOT}^{20}, \text{CNOT}^{21}, \text{CNOT}^{32}, \\ &\text{CNOT}^{24}, \text{CNOT}^{34}, U^0(\theta), U^1(\theta), U^2(\theta), \\ &U^3(\theta), U^4(\theta)\}\,, \end{aligned} \tag{1}$$

where $U^j(\theta)$ is an arbitrary gate on qubit j and CNOT^{jk} is a CNOT from control qubit j to target qubit k. In this article, we treat all one-qubit gates equally in the sense that all one-qubit gates are equally complex to implement, although our approach could easily be generalized to account for different complexities for different one-qubit gates.

We consider a generic gate sequence with depth d,

$$G_{\vec{k}}(\vec{\theta}) = A_{k_d}(\theta_d) \cdots A_{k_2}(\theta_2) A_{k_1}(\theta_1)\,, \tag{2}$$

where $\vec{k} = (k_1, ..., k_d)$ is the vector of indices describing which gates are employed in the gate sequence and $\vec{\theta} = (\theta_1, ..., \theta_d)$ is the vector of continuous parameters associated with these gates.

The measurement results give rise to an outcome probability vector $\vec{p} = (p_1, ..., p_l, ...)$. The desired output might be one of these p_l probabilities, or it might be some simple function of these probabilities. Hence, we allow for some simple classical post-processing of $\vec{p}$ in order to reveal the desired output. While there is enormous freedom in applying a function to $\vec{p}$, we consider a simple linear combination of probabilities:

$$g(\vec{p}) = \vec{c} \cdot \vec{p} = \sum_l c_l p_l \tag{3}$$

where $\vec{c}$ is a vector of coefficients whose elements are chosen according to $c_l \in \{-1, 0, 1\}$. This post-processing is sufficient for the application in this paper (state overlap), although other applications may require a more general form of post-processing.

In summary, the free parameters that we optimize over (while fixing the hyperparameters) are the gate sequence vector $\vec{k}$, the continuous parameter vector $\vec{\theta}$, and the post-processing vector $\vec{c}$. For a given set of resources, these three vectors define the quantum algorithm, which we denote $Q_{\vec{m}}$, where $\vec{m} = \left(\vec{k}, \vec{\theta}, \vec{c}\right)$ is the concatenated vector.

C. Optimization

Optimizing these parameters involves defining and minimizing a cost function. The cost quantifies the discrepancy between the desired output and the actual output for a given training data set.

Suppose we want to find the algorithm that computes the function $x \to f(x)$. We generate data of the form

$$T = \{(x^{(i)}, f(x^{(i)}))\}_{i=1}^{2N}\,. \tag{4}$$

Half of this data is used for training the algorithm, i.e., optimizing the cost function. The other half is used for testing, i.e., evaluating the algorithm's performance. The training data must be sufficiently general to cover the space of possible inputs. An estimate of the amount of training data needed is $N \approx 2^{2n_D}$, where n_D is the number of data qubits. This can be seen by noting that our algorithm (which includes both the gate sequence and the post-processing) acts as a linear map from the data qubits' operator space, which has dimension 2^{2n_D}, to the output which is just a number and hence has dimension one. So our algorithm is basically a 1×2^{2n_D} matrix, and an estimate of the number of constraints (and hence the number of training data points) needed to fix the algorithm's parameters is 2^{2n_D}.

For example, when training the algorithm that computes overlap, $x^{(i)} = (\rho^{(i)}, \sigma^{(i)})$ consists of two quantum states $\rho^{(i)}$ and $\sigma^{(i)}$, and $f(x^{(i)}) = \text{Tr}(\rho^{(i)}\sigma^{(i)})$ quantifies their overlap. One can show that any algorithm that computes pure-state overlap also computes mixed-state

