Efficient verification of continuously-parameterized quantum gates

Ryan Shaffer,

Hang Ren,

Emilia Dyrenkova,

Christopher G. Yale,

Daniel S. Lobser,

Ashlyn D. Burch,

Matthew N. H. Chow,

Melissa C. Revelle,

Susan M. Clark,

Hartmut Häffner

1Department of Physics, University of California, Berkeley, CA, USA
2Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA
3Challenge Institute for Quantum Computation, University of California, Berkeley, CA, USA
4Sandia National Laboratories, Albuquerque, NM 87123, USA
5Department of Physics and Astronomy, University of New Mexico, Albuquerque, NM 87131, USA
6Center for Quantum Information and Control, University of New Mexico, Albuquerque, NM 87131, USA

(Dated: May 27, 2022)

Most near-term quantum information processing devices will not be capable of implementing quantum error correction and the associated logical quantum gate set. Instead, quantum circuits will be implemented directly using the physical native gate set of the device. These native gates often have a parameterization (e.g., rotation angles) which provide the ability to perform a continuous range of operations. Verification of the correct operation of these gates across the allowable range of parameters is important for gaining confidence in the reliability of these devices. In this work, we demonstrate a procedure for efficient verification of continuously-parameterized quantum gates. This procedure involves generating random sequences of randomly-parameterized layers of gates chosen from the native gate set of the device, and then stochastically compiling an approximate inverse to this sequence such that executing the full sequence on the device should leave the system near its initial state. We show that fidelity estimates made via this technique have a lower variance than fidelity estimates made via cross-entropy benchmarking, which thus provides an efficiency advantage when estimating the error rate to some desired precision. We describe the experimental realization of this technique using a continuously-parameterized quantum gate set on the Sandia QSCOUT trapped-ion processor, and we demonstrate the efficiency advantage of this technique both numerically and experimentally.

I. INTRODUCTION

Verifying the correct operation of quantum computations is an essential step toward building a reliable and scalable quantum information processing device [1]. Most commonly, quantum computations are broken down into fundamental building blocks known as quantum gates, which may include gates such as the well-known Hadamard, Pauli, and CNOT operations. Verifying the behavior of a device’s physically-realizable gates, known as a native gate set, is a primary area of research in this field. The most complete techniques for gate verification belong to a family of techniques known as tomography. Such techniques include quantum state tomography [2, 3], quantum process tomography [4], and gate set tomography [5, 6]. Tomographic techniques produce a complete characterization of a quantum operation, which provides a detailed mathematical description of the errors present in the system. However, tomography is extremely resource-intensive, and although techniques exist to improve its scalability somewhat [7, 8], its cost still typically scales exponentially with qubit count. In contrast, benchmarking techniques for verifying quantum gates are typically resource-efficient and in principle can be scaled to much larger qubit counts than tomographic techniques. These techniques notably include randomized benchmarking (RB) [9, 10] and more scalable variants such as cycle benchmarking [11] and direct RB [12], which involve executing randomized circuits which are equivalent to the identity, as well as techniques such as cross-entropy benchmarking (xeb) [13] and match-gate benchmarking [14], which compare the ideal and experimental output probabilities of random quantum circuits. Benchmarking provides an incomplete characterization of a quantum system, typically producing a small number of values which attempt to characterize the average error rate of particular operations performed by the system. But as the name implies, such techniques are particularly useful when attempting to compare the performance of distinct devices, since they provide metrics which are ostensibly hardware-agnostic. For example, benchmarking techniques may provide an estimate of the average error rate of executing a CNOT gate or of a device’s average state preparation and measurement (SPAM) error.

Because many quantum algorithms and especially quantum error correction schemes are expressed in terms of particular fixed sets of gates – most commonly, the Cliff+T family, which is universal for quantum computation – much of the benchmarking literature is focused on verifying the operation of these fixed one-qubit and two-qubit gates, or their device-native equivalents. However, near-term quantum devices are unlikely to implement large-scale quantum error correction [15], and instead will implement circuits directly using the physical native gate set of the device. These native gates are often not limited to the fixed gate set used by er-
ror correction schemes, but rather have a parameterization which provides the ability to perform a continuous range of operations. For example, a single-qubit operation may frequently be parameterized as $R(\theta, \varphi)$, where $\theta$ is the rotation angle and $\varphi$ is the axis of rotation. Multi-qubit operations may also be parameterized. The typical multi-qubit gate for trapped-ion devices, based on the Mølmer-Sørensen interaction \cite{16,17}, can be parameterized as $MS(\theta, \varphi)$, where $\theta$ can be interpreted as the effective rotation angle in the multi-qubit space, and $\varphi$ is the effective multi-qubit axis of rotation \cite{18}. In many instances, compiling quantum circuits using continuously-parameterized native gate sets can produce more efficient compilations on physical devices than when limited to fixed gate sets \cite{19}.

Systematic and efficient verification techniques for continuously-parameterized quantum gates, therefore, are a key ingredient for near-term quantum computers to reliably take advantage of the full scope of physically-realizable operations. In this work, we discuss the application of the randomized analog verification (RAV) technique \cite{20} to the task of verifying continuously-parameterized quantum gates. In Section II, we provide an overview of the RAV technique and compare it to cross-entropy benchmarking, one of the primary existing techniques for this task; we describe the stochastic compilation scheme used in constructing the RAV sequences; and we provide details on the experimental setup of the trapped-ion quantum processor, the Quantum Scientific Computing Open User Testbed (QSCOUT) operated by Sandia National Laboratories \cite{21}, including the technical details of the functionality required to support RAV sequences. In Section III, we provide numerical simulations demonstrating the conditions under which we expect RAV to provide an efficiency advantage over existing techniques, and we report experimental demonstrations of this efficiency advantage on the QSCOUT trapped-ion device and on a superconducting quantum processor from the publicly-available IBM Q service \cite{22}. We conclude with additional discussion of these results in Section IV.

\section*{II. METHODS}

\subsection*{A. Randomized analog verification for continuously-parameterized quantum gates}

The verification technique introduced in this work is a gate-based adaptation of the randomized analog verification (RAV) protocol for analog quantum simulators \cite{20}. When applied to analog quantum simulations, the RAV protocol consists of running randomized analog sequences of subsets of terms of a target Hamiltonian. In particular, a set of unitary operators is chosen consisting of short, discrete time steps of each of the terms of the target Hamiltonian. A randomly-generated sequence of these operations is then applied, which evolves the system to some arbitrary state. Next, an approximate inverse of this sequence, generated using the same set of unitary operators by using a stochastic compilation protocol (see Section II.B), is applied to the system, which returns it to the initial state with high probability.

Because current gate-based, non-error-corrected quantum computers are realized by carefully tuning the underlying analog interactions to implement quantum gates with the highest fidelity possible, it is natural to adapt the RAV protocol for use in verifying the behavior of gate-based devices with continuously-parameterized native gates. To accomplish this, we use an approach similar to that used in cross-entropy benchmarking (XEB) by constructing random sequences of layers, each of which consists of some fixed number of each of the device’s native gates in some randomly-chosen order. Random parameter values are then provided to each of these gates, which allows the protocol to verify the behavior of the device across the range of allowable parameter values for each gate. But unlike XEB, which proceeds by sampling directly from the output of these random sequences, RAV appends a compiled sequence of layers which approximately inverts the initial sequence. A schematic of the RAV protocol is displayed in Figure 1 which illustrates the fact that the primary difference from XEB is this inversion sequence which returns the system nearly to the initial state.

Given a single XEB sequence on an $n$-qubit system, we can approximate the resulting fidelity as

$$\hat{F}_{\text{XEB}} = \sum_x P(x)Q(x) - \frac{1}{N}$$

where we have simplified the linear cross-entropy fidelity formula \cite{13} for the case of a single circuit. Here, $P(x)$ represents the classically-computed ideal output probability distribution for the sequence, $Q(x)$ is the observed fidelity distribution for the sequence, and $\sum_x P(x)^2 = 1/N$.
sample probability of obtaining measurement result \( x \), and \( N = 2^n \) is the dimension of the system. \( \hat{F}_{\text{XEB}} \) is constructed such that its observed value for a single circuit might not fall within the range \([0,1]\). But in general, the expected fidelity of the ideal output state (i.e., if \( P(x) = Q(x) \) \( \forall x \)) is 1, and the expected fidelity of a maximally-mixed output state is 0.

