

Benchmarking quantum computers with any quantum algorithm

Stefan K. Seritan,^{1,2} Aditya Dhumuntarao,¹ Aidan Q. Wilber-Gauthier,^{1,2} Kenneth M. Rudinger,¹
Antonio E. Russo,² Robin Blume-Kohout,¹ Andrew D. Baczewski,² and Timothy Proctor¹

¹Quantum Performance Laboratory, Sandia National Laboratories,
Albuquerque, NM 87185, USA and Livermore, CA 94550, USA

²Quantum Applications and Algorithms Collaboratory, Sandia National Laboratories,
Albuquerque, NM 87185, USA and Livermore, CA 94550, USA

Application-based benchmarks are increasingly used to quantify and compare quantum computers' performance. However, because contemporary quantum computers cannot run utility-scale computations, these benchmarks currently test this hardware's performance on "small" problem instances that are not necessarily representative of utility-scale problems. Furthermore, these benchmarks often employ methods that are unscalable, limiting their ability to track progress towards utility-scale applications. In this work, we present a method for creating scalable and efficient benchmarks from any quantum algorithm or application. Our *subcircuit volumetric benchmarking* (SVB) method runs subcircuits of varied shape that are "snipped out" from some target circuit, which could implement a utility-scale algorithm. SVB is scalable and it enables estimating a *capability coefficient* that concisely summarizes progress towards implementing the target circuit. We demonstrate SVB with experiments on IBM Q systems using a Hamiltonian block-encoding subroutine from quantum chemistry algorithms.

I. INTRODUCTION

Advances in quantum computing hardware since 2015 have spurred increasing interest in benchmarking the performance of quantum computing systems [1–8]. One approach is to run applications on the quantum computer, then measure and report the error in the results [1–3]. This methodology underpins many application-based quantum computer benchmarking suites introduced in recent years [3, 9–30]. But because contemporary quantum computers cannot yet solve practically useful problems, this approach has only enabled testing their ability to solve "small" problems using small quantum circuits (programs) that are far from the *teraquop* (one trillion quantum operations, or *quops*) regime expected to enable true quantum utility [1, 31–33]. It is unclear whether a quantum computer's performance on small problem instances, or on subroutines required to solve those small problems, can be used to assess technological progress towards the goal of quantum utility.

We introduce *subcircuit volumetric benchmarking* (SVB), a method for creating application-based benchmarks that do track progress towards solving a given computational problem, e.g. a utility-scale challenge problem. The SVB method, summarized in Fig. 1 and described in Sec. II, begins with one or more useful *target* circuits that solve a relevant problem. The target circuit[s] may be arbitrarily large. Subcircuits of various sizes and shapes are "snipped out" of the target circuit[s], and incorporated into experiments run on a specific quantum computer to measure its ability to run the subcircuits without error. The final output of an SVB experiment quantifies the largest subcircuits of the useful circuit[s] that the benchmarked quantum computer can reliably execute. This approach enables benchmarking a quantum computer's ability to solve a particular computational problem even when the computer is far from being able to successfully solve the problem, e.g. because its error rates are too high or it has too few qubits. SVB thus enables fine-grained tracking of *progress*

toward the capability to solve any specific challenge problem.

SVB can also be used to predict how well the full circuit[s] would run on the tested system or on a larger but otherwise similar system, by extrapolating the process fidelity of the full circuit's execution from direct measurements of the process fidelity of small subcircuits of varied sizes. We propose a simple formula for this extrapolation, based on "effective error rates", which we show to be accurate under strong assumptions and conjecture to be reliable in a wider range of practical scenarios. It generalizes a simple and popular heuristic for predicting circuit fidelity—sum up the error rates (process infidelities) for each individual gate in the circuit [5], as (usually) estimated by randomized benchmarking [2, 34–38]—but SVB extrapolation improves upon this simple heuristic by capturing commonly observed effects that the simple heuristic misses, including the effects of crosstalk [5, 39–42] and constructive or destructive interference of coherent (unitary) errors. SVB therefore provides an elegant link between application-based benchmarks and the prediction of application performance from low-level metrics and models derived from (e.g.) randomized benchmarking or tomography [2].

Application-based benchmarking is popular, in part, because it enables jointly testing an entire quantum computing "stack", including compilation algorithms [1–3, 8, 10, 43]. Until now, this kind of benchmarking was only possible for applications small enough to execute with reasonable fidelity on contemporary hardware [1–3], such as factoring 15 with Shor's algorithm. But an integrated quantum computer's performance on small, classically-tractable problems is not necessarily reflective of its performance on utility-scale problems [1, 44], because (e.g.) unscalable circuit compilation strategies can be applied. SVB enables full-stack benchmarking with utility-scale applications even on current systems, by first compiling a utility-scale application and then using SVB to measure a system's capability on snippets from that circuit.