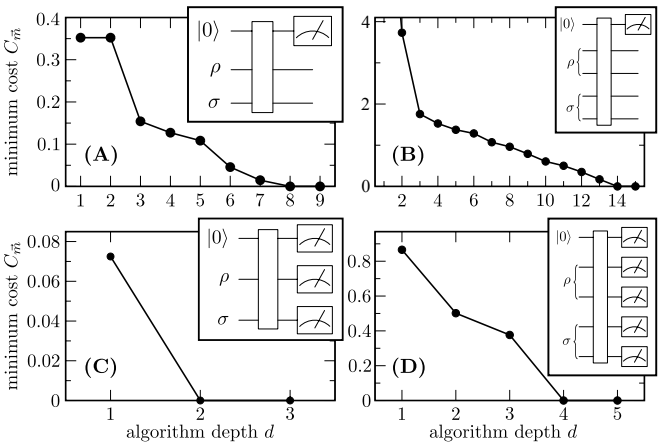

FIG. 3: Final cost that we obtained after minimizing our cost function versus the circuit depth d. (A) The resources allowed (shown in the inset) are the same as those allowed in the Swap Test, i.e., one ancilla qubit, two data qubits, and one measurement on the ancilla. This results in a minimum depth of $d_{\min} = 8$. (B) The scaling is obtained by increasing the number of qubits in ρ and σ, giving $d_{\min} = 14$ for $n = 2$ qubits. (C) Allowing for additional resources (shown in the inset) of measurements on all of the qubits results in a minimum depth of $d_{\min} = 2$. (D) Again we obtain the scaling by increasing the number of qubits in ρ and σ, giving $d_{\min} = 4$ for $n = 2$ qubits, when measurements on all qubits are allowed.

overlap. Hence, we generate our training data by randomly choosing pure states according to the Haar measure.

Next we define a cost function. For algorithm $Q_{\vec{m}}$, the cost is

$$C_{\vec{m}} = \sum_{i=1}^{N} (f(x^{(i)}) - y_{\vec{m}}^{(i)})^2 \,. \tag{5}$$

The cost quantifies the difference between the ideal output $f(x^{(i)})$ and the actual output $y_{\vec{m}}^{(i)}$ for each training data point. The actual output can be written as

$$y_{\vec{m}}^{(i)} = y_{(\vec{k},\vec{\theta},\vec{c})}^{(i)} = \vec{c} \cdot \vec{p}_{(\vec{k},\vec{\theta})}^{(i)} \tag{6}$$

where $\vec{c}$ is the post-processing vector and $\vec{p}_{(\vec{k},\vec{\theta})}^{(i)}$ is the outcome probability vector for input $x^{(i)}$. For example, in the Swap Test, the outcome probability vector corresponds to the ancilla qubit's measurement, and $\vec{c} = (1, -1)$ ensures that $y_{\vec{m}}^{(i)}$ is the expectation value of the Pauli Z operator.

For a fixed circuit depth d, we search over the algorithm space to minimize the cost. We consider various d, incrementing from small to large values. When an exact algorithm exists, we typically are able to minimize the cost. That is, we can find a $Q_{\vec{m}}$ with $C_{\vec{m}} \approx 0$, for $d \geqslant d_{\min}$, where $d_{\min}$ is the minimum depth needed to minimize the cost (see Fig. 3 for example plots of final cost versus d). Due to gauge freedom, there are typically many $Q_{\vec{m}}$ that give zero cost for $d \geqslant d_{\min}$. So, in the Main Results section, we present our simplest formulations of such algorithms.

D. Scaling

For a fixed problem size, we minimize the cost. If the cost goes to zero (which we define as a cost less than 10^{-6}), we say we have an *algorithm instance*. In particular, this corresponds to fixing the size of the data and hence fixing n_D, the number of data qubits. To study the scaling of the algorithm, we grow the size of the problem by increasing n_D. In some cases, one may also need to increase the number of ancilla qubits, n_A, and/or the number of measurements in order to minimize the cost.