To derive a formula for the fidelity of a single RAV sequence on an \( n \)-qubit system, we start with the XEB formula in Equation \( \[1\] \). We then note that, by construction of the RAV sequence, \( P(x_0) \approx 1 \), where \( x_0 \) is the initial state (and expected final state) of the RAV sequence. After some simplification (see Appendix A), we arrive at the following expression for the approximate RAV sequence fidelity:

\[
\hat{F}_{\text{RAV}} = \frac{Q(x_0) - \frac{1}{N}}{P(x_0) - \frac{1}{N}}. \tag{2}
\]

We use the hat on the symbols \( \hat{F}_{\text{XEB}} \) and \( \hat{F}_{\text{RAV}} \) to emphasize that they are only estimates of fidelity based on a single circuit instance. To obtain reliable information about the fidelity, these results must be aggregated over many circuit instances. This is especially true in the case of smaller qubit count, where the variance across different random circuit instances is most significant [23].

It is also worth clarifying that the fidelity estimated by \( \hat{F}_{\text{XEB}} \) and \( \hat{F}_{\text{RAV}} \) is the depolarization fidelity [24], which is not equivalent to the typical state fidelity that is used in many contexts when discussing fidelity. In particular, if we assume that a circuit’s ideal output state is \( |\psi\rangle \langle \psi| \) and its execution is subject to purely depolarizing errors, then we can represent the true output state as

\[
\rho_\lambda = (1 - \lambda) |\psi\rangle \langle \psi| + \frac{\lambda}{N} I \tag{3}
\]

where \( \lambda \in [0,1] \) is the fraction to which the output state is depolarized. The depolarization fidelity of this state is then defined as \( F = 1 - \lambda \). The average depolarization fidelity \( \overline{F} \) is related to the average state fidelity \( \overline{F} \) as

\[
\overline{F}^2 = F + \frac{1 - F}{N}. \tag{4}
\]

Intuitively, we expect that measuring the success of RAV sequences should be more efficient than measuring the success of XEB sequences. This is because RAV requires estimating the output probability \( Q(x_0) \) of only the initial state (under the assumptions used to derive Equation \( \[2\] \)), whereas XEB requires estimating the full output probability distribution \( Q(x) \).

More concretely, we can demonstrate this efficiency advantage by calculating the statistical variance associated with measurement of \( \hat{F}_{\text{RAV}} \) and \( \hat{F}_{\text{XEB}} \) for a single sequence. By making several simplifying assumptions (see Appendix B for full details), we can approximate the variance of these quantities as

\[
\text{Var}[\hat{F}_{\text{RAV}}] \approx \frac{1}{K} \left( \frac{1}{(1 - \epsilon)^{N/2}} \right)^2 \left[ (1 - \lambda)(1 - \epsilon) + \frac{\lambda}{N} \right] \times \left[ 1 - (1 - \lambda)(1 - \epsilon) - \frac{\lambda}{N} \right] \tag{5}
\]

\[
\text{Var}[\hat{F}_{\text{XEB}}] \approx \frac{1}{K} \left( \frac{N}{2N-1} \right)^2 \left[ \frac{1}{Q} (1 - \epsilon) \right] + \frac{1}{Q}(1 - \lambda) \left( 1 - \frac{\lambda}{N} \right) - \frac{1}{Q}(1 - \lambda)^2 \tag{6}
\]

where \( K \) is the number of independent experimental shots taken for the given sequence and \( \epsilon \ll 1 \) is the error in the compiled inversion of the RAV sequence.

We plot the standard deviation of these fidelity estimates (i.e., \( \sqrt{\text{Var}[\hat{F}_{\text{RAV}}]} \) and \( \sqrt{\text{Var}[\hat{F}_{\text{XEB}}]} \)) in Figure 2 for \( 2 \leq n \leq 16 \) qubits and \( 0 \leq \lambda \leq 1 \), assuming \( K = 100 \) and \( \epsilon = 0.04 \). The mean of the RAV and XEB fidelity estimates are in agreement in all cases, but we observe from these plots that RAV fidelity estimates have a lower standard deviation than XEB in all cases, with the advantage tending to shrink as \( \lambda \) and \( n \) increase. This implies that fewer RAV repetitions are necessary in order to get an equivalently-precise estimate of the fidelity of a given sequence. We note that this advantage is largest in the regime where \( \lambda \) is small, since this is exactly the regime where the final state of a RAV sequence is near a basis state, which minimizes the quantum projection noise associated with the computational basis measurement.

As a further illustration, we perform a simulation of RAV and XEB fidelity measurements across various regimes of fidelity decay assuming a purely depolarizing channel. We generate representative RAV and XEB sequences for systems with \( 2 \leq n \leq 8 \) qubits, and we calculate the standard deviation of fidelity measurements on these sequences for \( 0 \leq \lambda \leq 1 \). These plots, shown in Figure 3, show good agreement between the calculated standard deviation for the RAV circuits and the standard deviation extracted from simulations. We observe some quantitative disagreement for the XEB circuits, which is likely because the assumptions behind Equation \( \[5\] \) are not necessarily valid in our \( n \leq 8 \) regime (see Appendix B). Nonetheless, we observe qualitative agreement between the simulated results for the XEB circuits, which supports the existence of the RAV efficiency advantage even in the absence of the simplifying assumptions used to derive Equation \( \[6\] \).

We note that the assumption of purely depolarizing errors is not physical, since any real device will be subject to a variety of coherent and stochastic error sources which on their own would not act as a depolarizing channel. However, it is believed that ensembles of random quantum circuits effectively transform even local coherent errors into global depolarizing noise [13, 24], and so it is reasonable to expect that experimental results under various realistic noise sources will show good agreement with the idealized results under the assumption of purely depolarizing noise.
We now discuss our technique for compiling the approximate inversion sequence as part of RAV sequence generation. Compiling an inversion sequence for a given random quantum circuit in a non-trivial manner (i.e., other than simply reversing and inverting the original random sequence) into a sequence of continuously-parameterized gates is a difficult problem that in general is infeasible to solve exactly. To produce the inversion portion of the RAV sequences, we introduce a stochastic protocol for approximate quantum unitary compilation, which we abbreviate as STOQ. We note that this protocol generalizes a similar technique used for variational quantum compilation algorithms [26, 27].

The process of compilation requires specification of the unitary operation to be compiled, known as the target unitary. This is the $2^n$-dimensional unitary operator $U$ implementing some desired effect on the $n$-qubit system. In the case of RAV, the target unitary is the inverse of the product of the initial randomly-generated sequence of layers.

The set of gates used for the compilation may, in general, be fixed or parameterized. Fixed gates, such as Cliffords, are discrete operations that can be represented as a fixed unitary matrix. In contrast, parameterized gates, such as rotations, are continuous operations that can be represented as a unitary matrix with one or more continuously-variable parameters. The allowed set of gates for the compilation may then consist of some combination of fixed and parameterized gates. For an $n$-qubit system, the instruction set (often called native gate set) is a set of fixed gates and/or parameterized gates that represent the fundamental set of operations that can be physically applied to the system.

For a protocol such as RAV, it is desirable that gates occur in the sequence in specific patterns, which we refer to as layers. In this case, we can define the instruction set in terms of these layers of fixed and/or parameterized native gates, such that the resulting compilation will be a sequence of layers. Each layer consists of a fixed number of each type of native gate, where each gate is assigned a randomly-chosen parameter value (within some allowed parameter range) and the order of the gates within the layer is also randomly chosen. In the following description of the STOQ algorithm, we refer to the components of an instruction set as instructions, where each instruction can be either a single gate or a layer, depending on the definition of the instruction set for the given problem.

Given a target unitary $U$ and an instruction set $G$, then, the goal of an approximate compilation protocol is
function StochasticCompilation
    (params U, G, num_iterations):
        sequence := []
        beta := 0
        cost := Cost(U, Prod(sequence))
        for i in 1 to num_iterations:
            new_sequence := RandomChange(sequence, G)
            new_cost := Cost(U, Prod(new_sequence))
            if Accept(cost, new_cost, beta):
                sequence := new_sequence
                cost := new_cost
        return sequence

Figure 4. Pseudocode for STOQ stochastic compilation algorithm. The inputs to the algorithm are the target unitary $U$, the parameterized instruction set $G$, and the number of iterations to perform num_iterations. The algorithm is described in Section II.B with additional implementation details provided in Appendix C.2 to find a sequence of instructions $\{G_1, \ldots, G_M\}$ such that the product $G_MG_{M-1}\cdots G_1$ is as close as possible to $U$ for some reasonable choice of distance metric. Note that this definition does not require any particular closeness of approximation, but it does require that the quality of approximation can be measured. That is, given some appropriate distance metric which defines a distance $d$ between the sequence product $G_MG_{M-1}\cdots G_1$ and the target unitary $U$, an approximate compilation procedure treats $d$ as the value of a cost function to be minimized.