SVB can be applied to any quantum algorithm. In this paper, we demonstrate it by creating benchmarks from

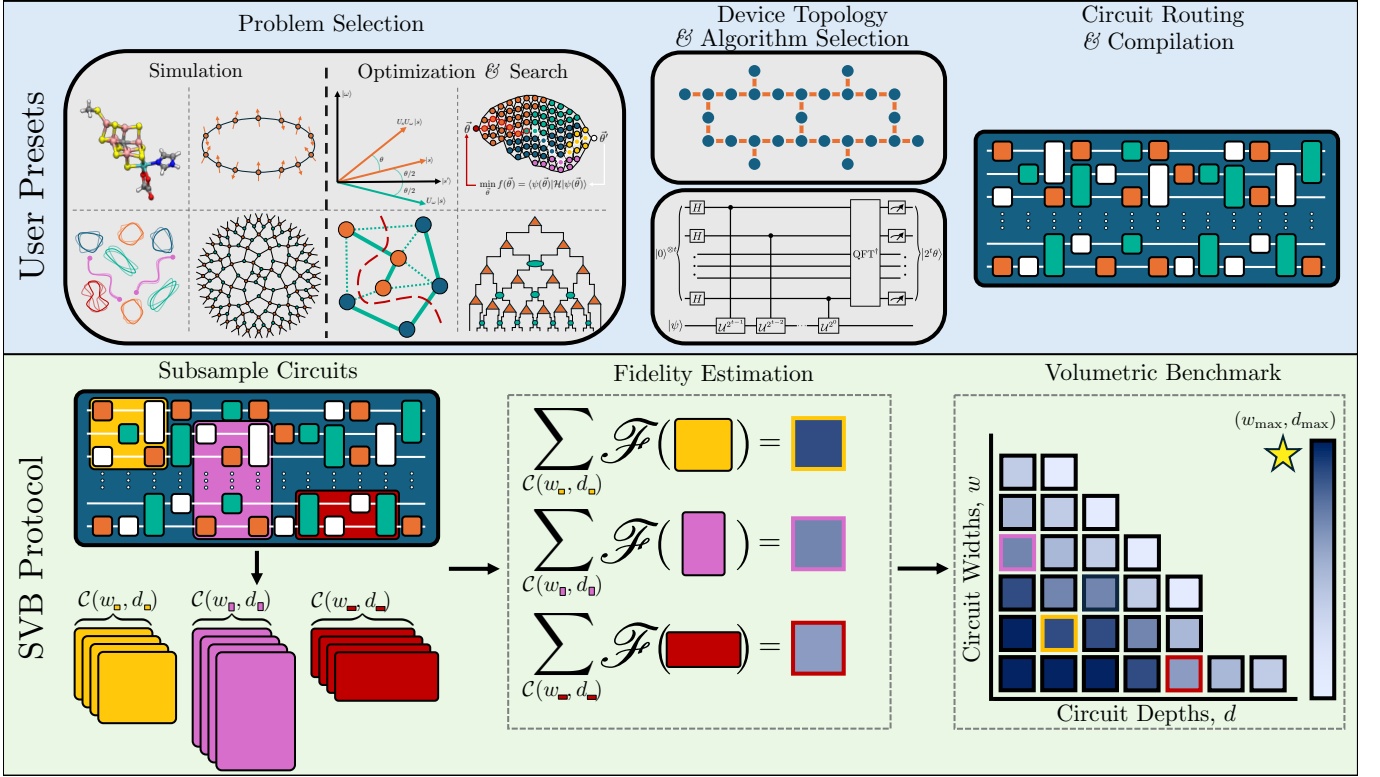


FIG. 1. **Subcircuit volumetric benchmarking.** A diagram of the subcircuit volumetric benchmarking (SVB) method that we introduce in this paper, which enables creating scalable and efficient benchmarks from quantum algorithms. The SVB method (green lower box) takes, as its input, a fully compiled *target circuit* c . Although any target circuit can be used, we envisage c implementing an interesting quantum algorithm, constructed by compiling an algorithm for an interesting computational problem (e.g., “factor RSA-2048” [1]) so that it can be run on a particular quantum computer. It may be too large to execute on contemporary quantum computers, containing millions or trillions of gates. SVB efficiently assesses how close a particular quantum computer is to being able to execute c with a high probability of success. It does so by “snipping out” subsamples of various shapes from c (lower left panel), executing efficient experiments that enable estimating the process fidelity with which those subsamples are executed (lower middle panel), and then summarizing performance in a volumetric plot (lower right panel) that shows how the subsamples’ average process fidelity decays with increasing circuit width (the number of qubits) and depth (the number of layers of gates). These volumetric plots provide a visual summary of how far a system is from successfully solving the problem of interest and can be used to quantify this “distance” in terms of a *capability coefficient*.

the Hamiltonian block-encoding subroutine [45–47] used in quantum chemistry applications, and running them on IBM Q systems (Sec. III). Although these demonstrations did not use utility-scale problems (the largest “application” circuits we used are actually a subroutine for solving a classically easy problem in quantum chemistry, and comprise about 10,000 circuit layers on 21 qubits), they show clearly how SVB can be used to (1) track progress towards implementing utility-scale quantum algorithms and quantify it by a simple *capability coefficient*, and (2) compare the performance of different algorithms or algorithm configurations that solve the same problem.