This gives us a set of algorithm instances for various problem sizes. An important challenge is to abstract a general algorithm from these instances. This challenge is particularly difficult because one can typically only find algorithm instances for small problem sizes. This is due to the fact that the search space for vectors $\vec{k}$ grows rapidly with problem size, namely as n_T^{2d}, where $n_T = n_D + n_A$ is the total number of qubits and d is the circuit depth.

In this work, we were able to manually recognize the pattern by which the algorithm scales by inspecting the various algorithm instances. In future work, we will explore automated methods for recognizing algorithm scaling.

III. MAIN RESULTS

A. Overview

Our main results are short-depth algorithms for quantifying overlap on idealized quantum computing hardware. For the latter, we consider full connectivity, and we allow for arbitrary one-qubit gates as well as CNOT gates between all of the qubits.

We consider two sets of resources. The first set of resources are identical to those allowed for the Swap Test, i.e., access to one ancilla qubit and two data qubits, as well as one measurement on the ancilla qubit. The cost versus depth for these resources is shown in Fig. 3(A), and we obtained essentially zero cost for $d = 8$. To see the scaling, we increase the number of qubits in ρ and σ to $n = 2$, giving a minimum depth of $d = 14$, as shown in Fig. 3(B). As discussed below this leads to an algorithm (shown in Fig. 4) that we refer to as our ancilla-based algorithm (ABA).

The second set of resources we consider allows for measurements on all of the qubits. For these additional resources, Figure 3(C) shows that zero cost is obtained for $d = 2$. To see the scaling, we increase the number of qubits to $n = 2$, giving a minimum depth of $d = 4$,

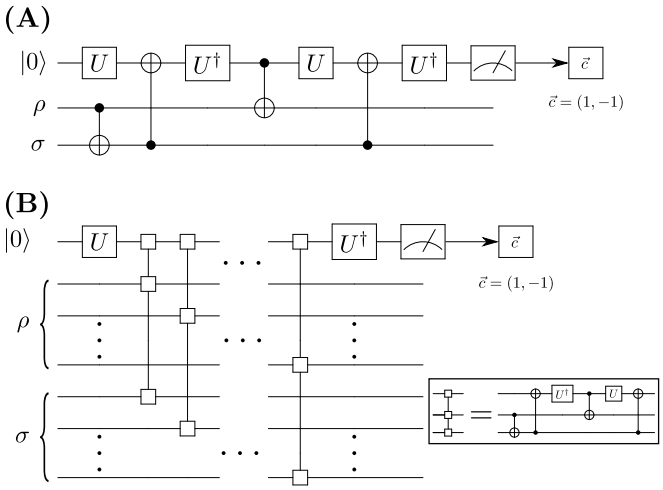

FIG. 4: Our ancilla-based algorithm, obtained by minimizing the cost for the resources shown in Fig. 3(A) and (B). (A) When ρ and σ are one-qubit states, we obtain a circuit with 4 CNOT gates and 4 one-qubit gates for a total of 8 gates. Here, $U = T^\dagger H$. (B) Six of these gates are combined to create a "building block" (see inset) that is used to scale the algorithm for input states ρ and σ of arbitrary size. The post-processing vector is $\vec{c} = (1, -1)$, independent of problem size.