The STOQ protocol for approximate compilation proceeds according to the pseudocode displayed in Figure 4. Intuitively, the STOQ algorithm can be thought of as a randomized exploration of the full space of possible $n$-qubit unitary operators (or the subspace that can be generated by the instruction set $G$, if $G$ is not universal), using a technique known as Markov chain Monte Carlo (MCMC) search [28]. The algorithm is always initialized with an empty sequence, meaning that it always starts from the identity operator in the search space. At each iteration, a random step is proposed, in which an item is either added to or removed from the sequence. If this step brings the product of the sequence closer to the target unitary as determined by the cost function, it is accepted; otherwise, it is either accepted or rejected with some probability, where the probability of accepting such “bad” steps decreases with each iteration. The algorithm continues until some maximum number of iterations is reached, at which point the final cost can be evaluated and the sequence either kept or discarded, depending on the accuracy requirements of the given problem.

One critical component of the algorithm is the choice of an appropriate and efficiently-computable cost function. Naturally, the cost function should be a distance measure between the the target unitary $U$ and the unitary $V$ which is the product of the currently-compiled sequence. One commonly-used and operationally-relevant choice, used also in variational quantum compilation approaches [26, 27], is the Hilbert-Schmidt distance

$$D_{HS}(U, V) = \left|\text{Tr}(V^\dagger U)\right|,$$

which is related to the fidelity of a process [29] and can be shown in certain cases to be closely related to the trace distance [30]. For these reasons, we use the cost function

$$\text{Cost}(U, V) = 1 - \frac{1}{2}D_{HS}(U, V),$$

noting that $\text{Cost}(U, V)$ ranges from 0 to 1 and vanishes if and only if $U$ and $V$ are equivalent up to a global phase.

Additional technical details and discussion of STOQ are provided in Appendix C including demonstrations of using STOQ for compilation of sequences to approximately implement time-evolution unitaries, as well as for approximate compilation of randomly-generated unitaries.

C. QSCOUT experimental setup

The Quantum Scientific Computing Open User Testbed (QSCOUT) is a quantum processor based on trapped ions housed at Sandia National Laboratories [21]. For the experiments shown here, two $^{171}\text{Yb}^+$ ions were used, in which the qubit states are defined by the hyperfine ‘clock’ transition of a $^{171}\text{Yb}^+$ ion, $^2\text{S}_{1/2} \ (|F=0, m_F=0\rangle (|0\rangle))$ and $|F=1, m_F = 0\rangle (|1\rangle))$. The ions are controlled via Raman transitions using a pulsed 355 nm laser in the counter-propagating configuration [31, 32], where one arm is a “global” beam that spans all qubits and the other arm consists of up to 32 individual addressing beams, each of which illuminates a single ion [33]. All beams, individual or global, are controlled via the Sandia developed “Octet” coherent control hardware with complete two-tone frequency, phase, and amplitude control via rf pulses. The individual addressing beams are created by a specialized multichannel acousto-optic modulator (AOM) from L3Harris, which divides a single laser beam into 32 beams and propagates each through a separate AOM crystal.

The single- and two-qubit gates used in the system are all generated via Raman transitions and are parameterized. The single-qubit gates utilize the appropriate individual beam with two tones applied the AOM, generating the necessary transitions in what is known as a co-propagating configuration. Gates about an equatorial axis on the Bloch sphere are physical gates called $R(\theta, \varphi)$, and defined by both $\theta$, which is determined by the duration of the pulse, and $\varphi$, determined by the relative phase of the two tones in the Raman transition. The pulse amplitudes are square-shaped and gapless, meaning there is no “off” time between single qubit gates. Fidelities for physical single-qubit gates, $R(\pi/2, 0)$ and $R(\pi/2, \pi/2)$ have been estimated to be $99.5 \pm 0.3\%$ using a variety of techniques including gate set tomography. The gates
used for this work do not contain any form of compensation, such as SK1 [34], simplifying the bare gate error analysis and reducing the data acquisition time to limit the effects of drift, especially in the case of significant layer sizes. $R_Z(\theta)$ gates are applied virtually by Octet and seen by the qubits as a cumulative phase shift.

Two-qubit gates in the system are Mølmer-Sørensen (MS) interactions of the form $XX(\theta) = e^{-i \frac{\theta}{2} \sigma_x \otimes \sigma_x}$ and are also parameterized. They are defined by a desired phase, $\varphi$ and angle, $\theta$. The MS gate pulses have a Gaussian-shaped amplitude and are composed of one tone on the global beam and two tones each on the individual beams, generating Raman transitions which are detuned symmetrically from a red and blue motional sideband pair.

Unlike the single-qubit gates, the MS gate has a fixed duration of 200 $\mu$s, and the rotation angle $\theta$ is determined by the global beam amplitude, accounting for distortions and saturation effects in the global beam amplifier and AOM. In addition, negative rotation angles are generated by changing the relative phase on one of the two ions by $\pi$ radians. For the purposes of this demonstration, all $MS(\theta, \varphi)$ gates had $|\theta| \leq \pi/10$, and for these small values of $\theta$, calibrations suggest deviations in the resultant rotation angle of $\lesssim 8\%$.

Additionally, the MS gates also account for the AC Stark effect through the use of frame rotations, which are virtual Z rotations, applied during the MS gate to cancel phase accumulation from the AC Stark effect. As the global beam and individual beams both contribute to phase shifts caused by the AC Stark shift, the frame rotation also changes depending on the desired $\theta$. These are calibrated to within $\pm 3.5 \times 10^{-3}$ radians, or $\pm 0.2$ degrees, for the range of $\theta$ used.

The MS gates are performed in a counter-propagating beam configuration while the single-qubit gates are performed in a co-propagating configuration. Because the relative phase stability between the counter-propagating beams is less stable than that of co-propagating beams, intermixing gates from the two configurations leads to unpredictable phase relationships. To combat this instability, we perform basis transformations on all two-qubit gates. We first surround the MS gate with counter-propagating single-qubit $\pi/2$ gates to transform an $XX$ interaction into a $ZZ$ interaction [32]. We then further surround those with co-propagating single-qubit gates to bring the interaction back onto an equatorial axis. The desired phase of the MS gate, $\varphi$, is then introduced through the respective phase of these co-propagating single-qubit gates. When including the basis transformation gates, we estimate fidelities for $MS(\pi/2, 0)$ to be $97 \pm 1\%$.

Due to the nature of the RAV sequences, some extra consideration was needed to deal with non-standard sequences. Low-level pulse data is compressed and stored in a series of lookup tables (LUTs) in Octet’s programmable logic for fast readout of data-intensive gate sequences [39]. The topology of these LUTs and the compression scheme is designed to leverage redundant gate information common to a wide array of quantum circuits. Because of the numerous unique gate calls in RAV sequences, the compression ratio is limited and more LUT storage is required. While storage was increased for certain LUTs using a custom addressing scheme for dense packing of data (not limited to byte-write boundaries), finite memory availability is still a limitation.

Increasing the LUT storage was supplemented by a compilation technique developed to partially reprogram large segments of the LUTs mid circuit. In this work, the RAV sequences used numerous virtual $R_Z$ gates. Due to their virtual nature, $R_Z$ gates are typically on the order of 10 ns, and continuous streaming of raw data for such short gates exceeds the maximum data throughput supported by the device. The compiler was set up to run recursively, essentially breaking long circuits into smaller pieces based on where the LUT capacity was exceeded. To prevent underflow conditions, partial reprogramming data was placed after gates with long durations by strategic adjustment of the initial boundaries determined by the compiler.