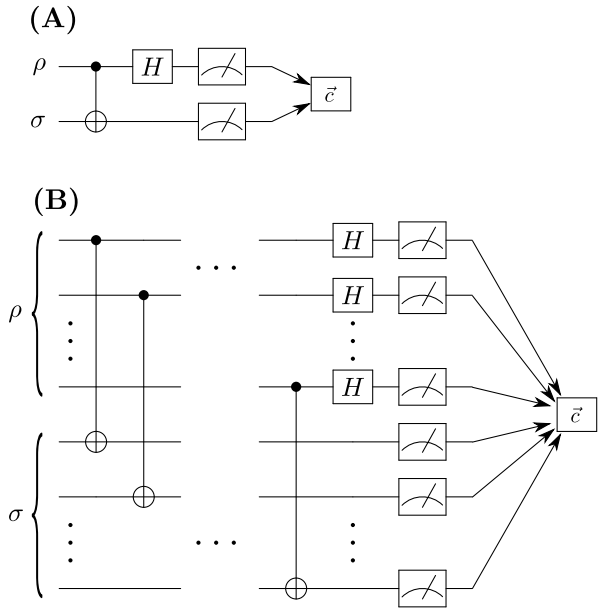

FIG. 5: Our bell-basis algorithm, obtained by minimizing the cost for the resources shown in Fig. 3(C) and (D). (A) When ρ and σ are one-qubit states, we obtain a circuit with one CNOT followed by a Hadamard and measurements on both qubits with a post-processing vector $\vec{c} = (1, 1, 1, -1)$. (B) The CNOT and Hadamard gates form a "building block" that is used to scale the algorithm for input states ρ and σ of arbitrary size. Since these gates can be parallelized, the quantum circuit depth is independent of problem size. On the other hand, the complexity of classical post-processing grows linearly with n, and the post-processing vector can be written as $\vec{c} = (1, 1, 1, -1)^{\otimes n}$ if one orders the qubits into pairs from ρ and σ.

as shown in Fig. 3(D). The surprising result is that the ancilla qubit is not used at all in this algorithm, even though we train the algorithm in the presence of an ancilla. This allows us to display the resulting algorithm, our Bell-basis algorithm (BBA), in Fig. 5 without the ancilla qubit.

B. Ancilla-based algorithm

Figure 4(A) shows the ABA for one-qubit states ρ and σ. The unitary U in this circuit is

$$U = T^\dagger H \,. \tag{7}$$

This circuit employs 4 CNOT gates and 4 one-qubit gates for a total of 8 gates. It uses a simple post-processing vector $\vec{c} = (1, -1)$ that amounts to measuring the Pauli Z operator on the ancilla qubit, which is the same observable measured in the Swap Test. Note that this circuit has shorter depth than typical implementations of the Swap Test, e.g., the circuit in Fig. 1(B).

The scaling of this algorithm is given in Fig. 4(B). There is a repeating unit, shown in the inset of the figure, that is applied on each pair of qubits composing ρ and σ. This unit has 4 CNOT gates, so the overall algorithm employs $4n$ CNOT gates and $6n + 2$ total gates. Hence, the circuit depth grows linearly with the number of data qubits.

C. Bell basis algorithm

Figure 5(A) shows the BBA for one-qubit states ρ and σ. This circuit employs one CNOT gate followed by one Hadamard gate, with both qubits being measured. It is straightforward to show that this corresponds to a Bell basis measurement. The post-processing is a bit more complicated, with $\vec{c} = (1, 1, 1, -1)$, which corresponds to summing the probabilities for the 00, 01, and 10 outcomes and subtracting probability of the 11 outcome.

The scaling of this algorithm is given in Fig. 5(B). The repeating unit is simply a CNOT and Hadamard, applied on each pair of qubits composing ρ and σ. Furthermore, every qubit is measured at the output. The total number of gates is simply $2n$, and hence grows linearly with the number of qubits. However, more importantly, the CNOT and Hadamard on each qubit pair can be performed in parallel. This crucial fact means that this algorithm has a *constant depth, independent of problem size*. Namely, the depth is two quantum gates.

On the other hand, the classical post-processing is somewhat complicated, and its complexity scales linearly