III. RESULTS

A. Numerical demonstrations

To demonstrate the efficiency advantage of RAV for verification of continuously-parameterized gates, we generated 50 RAV and 50 XEB sequences of varying lengths for a five-qubit system. We generated these sequences using a continuously-parameterized native gate set $\{R(\theta, \varphi), R_Z(\theta), MS(\theta, \varphi)\}$, where each layer consists of three $R(\theta, \varphi)$ gates, three $R_Z(\theta)$ gates, and one $MS(\theta, \varphi)$ gate. The target qubit(s) for each gate are chosen uniformly at random from the set of all qubits in the system. Values for each $\theta$ rotation angle are chosen uniformly at random in the interval $[-\pi/10, \pi/10]$, and values for each $\varphi$ axis angle are chosen uniformly at random in the interval $[-\pi, \pi]$. Detailed descriptions of the instruction set and layer construction are provided in Appendix D.

Figure 5 depicts the mean and standard deviation of error rates per layer obtained via both RAV and XEB for sequences under varying simulated depolarization rates. We observe that, over a wide range of depolarization rates, the mean fitted error rates from RAV and XEB are closely aligned, but the error rates estimated by RAV have a significantly smaller standard deviation than those obtained via XEB, indicating that RAV provides more precise information about the overall error rate of these sequences. We also observe that this advantage is more significant in the regime of lower depolarization rates. For a depolarization rate around $10^{-2}$, the RAV error rate estimates are roughly twice as precise as the XEB error rate estimates, whereas around $10^{-4}$, they are roughly three times as precise.
We note that because the RAV efficiency advantage comes partially from reduced quantum projection noise due to measurement, we expect that we will find the largest advantage when operating in the early part of the RAV decay curve (which in this simulation is realized by lower depolarization rate). This is the regime where the RAV sequence results remain closest to a basis state, where quantum projection noise is minimized.

**B. Experimental demonstrations**

To demonstrate the RAV efficiency advantage experimentally, we generated RAV and XEB sequences of varying lengths for a 2-qubit system using the same native gate set as in Section IIIA. In this subsection, we report the results from executing these sequences on quantum processors from QSCOUT and IBM Q.

1. **QSCOUT trapped-ion processor**

As a first experimental demonstration, we executed 50 RAV and 50 XEB sequences on the two-qubit trapped-ion processor at the Quantum Scientific Computing Open User Testbed (QSCOUT) operated by Sandia National Laboratories [21]. Details of the experiment are provided in Section IIIC. We note that this device directly implements the parameterized native gate set \( \{ R(\theta, \phi), R_Z(\theta), MS(\theta, \varphi) \} \) that we used to generate these sequences.

Figure 6(a) and Figure 6(b) show the results of 20 independent runs of the entire set of RAV and XEB sequences, using \( K = 25 \) shots per experiment. (Note that for experimental simplicity, 500 executions were performed for each sequence, and we then separated the shot-level results and interpreted them as 500/\(K \) independent runs of \( K \) shots each.) It is clear visually that the XEB fit curves vary significantly more from the mean than the RAV fit curves. The standard deviations of these fitted error rates for various values of \( K \) are plotted in Figure 6(c), and the corresponding statistics are tabulated in Figure 6(d). As expected, we observe that the error rates obtained via RAV runs have a significantly smaller standard deviation (by a factor of 2.5 to 5) than those obtained from XEB runs. Since the standard deviation of the fidelity estimate goes as \( 1/\sqrt{K} \) (which is supported by the data in Figure 6(c)), this implies that XEB would require approximately 6 to 25 times as many experimental shots as RAV to produce an error rate estimate for this device with equivalent precision. An example of this can be seen visually by the dashed line in Figure 6(c), which shows that the RAV \( K = 5 \) runs provide a more precise error rate estimate than the XEB \( K = 50 \) runs.

2. **IBM Q superconducting processor**

To provide further experimental results, we executed 72 RAV and 72 XEB sequences on the publicly-available IBMQ_Manila superconducting processor from IBM [22]. We note that this device does not directly implement the parameterized native gate set \( \{ R(\theta, \phi), R_Z(\theta), MS(\theta, \varphi) \} \). To adapt the sequences for this device, we translated the sequences from the original parameterized native gate set into the instruction set...
that is accepted by the IBM Q framework. More specifically, $R(\theta, \varphi) = U(\varphi, \theta - \pi/2, \pi/2 - \theta)$ and $MS(\theta, \varphi) = R_Z(-\varphi)^{\otimes 2}R_X(\theta)R_Z(\varphi)^{\otimes 2}$, where $R_X(\theta)$ is itself a composite gate that can be implemented through multiple native gates. $R_Z(\theta)$ is implementable directly and does not need to be translated.

Figure 7(a) and Figure 7(b) show the results of 10 independent runs of the entire set of RAV and XEB sequence, using $K = 1000$ shots per experiment. XEB curves are visually more dispersive than RAV curves, which indicates RAV’s smaller uncertainty in estimating fidelity. The advantage is verified for different numbers of shots, as is shown in Figure 7(c). Figure 7(d) tabulates the specific statistics. The standard deviation of the fidelity estimates from RAV experiments is about one to three times smaller than that from XEB experiments, which implies that to achieve the same accuracy of fidelity estimation, XEB would take more experimental resources than RAV.

Based on the IBM Q-published gate error rates at the time the experiments were run, we can calculate that the error per layer of our RAV and XEB sequences should be about $1.57 \times 10^{-2}$, which corresponds well with the experimental results. We also note that the IBM Q results in Figure 7(c) do not demonstrate the expected scaling of the standard deviation of the fidelity estimate as $1/\sqrt{K}$. We believe that this is because the true error rate of the ibmq_manila device was fluctuating during the course of the runs, which were spread over the course of several hours. This led directly to increased variance in fitted error rates for some of the experiments. Therefore, our estimated error rate variances are including the effects of these physical fluctuations in addition to the inherent variance from the RAV and XEB estimates. We expect that runs in which all of the sequences are executed in rapid succession would help to reduce the effect of such fluctuations.

### IV. DISCUSSION

We note that because of its efficiency advantages, RAV may be particularly useful in the context of frequent calibration runs for devices with continuously-parameterized
Figure 7. Experimental results from IBM Q ibmq_manila device for two-qubit RAV and XEB runs. Each run consists of 72 sequences executed $K$ times each. (a,b) Results of 10 independent RAV (or XEB) runs using $K = 1000$ shots per sequence. Colored data points indicate the fidelity estimate $\hat{F}_{\text{RAV}}$ (or $\hat{F}_{\text{XEB}}$) for $K$ shots of a single sequence, calculated according to Equation 2 (or Equation 4). Thin colored curves are three-parameter exponential decay fits for the data points from each run. Data points and fit curves from a particular run are assigned the same color. The thick curve is the mean of the individual fit curves. (c) Standard deviation of the fitted error per layer for RAV and XEB runs for various values of $K$. Smaller standard deviation indicates a more precise estimate of the error rate. (d) Fitted error per layer statistics for RAV and XEB runs for various values of $K$.

We demonstrate these benchmarking approaches on continuously-parameterized gate sets of both a trapped-ion system, QSCOUT, and a superconducting system, IBM Q. In particular, the gate sets used in this demonstration more closely align with the native physical gates of trapped-ion systems. In QSCOUT, one circuit layer contains a single physical two-qubit gate, where the parameterized $\theta$ is directly associated with physical inputs to that gate; however, on the IBM Q system, one circuit layer contains 2 physical two-qubit gates, and the parameterized $\theta$ is instead passed into the surrounding single-qubit gates. As such, the particular RAV construction demonstrated here provides an error rate that may provide more direct assessment of physical two-qubit controls for the trapped-ion QSCOUT system, yet still provides an error rate for the superconducting IBM Q system that is comparable to their published rate. Additionally, RAV could easily be adapted to include gates more closely aligned with the native physical gates of any system, including superconducting systems.

We also detail the experimental realization of continuously-parameterized two-qubit gates (and relevant calibration bounds to these gates) on the QSCOUT system. We note that additional control-system developments, in the form of extra storage and partial reprogramming, were needed to support the lengthy sequences of unique gate instantiations inherent to verification techniques for continuously-parameterized gate sets. With these capabilities, the RAV and XEB sequences were reasonably practical to experimentally implement on a trapped-ion system such as QSCOUT.
We have demonstrated the generation of RAV sequences on systems of up to $n = 8$ qubits. The bottleneck in the RAV sequence generation is the compilation of the approximate inversion sequence via STOQ, which scales poorly with system size (see Appendix C5) and is unlikely to be feasible for $n \gg 10$ qubits. However, this is not an inherent limitation of RAV. If a more efficient technique can be applied to generating the approximate inversion sequence, then RAV sequences could be generated for larger systems. We believe that one promising area of future work would be to adapt the efficient inversion techniques from mirror RB [37] to the context of RAV and verification of continuously-parameterized approximation techniques from mirror RB [37] to the context of RAV.