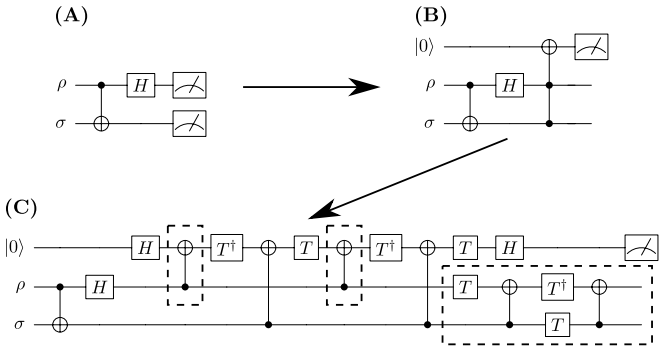

FIG. 6: Equivalence between our ABA and BBA. The two-qubit measurement and classical post-processing in the BBA can be converted to a Toffoli gate with an ancilla as the target followed by a measurement on the ancilla. This takes us from circuit (A) to circuit (B). Inserting into circuit (B) the optimal decomposition of the Toffoli gate from Ref. [26] gives circuit (C). Finally one does three simplifications of this circuit to obtain the ABA, indicated by the dashed boxes in (C). Namely, the first boxed CNOT in (C) has trivial action and hence can be removed. The second boxed CNOT in (C) can be flipped such that the ancilla is the control qubit, which introduces some Hadamards. One of these Hadamards cancels with the first Hadamard in (C), and two others combine with T and $T^\dagger$ to make the $U^\dagger$ and U shown in Fig. 4(A). Finally the five gates enclosed in the last dashed box in (C) have no effect on the measurement and hence can be removed.

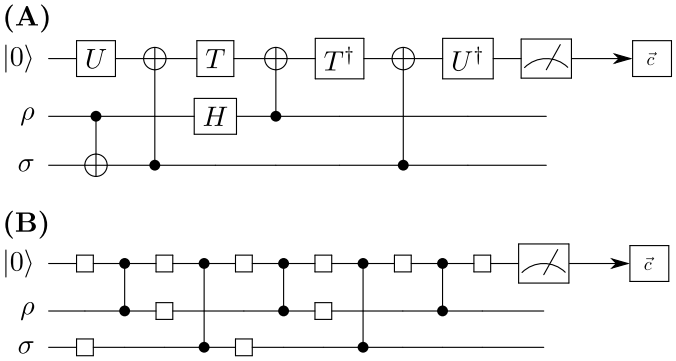

FIG. 7: Ancilla-based algorithm adapted (via our machine-learning approach) to commercial hardware. (A) ABA adapted to IBM's 5-qubit computer. (B) ABA adapted to Rigetti's 19-qubit computer. For simplicity, the locations (but not the precise unitaries) of the one-qubit gates are shown.

with the problem size. Namely, the post-processing vector can be written as $\vec{c} = (1, 1, 1, -1)^{\otimes n}$, provided that one arranges the qubits in the order $A_1B_1A_2B_2....A_nB_n$, where $A_1A_2...A_n$ and $B_1B_2...B_n$ are the subsystems composing ρ and σ respectively. Nevertheless, due to decoherence and gate infidelity, it is better for the classical post-processing to grow linearly in n than for the quantum circuit depth to grow linearly in n. Hence, the BBA seems to be the superior algorithm.

D. Discussion

In 2013, Garcia-Escartin and Chamorro-Posada discovered the Bell-basis algorithm for computing state overlap [16]. We were unaware of this important article until after our machine-learning approach found our BBA. More generally, it appears that the quantum computing community seems to be unaware of this article, perhaps because the article was presented in the language of quantum optics rather than that of quantum computing. Indeed, the ancilla-based version of the Swap Test, shown in Fig. 1, continues to be the algorithm employed in the quantum computing literature (e.g., see Refs. [19, 23]).

Although our two algorithms look very different, one can actually show a simple equivalence between our ABA and our BBA. One can see this by converting the classical post-processing in the BBA into a quantum gate. In particular, this gate would be a Toffoli gate, controlled by the two data qubits with the target being an ancilla qubit prepared in the $|0\rangle$ state. One would then do a measurement of the Pauli Z observable on the ancilla to decode the state overlap. By inserting the decomposition of the Toffoli gate from Ref. [26] and simplifying the resulting circuit, one can then obtain our ABA. In this sense, our ABA is essentially our BBA but with the classical post-processing transformed into a Toffoli gate. This equivalence is shown in Fig. 6.