This would likely require some restrictions on the construction of layers of the generated RAV circuits, such as requiring that each layer approximately implements a Clifford operation, so that the mirroring can be done efficiently.

**CODE AVAILABILITY**

A Python implementation of the STOQ compilation protocol, which includes functionality to generate RAV sequences, is available at https://github.com/rmshaffer/stoq-compiler.

**ACKNOWLEDGEMENTS**

R.S., H.R., E.D., and H.H. acknowledge support from the Challenge Institute for Quantum Computation (CIQC) via the NSF Quantum Leap Challenge Institute (QLCI) program under grant number OMA-2016245, from the Army Research Office under grant number W911NF-18-1-0170, and from the NSF STAQ project under grant number PHY-1818914. R.S. acknowledges support from the QISE-NET fellowship under NSF award DMR-1747426, as well as from the National Defense Science and Engineering Graduate (NDSEG) fellowship under contract FA9550-11-C-0028 and awarded by the Department of Defense, Air Force Office of Scientific Research, 32 CFR 168a.

This material was also funded in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Quantum Testbed Program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND2022-7107O.

We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.
Appendix A: Derivation of RAV fidelity estimate

In this appendix, we derive the formula for the approximate fidelity of a RAV sequence on an n-qubit system. We start with the XEB fidelity estimate in Equation [1], where \( P(x) \) represents the classically-computed ideal output probability distribution for the sequence, \( Q(x) \) is the observed sample probability of obtaining measurement result \( x \), and \( N = 2^n \) is the dimension of the system. The RAV sequence is constructed to return nearly all of the population to the initial state \( x_0 \). Therefore, we let \( P(x_0) = 1 - \epsilon \) for some small \( \epsilon \ll 1 \), which represents the approximation error of the inversion sequence. As a further simplification, we assume that the remaining probability is spread evenly among the remaining states, such that \( P(x) = \frac{\epsilon}{N-1} \) for each \( x \neq x_0 \).

We can then derive the RAV fidelity from the XEB fidelity formula as follows:

\[
\hat{F}_{\text{RAV}} = \frac{\sum_x P(x)Q(x) - \frac{1}{N}}{\sum_x P(x)^2 - \frac{1}{N}}
\]

\[
= \frac{P(x_0)Q(x_0) + \sum_{x \neq x_0} P(x)Q(x) - \frac{1}{N}}{P(x_0)^2 + \sum_{x \neq x_0} P(x)^2 - \frac{1}{N}}
\]

\[
= \frac{(1 - \epsilon)Q(x_0) + \frac{N}{N-1}(1 - Q(x_0)) - \frac{1}{N}}{(1 - \epsilon)^2 + \frac{1}{N-1} \epsilon^2 - \frac{1}{N}}
\]

\[
= \frac{NQ(x_0) - 1}{N-1} + \epsilon \frac{N(NQ(x_0) - 1)}{(N-1)^2} + \epsilon^2 \frac{N^2(Q(x_0) - 1)}{(N-1)^3} + \ldots
\]

\[
= \frac{NQ(x_0) - 1}{N-1} \left[ 1 + \frac{N\epsilon}{N-1} + \left(\frac{N\epsilon}{N-1}\right)^2 + \ldots \right]
\]

where in the next-to-last step we have performed a Taylor expansion around \( \epsilon = 0 \). Then, in the final step we note that the expression inside the square brackets is just a geometric series in \( \frac{N\epsilon}{N-1} \), and therefore we have

\[
\hat{F}_{\text{RAV}} = \frac{NQ(x_0) - 1}{N-1} \left[ \frac{1}{1 - \frac{N\epsilon}{N-1}} \right]
\]

\[
= \frac{Q(x_0) - \frac{1}{N}}{(1 - \epsilon) - \frac{1}{N}}
\]

which, using the fact that \( P(x_0) = 1 - \epsilon \), becomes:

\[
\hat{F}_{\text{RAV}} = \frac{Q(x_0) - \frac{1}{N}}{P(x_0) - \frac{1}{N}}
\]

Appendix B: Derivation of variance in RAV and XEB fidelity estimates

In this appendix, we derive formulas for the variance of the fidelity estimates resulting from \( K \) independent shots of a single RAV or XEB circuit. We start by assuming that the errors in our device are purely depolarizing. Under this assumption, we can represent the output state of any \( n \)-qubit circuit as a mixture of the circuit’s ideal output state \( |\psi\rangle\langle\psi| \) and the maximally-mixed state \( \frac{1}{N} I \) (where \( N = 2^n \), where \( \lambda \in [0, 1] \) is the fraction to which the output state is depolarized:

\[
\rho_\lambda = (1 - \lambda)|\psi\rangle\langle\psi| + \frac{\lambda}{N} I
\]

We can then define \( P(x) \) and \( Q_\lambda(x) \) as the probabilities of measuring outcome \( x \) when measuring the states \( |\psi\rangle\langle\psi| \) and \( \rho_\lambda \), respectively, as:

\[
P(x) = \langle x|\psi\rangle\langle\psi|x\rangle = |\langle x|\psi\rangle|^2
\]

\[
Q_\lambda(x) = \langle x|\rho_\lambda|x\rangle = (1 - \lambda)P(x) + \frac{\lambda}{N}
\]
We can restate Equation 1 and Equation 2 to define the fidelity estimates $\hat{F}_{\text{RAV}}$ and $\hat{F}_{\text{XEB}}$ in terms of $P(x)$ and $Q_\lambda(x)$ as follows:

$$\hat{F}_{\text{RAV}} = \frac{Q_\lambda(x_0) - \frac{1}{N}}{P(x_0) - \frac{1}{N}} \quad (\text{B4})$$

$$\hat{F}_{\text{XEB}} = \frac{\sum_x P(x)Q_\lambda(x) - \frac{1}{N}}{\sum_x P(x)^2 - \frac{1}{N}} \quad (\text{B5})$$

Now, to calculate the expected variance of our these fidelity estimates, we first need to determine their distribution. To do this, we let $Q_{\lambda,x} \sim \text{Multinomial}(K, N, Q_\lambda(x))$ be a random variable representing the number of times outcome $x$ is observed when taking $K$ independent shots of a RAV or XEB circuit, where $Q_\lambda(x)$ represents the “true” experimental probability distribution of each of $N$ possible outcomes when measuring the state $\rho_\lambda$.

By the properties of the multinomial distribution, then, the variance of $Q_{\lambda,x}$ is:

$$\text{Var}[Q_{\lambda,x}] = KQ_\lambda(x)[1 - Q_\lambda(x)]$$

$$= K \left[ (1 - \lambda)P(x) + \frac{\lambda}{N} \right] \left[ 1 - (1 - \lambda)P(x) - \frac{\lambda}{N} \right]. \quad (\text{B6})$$

We can now construct random variables corresponding to measurements of $\hat{F}_{\text{RAV}}$ and $\hat{F}_{\text{XEB}}$ by replacing $Q_\lambda(x)$ in Equation B4 and Equation B5 with the scaled random variable $\frac{1}{N}Q_{\lambda,x}$, which represents the sample probability of observing outcome $x$ when taking $K$ independent shots:

$$\hat{F}_{\text{RAV}} = \frac{\frac{1}{N}Q_{\lambda,x_0} - \frac{1}{N}}{P(x_0) - \frac{1}{N}} \quad \hat{F}_{\text{XEB}} = \frac{\sum_x P(x)\frac{1}{N}Q_{\lambda,x} - \frac{1}{N}}{\sum_x P(x)^2 - \frac{1}{N}} \quad (\text{B8})$$