We argue that our ABA is inequivalent to the Swap Test as follows. Let $S_{\rm ABA}$ denote the Schmidt rank (across the cut between ancilla and the data qubits) of the unitary $G_{\rm ABA}$ associated with gate sequence in the ABA. It can be verified that $S_{\rm ABA} = 3$. This means that $G_{\rm ABA}$ is not locally equivalent to controlled-SWAP, whose analogously defined Schmidt rank is 2. We have therefore shown that ABA is fundamentally different from the Swap Test: it cannot be obtained from the Swap Test by local operations.

IV. HARDWARE-SPECIFIC ALGORITHMS

Our BBA can be directly implemented on IBM's and Rigetti's quantum computers without any concern about connectivity issues (except for the minor issue that Rigetti uses controlled-Z instead of CNOT - their compiler easily makes the translation).

However, our ABA needs to be modified to account for IBM's and Rigetti's connectivity. While it is possible to manually modify the ABA to fit the connectivity, to illustrate our machine-learning approach, we numerically optimized the algorithm with the same resources as that shown in Fig. 3(A). The only difference is that we specified the gate set to match the gate set (and hence the connectivity) of IBM's and Rigetti's computers.

The resulting algorithms that we obtained with our

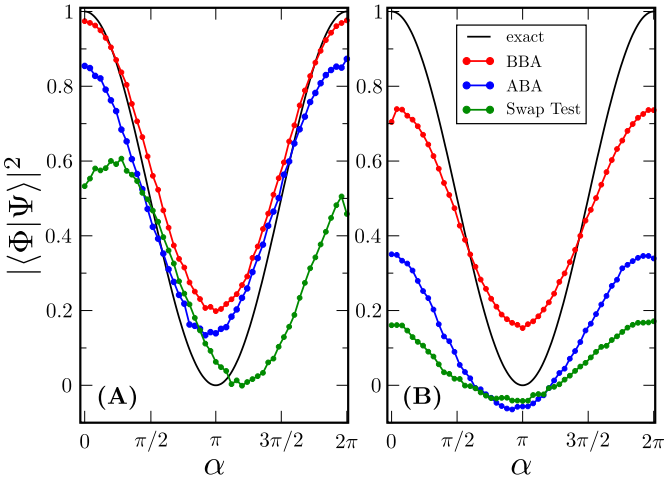

FIG. 8: Experimentally observed overlaps on commercial hardware for the states $|\Psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|\Phi\rangle = (|0\rangle + e^{i\alpha}|1\rangle)/\sqrt{2}$. (A) Results from IBM's 5-qubit computer called "ibmqx4", with 49,152 quantum computer runs per data point. The black curve is the analytical overlap $|\langle\Phi|\Psi\rangle|^2$. The red, blue, and green curves are respectively the results for the BBA from Fig. 5(A), the ABA from Fig. 7(A), and the Swap Test from Fig. 2(B). (B) Results from Rigetti's 19-qubit computer, with 200,000 quantum computer runs per data point. The curves are analogous to those from panel (A). Namely, the red, blue, and green curves are respectively for the BBA from Fig. 5(A), the ABA from Fig. 7(B), and the Swap Test from Fig. 2(A) which Rigetti compiled to Fig. 2(C).

machine-learning approach are shown in Fig. 7. The ABA adapted to IBM's 5-qubit computer only requires one additional gate, a Hadamard gate. The ABA adapted to Rigetti's 19-qubit computer requires an additional two-qubit gate and several additional one-qubit gates.

V. TESTING OUR ALGORITHMS

We implemented our algorithms on IBM's 5-qubit and Rigetti's 19-qubit computers. The resulting data are shown in Fig. 8. A caveat is that the different qubit counts for the two hardwares make it difficult to directly compare the results between these hardwares.

We considered two pure states of the form

$$|\Psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2} \tag{8}$$

$$|\Phi\rangle = (|0\rangle + e^{i\alpha}|1\rangle)/\sqrt{2}\,, \tag{9}$$

and we compared our results to the exact overlap $|\langle\Phi|\Psi\rangle|^2$ (black curve in Fig. 8). The root-mean-square (RMS) errors are shown in Table I.

On both computers, the Swap Test (green curve in Fig. 8) performed poorly. It is noteworthy that these are only single-qubit states, and hence the results are expected to be even worse as one grows the size of these states.

	IBM (5 qubits)	Rigetti (19 qubits)
Swap Test	0.311	0.537
ABA	0.106	0.432
BBA	0.116	0.160

TABLE I: RMS errors for the data shown in Fig. 8.

Our ABA performed significantly better than Swap Test, while using the same resources. The BBA, which allows for measurements on all qubits, dramatically outperformed the other algorithms on Rigetti's computer and performed roughly the same as ABA on IBM's computer. The relatively high accuracy of BBA is naturally expected due to its short depth, which mitigates the effects of decoherence and gate infidelity.

VI. CONCLUSIONS

This work shows that even well-known algorithms can be improved upon using an automated approach. As noted in the Introduction, there are many applications that require state overlap computation, including the emerging new field of quantum machine learning. While the Swap Test appears as a subroutine in many of these applications, we show that there are more efficient circuits to perform this subroutine.

We have found a constant depth algorithm (denoted BBA above) for computing state overlap, which is better than the linear scaling of Swap Test. Furthermore, this algorithm performs better - with significantly lower error - even in the single-qubit case. We recommend that researchers use this algorithm henceforth for computing state overlap on near-term quantum computers. This algorithm essentially corresponds to a measurement in the Bell basis for corresponding pairs of qubits. A key aspect of our approach that aided this algorithm's discovery was to allow for non-trivial classical post-processing, a strategy that has been used previously to shrink the depth of quantum algorithms [27]. The complexity of the post-processing for the BBA scales only linearly in the problem size (i.e., the number of qubits), ensuring that the quantum speedup that this algorithm provides is not due to the transfer of exponential complexity to the classical post-processing.

Our main technical tool was a machine-learning method that allowed for task-oriented discovery of quantum algorithms. By task-oriented, we mean that this method defines a cost function based upon training data that are representative of the desired computation, i.e., the training data define the task. Minimizing the cost function results in a general algorithm for this computation. We emphasize that this goes far beyond quantum compiling since it allows for algorithm discovery when no algorithm is known.

Conceptually, our method separates quantum resources (ancillas, data qubits, and measurements) from algorithm parameters (gate sequence and classical post-processing). The former are fixed as hyperparameters while we optimize the latter. The algorithm's scaling is obtained by training for various problem sizes and recognizing the pattern. In future work, we plan to automate the process of pattern recognition for algorithm scaling.

Other automated algorithm-discovery approaches have been employed in the literature. Gepp and Stocks review much of the early work to evolve quantum algorithms using genetic programming [8]. They remark that this field will be even more promising when quantum computers become available. This is due to the exponential speedup they provide in evaluating algorithm cost, i.e., by avoiding the exponential overhead of quantum simulation on classical computers. Indeed, some recent works propose to use quantum computers in automated algorithm learning [6, 28]. Likewise our method can be extended to learning on a quantum computer by outsourcing cost evaluation to the quantum computer. This will be a topic of our future work.

VII. ACKNOWLEDGEMENTS

We thank Rigetti and IBM for providing access to their quantum computers. LC was supported by the U.S. Department of Energy through the J. Robert Oppenheimer fellowship. YS acknowledges support of the LDRD program at Los Alamos National Laboratory (LANL). ATS and PJC were supported by the LANL ASC Beyond Moore's Law project.

[1] Preskill, J. Quantum computing and the entanglement frontier. *arXiv:1203.5813* (2012).

[2] Neill, C. *et al.* A blueprint for demonstrating quantum supremacy with superconducting qubits. *arXiv:1709.06678* (2017).

[3] Ball, P. The era of quantum computing is here. Outlook: cloudy. *Quanta Magazine* (2018).

[4] Fowler, A. G., Mariantoni, M., Martinis, J. M. & Cleland, A. N. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* **86**, 032324 (2012).