Once we have done this, calculating the variance of these random variables is straightforward algebra:

$$\text{Var}[\hat{F}_{\text{RAV}}] = \frac{1}{K^2} \left( \frac{1}{P(x_0) - \frac{1}{N}} \right)^2 \text{Var}[Q_{\lambda,x_0}]$$

$$= \frac{1}{K} \left( \frac{1}{P(x_0) - \frac{1}{N}} \right)^2 \left[ (1 - \lambda)P(x_0) + \frac{\lambda}{N} \right] \left[ 1 - (1 - \lambda)P(x_0) - \frac{\lambda}{N} \right] \quad (\text{B9})$$

$$\text{Var}[\hat{F}_{\text{XEB}}] = \text{Var} \left[ \frac{\frac{1}{N} \sum_x P(x)Q_{\lambda,x} - \frac{1}{N}}{\sum_x P(x)^2 - \frac{1}{N}} \right]$$

$$= \frac{1}{K^2} \left( \frac{1}{\sum_x P(x)^2 - \frac{1}{N}} \right)^2 \text{Var}[Q_{\lambda,x}]$$

$$= \frac{1}{K^2} \left( \frac{N}{\sum_x P(x)^2 - 1} \right)^2 \sum_x P(x)^2 \left[ (1 - \lambda)P(x) + \frac{\lambda}{N} \right] \left[ 1 - (1 - \lambda)P(x) - \frac{\lambda}{N} \right] \quad (\text{B10})$$

The above formulas are sufficient to predict the variance in fidelity measurements for a particular circuit, assuming we know the ideal probabilities $P(x)$ of measuring the circuit output. But of course, these probabilities will be different for every circuit. To make more general predictions, we need to make additional assumptions.

For RAV, the sequences have been explicitly constructed such that most of the population is returned to the initial state $x_0$. Therefore, we assume that $P(x_0) \approx 1 - \epsilon$ for some small $\epsilon \ll 1$, which represents the approximation error of the inversion sequence. Substituting this into Equation B10 gives

$$\text{Var}[\hat{F}_{\text{RAV}}] \approx \frac{1}{K} \left( \frac{1}{(1 - \epsilon) - \frac{1}{N}} \right)^2 \left[ (1 - \lambda)(1 - \epsilon) + \frac{\lambda}{N} \right] \left[ 1 - (1 - \lambda)(1 - \epsilon) - \frac{\lambda}{N} \right]. \quad (\text{B15})$$
For XEB, it is known that for systems with tens of qubits and circuits of depth \( \gtrsim 10 \), the distribution of ideal output state probabilities can be well-approximated by a Porter-Thomas distribution [13], in which the probabilities follow an exponential distribution \( \Pr(N_p) = Ne^{-N_p} \). By the properties of the exponential distribution, then, we have \( \sum_x P(x)^k \approx \frac{1}{k} \), and substituting this into Equation B14 gives

\[
\text{Var}[\hat{F}_{\text{XEB}}] \approx \frac{1}{K} \left( \frac{N}{2N-1} \right)^2 \left[ \frac{1}{2} \left( \frac{\lambda}{N} \right) \left( 1 - \frac{\lambda}{N} \right) + \frac{1}{3} \right] (1 - \lambda) \left( 1 - \frac{2\lambda}{N} \right) - \frac{1}{4} (1 - \lambda)^2 .
\] (B16)

It is important to note that in Figure 3 we are working with simulations of \( \leq 8 \) qubits, and so the Porter-Thomas assumption is not necessarily valid in this regime. This is the most likely explanation for the systematic discrepancies between this estimate of the variance and the observed variance in our simulations.

Appendix C: Approximate unitary compilation via stochastic search (STOQ)

In this appendix, we supplement the main text with additional implementation details of STOQ and examples of its use. We report results of applying STOQ to Hamiltonian time-evolution unitaries, where we compare its performance to existing methods on various metrics. We also demonstrate the use of STOQ to approximately compile gate sequences for randomly-generated unitaries, although as one would expect, its performance scales poorly with the required circuit depth. We conclude with additional discussion of STOQ, including its features and limitations.

1. Background

A critical prerequisite to executing any algorithm on a physical quantum computer is the process commonly known as quantum compilation. One of the primary tasks of quantum compilation is the conversion of a target unitary operation into a sequence of quantum gates that are native to the physical device being used [38–41]. Because unitary operators belong to a continuous space, such compilation in general results in gate sequences which are only approximately equivalent to the target unitary. For example, one of the earliest quantum compilation techniques, the Solovay-Kitaev method [42], compiles gate sequences that differ from the target unitary by an amount that can be made as small as desired.

Traditional compilation, both in the classical and quantum realms, is most often a deterministic process, using rules and heuristics to efficiently synthesize a desired program from the native assembly instructions (in classical compilation) or native physical gates (in quantum compilation). But in some cases, adding stochasticity to the compilation process has been shown to produce advantages in the resulting program. In classical compilation, a technique known as stochastic superoptimization [43] has been shown in certain cases to produce significantly shorter programs than the best-in-class compilers and optimizers. In quantum compilation, techniques such as randomized compiling [44] have been demonstrated to improve noise resilience by randomizing errors that occur during program execution.

In the field of quantum compilation, special attention has been paid to compilation of unitaries which result from the time-evolution of physically-realizable Hamiltonians. The compiled sequences in these cases can be executed to perform what is known as “Hamiltonian simulation”, or more broadly, “quantum simulation”. Such approaches are of special interest in fields such as quantum chemistry, where it is desirable to use a quantum computer to simulate the dynamics of physical systems. Common approaches to this problem include product formula techniques such as the Suzuki-Trotter decomposition [45] and qubitization [46], which deterministically compile the time-evolution unitary for a given Hamiltonian into a sequence of quantum gates.

Approaches involving stochasticity have recently been shown to be advantageous in some cases. Adding randomization to the Suzuki-Trotter decomposition [47] creates approximate compilations that are better both theoretically and empirically. A stochastic compilation protocol known as QDRIFT [48], where gate probabilities are weighted according to the strength of each term in the Hamiltonian rather than using a product formula directly, has been shown to produce much more efficient compilations in many cases. An interpolation of these two methods [49] has also been proposed, which takes some of the advantages of each method. The efficiency of these compilation methods is generally independent of system size when applied to problems involving sparse Hamiltonians.

However, these specialized methods cannot be applied to general-purpose compilation tasks, which is where we focus specifically here. In Section IIB of the main text, a stochastic approximate quantum unitary compilation procedure, abbreviated as STOQ, is described in detail. The STOQ protocol was originally developed as part of a verification scheme for analog quantum simulators called randomized analog verification (RAV) [20]. And in this work, an adaptation of RAV for gate-based quantum devices is summarized in Section II A, demonstrated numerically.
in Section IIIA and demonstrated experimentally in Section IIIB. This gate-based version of RAV also uses STOQ to compile the approximate inversion portion of each sequence.

2. Additional implementation details

This section fills in a few important details of the STOQ protocol implementation outlined in Section IIIB, specifically referring to the pseudocode representation in Figure 4.

The compiled sequence is stored in the sequence variable, which is initially empty. The RandomChange function returns a modified sequence on each iteration, either by adding a randomly-drawn instruction to the sequence from the parameterized instruction set $G$ with randomly-generated parameter values, or by removing an instruction from the sequence. The Prod function calculates the unitary that represents the product of all of the operations in the sequence, and the Cost function is implemented as described in Equation 8.

The variable beta is used as an annealing parameter for the compilation process. The function IncreaseBeta returns a slightly increased value of beta on each iteration. Defining the annealing parameter as $\beta = beta$ and the cost difference of such a proposed change as $\Delta = new\_cost - cost$, the Accept function calculates the probability of accepting a proposed change as

$$P_{accept} = \begin{cases} e^{-\beta\Delta} & \Delta > 0 \\ 1 & \Delta \leq 0 \end{cases}$$

(C1)

The probability of accepting “bad” proposed changes where the cost increases (i.e., where $\Delta > 0$) approaches zero as $\beta$ increases.

3. Compilation of time-evolution unitaries

To demonstrate one possible (although not necessarily useful) application of STOQ, we choose an Ising-type Hamiltonian with nearest-neighbor coupling and transverse field

$$H = \sum_{<i,j>} J_{ij} \sigma_x^{(i)} \sigma_x^{(j)} + \sum_i h_i \sigma_y^{(i)}$$

where the coefficients $J_{ij}$ and $h_i$ are energies with particular values for each system size, as shown in Table I.