[5] You, H., Geller, M. R., Stancil, P. *et al.* Simulating the transverse Ising model on a quantum computer: Error correction with the surface code. *Physical Review A* **87**, 032341 (2013).

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

[7] Khaneja, N., Reiss, T., Kehlet, C., Schulte-Herbrüggen, T. & Glaser, S. J. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of magnetic resonance* **172**, 296–305 (2005).

[8] Gepp, A. & Stocks, P. A review of procedures to evolve quantum algorithms. *Genetic programming and evolvable machines* **10**, 181–228 (2009).

[9] Venturelli, D., Do, M., Rieffel, E. & Frank, J. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology* **3**, 025004 (2018).

[10] Häner, T., Steiger, D. S., Svore, K. & Troyer, M. A software methodology for compiling quantum programs. *Quantum Science and Technology* (2018).

[11] Maslov, D. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics* **19**, 023035 (2017).

[12] Martinez, E. A., Monz, T., Nigg, D., Schindler, P. & Blatt, R. Compiling quantum algorithms for architectures with multi-qubit gates. *New Journal of Physics* **18**, 063029 (2016).

[13] Chong, F. T., Franklin, D. & Martonosi, M. Programming languages and compiler design for realistic quantum hardware. *Nature* **549**, 180 (2017).

[14] Buhrman, H., Cleve, R., Watrous, J. & De Wolf, R. Quantum fingerprinting. *Physical Review Letters* **87**, 167902 (2001).

[15] Gottesman, D. & Chuang, I. Quantum digital signatures. *quant-ph/0105032* (2001).

[16] Garcia-Escartin, J. C. & Chamorro-Posada, P. Swap test and Hong-Ou-Mandel effect are equivalent. *Physical Review A* **87**, 052330 (2013).

[17] Patel, R. B., Ho, J., Ferreyrol, F., Ralph, T. C. & Pryde, G. J. A quantum Fredkin gate. *Science advances* **2**, e1501531 (2016).

[18] Ferreyrol, F., Ralph, T. C. & Pryde, G. J. Implementation of a quantum Fredkin gate using an entanglement resource. In *Lasers and Electro-Optics Europe (CLEO EUROPE/IQEC), 2013 Conference on and International Quantum Electronics Conference*, 1–1 (2013).

[19] Linke, N. M. *et al.* Measuring the Renyi entropy of a two-site Fermi-Hubbard model on a trapped ion quantum computer. *arXiv:1712.08581* (2017).

[20] Lloyd, S., Mohseni, M. & Rebentrost, P. Quantum algorithms for supervised and unsupervised machine learning. *arXiv:1307.0411* (2013).

[21] Wiebe, N., Kapoor, A. & Svore, K. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv:1401.2142* (2014).

[22] Rebentrost, P., Mohseni, M. & Lloyd, S. Quantum support vector machine for big data classification. *Physical Review Letters* **113**, 130503 (2014).

[23] Johri, S., Steiger, D. S. & Troyer, M. Entanglement spectroscopy on a quantum computer. *Physical Review B* **96**, 195136 (2017).

[24] Kobayashi, H., Matsumoto, K. & Yamakami, T. Quantum Merlin-Arthur proof systems: Are multiple Merlins more helpful to Arthur? In *International Symposium on Algorithms and Computation*, 189–198 (Springer, 2003).

[25] Smolin, J. A. & DiVincenzo, D. P. Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate. *Physical Review A* **53**, 2855 (1996).

[26] Shende, V. V. & Markov, I. L. On the CNOT-cost of Toffoli gates. *Quantum Information and Computation* **9**, 0461–0486 (2009).

[27] Svore, K. M., Hastings, M. B. & Freedman, M. Faster phase estimation. *Quantum Information & Computation* **14**, 306–328 (2014).

[28] Mitarai, K., Negoro, M. & Kitagawa, K., Masahiro nd Fujii. Quantum circuit learning. *arXiv:1803.00745* (2018).