We then define the time-evolution unitary as $U_i(\tau) = e^{iH\tau}$, where we choose units such that $\hbar = 1$, and we concretely choose $\tau = 0.5$ ms, such that

$$U = U_i(0.5 \text{ ms}) = e^{iH(0.5 \text{ ms})}$$

is the target unitary for compilation.
To apply STOQ, we need also to choose a parameterized instruction set $G$ from which to approximately compile a sequence. In a physical device, it is often the case that the dynamics are implemented such that each term in $H$ can be individually controlled. To define $G$ for such a device, we express the Hamiltonian as $H = \sum_k H_k$, where each $H_k$ is one of the $\sigma_x \sigma_x$ or $\sigma_y$ terms from Equation C2 and choose

$$G = \bigcup_k \{ e^{iH_k t} \} \quad -\epsilon \tau \leq t \leq \epsilon \tau$$

(C4)

where the allowed range for $t$ is chosen such that the duration of each instruction is relatively short in comparison to the timescale of the dynamics of $H$. (In this demonstration we use $\epsilon = 0.2$.) Negative times correspond to reversing the sign of the coefficient of a given term. (We note that for a general Hamiltonian, it is unlikely that each $H_k$ term is part of the native gate set of the device. In this case, $G$ should instead represent the interactions which are implemented natively on the device.)

We then apply STOQ to compile many sequences that approximately implement $U$, using two-qubit, three-qubit, five-qubit, and eight-qubit versions of the corresponding Hamiltonian. Figure 8 reports the cost for 16 such compilations as a function of the number of iterations. (Each run of 10,000 iterations for the five-qubit system takes around 15 minutes to complete on a typical desktop computer.) We observe that the stochastic search process rapidly reduces the cost at first before noticeably leveling off. For the two-qubit and three-qubit systems, this cost approaches a limit near $10^{-2}$ after 10,000 iterations. For the larger systems, the final average cost is higher, although even for the eight-qubit system, the final cost reaches a value below $10^{-1}$ for some compilations.

To compare STOQ to existing compilation techniques, we also compile sequences to approximately implement $U$ using the randomized Suzuki-Trotter decomposition \[17\] and the QDRIFT stochastic compilation protocol \[48\]. STOQ is designed to create more randomness in the resulting path taken through state space. To compare these paths quantitatively, we choose to compare the various methods to an ideal version where $H$ is directly implemented for time $\tau$. We define the ideal path as the path taken by this ideal time evolution, and we define the compiled path as the path taken by the compiled sequence, which we represent as a sequence of instructions $\{G_1, \ldots, G_M\}$. We then calculate the path distance $d_m$ from the ideal path to step $m$ of the compiled path, where $1 \leq m \leq M$, as

$$d_m = \min_{t \in [0, \tau]} D_{\text{HS}}\left( e^{iHt}, G_m G_{m-1} \cdots G_1 \right),$$

(C5)

where $D_{\text{HS}}$ is the Hilbert-Schmidt distance defined in Equation 7. Thus $d_m$ is the shortest distance from step $m$ of the compiled path to any point in the ideal path.

Results for each compilation technique are shown in Figure 8 and statistics for the five-qubit example are displayed in Table I. We observe that the STOQ compilations result in a significantly greater path distance from the ideal evolution than the other approaches, and that the total running time of the compiled sequence resulting from the various compilations is within a factor of two.

However, the final cost of the STOQ compilations is typically at least an order of magnitude larger than the compilations created using the randomized Suzuki-Trotter and QDRIFT techniques, both of which can reach arbitrarily low costs by increasing the number of steps. This implies that STOQ would not be a useful tool for applications that require high-fidelity compilations.

### 4. Compilation of random unitaries

In addition to being used for sparse or highly structured unitaries such as those generated from Hamiltonian time-evolution, the STOQ protocol can also be used to compile sequences that approximately implement purely random unitaries in terms of an arbitrary instruction set, without having any prior knowledge of the structure of the unitary. Such an application of STOQ is not necessarily useful in general, but is included here for illustrative purposes.

---

### Table I. Coefficients used for application of STOQ to the $n$-qubit Ising model Hamiltonian in Equation C2

| $n$ | $J_{12}$ | $J_{23}$ | $J_{34}$ | $J_{45}$ | $J_{56}$ | $J_{67}$ | $J_{78}$ | $h_1$ | $h_2$ | $h_3$ | $h_4$ | $h_5$ | $h_6$ | $h_7$ | $h_8$ |
|-----|----------|----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|
| 2   | 1.27     |          |          |          |          |          |          | 1.54  | 1.19  |        |       |       |       |       |       |
| 3   | 1.81     | 1.27     |          |          |          |          |          | 1.54  | 1.19  | 0.53  |       |       |       |       |       |
| 5   | 1.20     | 1.40     | 1.60     | 1.80     |          |          |          | 1.60  | 1.30  | 1.00  | 0.70  | 0.40  |       |       |       |
| 8   | 1.20     | 1.30     | 1.40     | 1.50     | 1.60     | 1.70     | 1.80     | 1.40  | 1.10  | 0.80  | 1.00  | 1.20  | 1.50  | 1.70  | 1.30  |

Values are energies in kHz where $\hbar = 1$. 

---

In addition to being used for sparse or highly structured unitaries such as those generated from Hamiltonian time-evolution, the STOQ protocol can also be used to compile sequences that approximately implement purely random unitaries in terms of an arbitrary instruction set, without having any prior knowledge of the structure of the unitary. Such an application of STOQ is not necessarily useful in general, but is included here for illustrative purposes.
Figure 9. Distance from ideal path to compiled path, as defined in Equation C5 for the time-evolution unitary from Equation C3, illustrating “how different” a particular compilation is from the path that would be followed by the ideal time evolution of the full Hamiltonian. The distance from the horizontal axis quantifies the deviation in the system state from the ideal simulation when executing the specified compilation. The ideal simulation would be a flat curve along the horizontal axis from \( \tau = 0.0 \) ms to \( \tau = 0.5 \) ms. Results are shown for 2-qubit, 3-qubit, 5-qubit, and 8-qubit implementations of the Ising model Hamiltonian from Equation C2. Each curve represents the execution of one compiled sequence. Filled squares are used to plot the overall running time of the compiled sequence and final cost of each compilation. Top row depicts the execution of 16 independent STOQ compilations, each using 10,000 iterations. Each curve corresponds to a curve of the same color in Figure 8. Middle row depicts the execution of a typical randomized Suzuki-Trotter compilation using 10 steps. Bottom row depicts the execution of a typical QDRIFT compilation using 1,000 repetitions. The horizontal axis of each plot represents the execution time of the compiled circuits.

Table II. Statistics resulting from various compilations of the five-qubit time-evolution unitary from Equation C3, where the ideal evolution occurs for \( \tau = 0.5 \) ms. For each of the compilation techniques, means are listed for each of the following quantities: (a) total execution time of the compiled sequence, (b) average distance \( \sum_{m=1}^{M} d_m \), (c) maximum distance \( \max_{m \in [1, M]} d_m \), and (d) final cost \( d_M \). Corresponds to five-qubit plots in Figure 9.

|          | Ideal | Trotter | QDRIFT | STOQ |
|----------|-------|---------|--------|------|
| (a) Execution time (ms) | 0.50  | 4.50    | 5.50   | 7.32 |
| (b) Average distance     | —     | 0.0032  | 0.0053 | 0.0469 |
| (c) Maximum distance     | —     | 0.0056  | 0.0099 | 0.1133 |
| (d) Final cost           | —     | 0.0003  | 0.0077 | 0.0328 |

Figure 10 shows typical results of repeatedly using the STOQ protocol to compile sequences for random two-qubit, three-qubit, and five-qubit unitaries, generated according to [50], using a simple universal instruction set \( G = \{ R(\theta, \varphi), XX(\theta) \} \). \( R(\theta, \varphi) \) is a parameterized single-qubit rotation

\[
R(\theta, \varphi) = \begin{bmatrix}
\cos \frac{\theta}{2} & -ie^{i\varphi} \sin \frac{\theta}{2} \\
-ie^{-i\varphi} \sin \frac{\theta}{2} & \cos \frac{\theta}{2}
\end{bmatrix}
\]  

(C6)
Figure 10. Compilation via STOQ of randomly-generated unitaries. The left three plots show the cost during the STOQ compilation process for randomly-generated 2-qubit, 3-qubit, and 5-qubit target unitaries. Each of the 20 thin curves shows the value of the cost function from Equation 8 during a single compilation using 100,000 iterations. The thick curve is the average final cost of all runs. The rightmost plot shows the final cost of the STOQ compilation for target unitaries generated by creating random 5-qubit circuits of varying average circuit depth. Circuit depth is calculated as the total number of instructions divided by the number of qubits. Each point is the average of 20 compilations using 100,000 iterations. Error bars indicate standard error of the mean. The solid line is an exponential decay fit with one free parameter. The dashed line represents the average final cost of compiling a randomly-generated 5-qubit unitary. Note that one would expect a similar scaling with circuit depth for general quantum circuits, regardless of whether they are generated randomly.

with \(0 \leq \theta < 2\pi\) and \(0 \leq \varphi < 2\pi\). \(XX(\theta)\) is a parameterized two-qubit entangling gate

\[
XX(\theta) = \begin{bmatrix}
\cos \theta & 0 & 0 & -i \sin \theta \\
0 & \cos \theta & -i \sin \theta & 0 \\
0 & i \sin \theta & \cos \theta & 0 \\
-i \sin \theta & 0 & 0 & \cos \theta \\
\end{bmatrix}
\]  

(C7)

with \(0 \leq \theta < 2\pi\). We note that the instruction set \(G\) is a typical native gate set that can be implemented by trapped-ion quantum devices.

We observe that the final costs of compilation of these random unitaries are significantly larger than for compilation of the time-evolution unitaries discussed in Appendix C3. In particular, the final cost is approximately 0.1 for two-qubit random unitaries, 0.5 for three-qubit random unitaries, and 0.8 for five-qubit random unitaries. This indicates that the quality of the approximation for such random unitary compilations scales poorly with system size. This is not surprising, since reaching the vast majority of states in the Hilbert space of a system requires circuits of depth which grows exponentially with the dimension of the Hilbert space [51, 52]. Nonetheless, the compilations generated by this method may be useful in scenarios where high-fidelity approximations are not required.

We also observe that the final cost of such random unitary compilations is relatively stable over a wide range of STOQ parameter values. Two primary parameters that can be adjusted in the STOQ algorithm in Figure 14 are the annealing rate \(\Delta \beta\), which is used to increment \(\beta\) at each step inside the \texttt{IncreaseBeta} function, and the probability \(p_{\text{append}}\) that the search appends an instruction (as opposed to removing an instruction) at each step, which occurs inside the \texttt{RandomChange} function. For compilation of three-qubit random unitaries, and for values \(\Delta \beta \in \{0.001, 0.01, 0.1, 0.5\}\) and \(p_{\text{append}} \in \{0.2, 0.5, 0.8\}\), we find that the average final cost remains between 0.398 (for \(\Delta \beta = 0.5\) and \(p_{\text{append}} = 0.2\)) and 0.448 (for \(\Delta \beta = 0.001\) and \(p_{\text{append}} = 0.5\)), where each pair of parameter values is averaged over 32 compilations using 100,000 iterations each.

To provide insight into the low-fidelity approximations of random unitaries produced by STOQ, we consider the case of target unitaries generated by random circuits of varying depth. To do this, we generate random five-qubit circuits of average depth ranging from 1 to 40, where the average depth is calculated as the total number of instructions divided by the number of qubits. The rightmost plot in Figure 10 shows the final compilation cost after applying STOQ to generate an approximate compilation of the unitary corresponding to each random circuit. As might be expected, we observe that STOQ generates relatively high-fidelity approximations for shallow circuits, since such unitaries are known to be reachable with a fixed number of instructions. But as the circuit depth increases, the resulting unitaries begin to look more like random unitaries, and the final compilation cost approaches that of the randomly-generated five-qubit unitary discussed previously. Indeed, we expect a similar scaling with circuit depth for...
quantum circuits in general, regardless of whether they are randomly generated, since the size of the reachable state space grows exponentially with the depth of the circuit.

5. Discussion

We note that because the STOQ protocol requires calculating the product of the compiled sequence during each iteration, the computational cost of each iteration grows exponentially in the system size \( n \). Therefore, STOQ is unlikely to be useful for system sizes of more than around 10 qubits. In particular, for compilation of time-evolution unitaries, this clearly means that STOQ will be less computationally efficient when compared to compilation methods based on product formulas, which in general have a computational cost that depends only on the number of terms in the Hamiltonian and is independent of the system size.

We note that unitaries generated via time evolution of a Hamiltonian often benefit from the sparsity of the Hamiltonian. In general, an \( n \)-qubit Hamiltonian has \( 4^n \) coefficients when expressed in the basis of Pauli operators. For the five-qubit version of the Hamiltonian in Equation C2, only nine of these 1024 coefficients are non-zero. Sparsity in the Hamiltonian greatly limits the subspace of the full operator space that can be reached by via time evolution, which in turn makes compilation a more feasible task and allows techniques such as Suzuki-Trotter and QDRIFT to be highly efficient. Because the number of possible step proposals during each iteration of the STOQ search process is determined by the number of terms in the Hamiltonian, it is reasonable to infer that STOQ is similarly more effective when the problem structure contains such sparsity. This is further evidenced by the inability of the STOQ protocol to efficiently obtain low cost values when compiling sequences for random target unitaries, which are not sparse in general.

As demonstrated, STOQ has some features that are distinct from other methods. One notable feature is the capability of generating results with arbitrary gate sets, regardless of whether the gates are fixed or parameterized or whether the gate set is universal. In addition, repeated application of STOQ provides many independent approximate compilations of the same unitary, and each compilation creates a sequence that will cause the system state to traverse a different path in state space. As depicted in Figure 9, even stochastic techniques such as randomized Suzuki-Trotter or QDRIFT result in a compiled sequence that will cause the system state to follow very nearly the same path in state space as the deterministic version, whereas sequences generated by STOQ cause the system to traverse unique paths that can differ greatly from the ideal path and from each other.

Appendix D: Instruction set used in numerical and experimental demonstrations

In this appendix, we describe the instruction set used to generate the RAV and XEB sequences used for the demonstrations in Section III.

The sequences consist of layers built from a continuously-parameterized gate set \{\( R(\theta, \varphi) \), \( R_Z(\theta) \), \( MS(\theta, \varphi) \)\}, where these operations are defined in matrix form as follows:

\[
R(\theta, \varphi) = \begin{bmatrix} \cos \frac{\theta}{2} & -ie^{i\varphi} \sin \frac{\theta}{2} \\ ie^{-i\varphi} \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (D1)
\]

\[
R_Z(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \quad (D2)
\]

\[
MS(\theta, \varphi) = \begin{bmatrix} \cos \frac{\theta}{2} & 0 & 0 & -ie^{-i2\varphi} \sin \frac{\theta}{2} \\ 0 & \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} & 0 \\ 0 & -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} & 0 \\ -ie^{i2\varphi} \sin \frac{\theta}{2} & 0 & 0 & \cos \frac{\theta}{2} \end{bmatrix} \quad (D3)
\]

We chose this gate set because it aligns most directly with the native gate set of the QSCOUT trapped-ion processor. Specifically, \( R(\theta, \varphi) \) and \( MS(\theta, \varphi) \) are implemented by the QSCOUT device as native one-qubit and two-qubit physical
operations, and $R_Z(\theta)$ is implemented by the QSCOUT device as a virtual one-qubit operation which requires no physical interaction with the qubits.

In our implementation of RAV and XEB sequence generation, we used the Python representation of these parameterized operations provided by Sandia at https://gitlab.com/jaqal/qscout-gatemodels/-/blob/ebfe44a842485b1c170ad996690a150659f5c52a/src/qscout/v1/native_gates.py.

Each generated layer consists of three $R(\theta, \varphi)$ gates, three $R_Z(\theta)$ gates, and one $MS(\theta, \varphi)$ gate. The target qubit(s) for each gate are chosen uniformly at random from the set of all qubits in the system. A layer is constructed by first choosing random parameter values for each of these seven gates. Values for each $\theta$ rotation angle are chosen uniformly at random in the interval $[-\pi/10, \pi/10]$, and values for each $\varphi$ axis angle are chosen uniformly at random in the interval $[-\pi, \pi]$. After the parameter values are chosen, the order of the gates within the layer is then randomly permuted, which produces the layer that is ultimately used in the sequence.