II. SUBCIRCUIT VOLUMETRIC BENCHMARKING

Subcircuit volumetric benchmarking (SVB) is illustrated schematically in Fig. 1. It is designed to estimate how reliably a particular quantum computer can, or could, run a given quantum algorithm. The input to SVB is a quantum circuit

c that is *fully compiled* for a quantum computer Q into gates intended to apply unitary operations. By “fully compiled”, we mean a circuit c that contains only gates native to Q , mapped to specific qubits in Q (see Appendix A for more details), as illustrated in the top row of Fig. 1. SVB can be applied in principle to any circuit, but we envisage that c will typically implement some interesting algorithm—perhaps a utility-scale algorithm requiring thousands of qubits and trillions of quantum gates (a.k.a. *quops*)—or a subroutine from such an algorithm. Because SVB is designed to estimate the error in Q ’s execution of c , it does not directly quantify other aspects of a “full stack” quantum computing system like compiler performance or gate speed. But SVB can be used as part of a broader methodology for assessing full-stack performance, including algorithmic performance, e.g., by applying SVB to circuits created with different algorithmic parameters or created with different compilation algorithms.

Given a circuit c for system Q , an SVB experiment proceeds in two steps:

1. For each circuit shape (w_i, d_i) in some set of user-chosen circuit shapes, sample K subcircuits or “snippets” $c_{w_i, d_i, j}$ with that shape from c .
2. Experimentally estimate a metric for the quality with which each sampled subcircuit $c_{w_i, d_i, j}$ can be executed on Q using an efficient estimation protocol.

Any procedure of this sort is an SVB experiment, but to design a specific SVB experiment it is necessary to choose a circuit snipping procedure, a quality metric, and a method for estimating that quality metric. In Section II B, we explain precisely how we sample circuit snippets, and in Section II C we explain the success metric we use—process fidelity—why we use that metric, and how we efficiently estimate it.

SVB data can be directly presented on a volumetric plot [5, 6] as illustrated in the lower right of Fig. 1 and demonstrated with experimental data in Figs. 2 and 3. SVB volumetric plots show (i) how the estimated quality metric (in our experiments, process fidelity) of circuit snippets depends on their shape (w, d) , and (ii) the shape of the full or “target” circuit c (the yellow star in Fig. 1 and other figures). They illustrate both quantitatively and qualitatively how far the benchmarked quantum computer is from being able to run c with a high probability of success. However, SVB data can be analyzed in greater detail to extract (e.g.) predictions for c ’s fidelity and a capability coefficient, which we discuss later. Before doing so, we contrast SVB with existing approaches to application benchmarking.

A. Features and advantages of SVB

SVB is designed to overcome certain technical challenges to quantum computer benchmarking: efficiency, scalability, and the current unavailability of utility-scale quantum computers [1]. These challenges are apparent in the simplest extant algorithm for estimating the error in a quantum computer’s execution of a circuit c , which is to run c many times on that quantum computer and compare the results to what *would have* resulted from running c on an ideal, error-free quantum computer. This approach is commonly used in existing quantum computer benchmarks [1–3, 9–30]. To implement it, (i) a classical computer is used to compute or estimate the probability distribution over bit strings (P_{ideal}) from which c would sample in the absence of errors, (ii) the circuit c is executed many times on the quantum computer to estimate the distribution (P_{actual}) that is actually being sampled from, and (iii) the error in the execution in c is quantified by computing a discrepancy metric S between the *estimated* P_{ideal} and P_{actual} . An example of such a metric that is commonly used in benchmarking is the classical (Hellinger) fidelity [1–3].

Unfortunately, that simple approach is generally intractable [1, 2]. In general, the computational effort required to accurately estimate the quality metric S grows superpolynomially with c ’s width w (the number of qubits). Either or both of the quantum resources (number of circuit executions) or the classical resources (the amount of classical computing) required may blow up. Computing P_{ideal} generally requires classical

computations whose time and/or space requirements grow exponentially with c ’s width. Similarly, a reasonable-precision estimate of the success metric S (via estimating P_{actual}) generally requires a very large number of executions of the circuit c .

Important exceptions exist. For example, a success metric can be directly estimated without first computing P_{ideal} if c solves a problem for which a candidate solution can be efficiently classified as correct or incorrect (such as factoring). But for general circuits c , the simple approach remains intractable. Even when solutions can be efficiently classified, distinguishing very small success probabilities (S) from zero demands very many circuit executions. SVB avoids these problems; it neither computes P_{ideal} nor estimates P_{actual} nor even requires running the target circuit c .

Another problem with simply running the target circuit c is that it requires having access to a quantum computer Q large enough to run c . One of the main reasons to benchmark contemporary quantum computers is to assess technological progress towards running “useful” utility-scale quantum algorithms [1] (e.g., factoring RSA-2048), and these algorithms currently demand more qubits than are available in contemporary quantum computers. Today’s quantum computers can be accurately regarded as *prototypes* for future utility-scale quantum computers. When a user does not have Q , but does have a prototype Q' that is similar but smaller (provides fewer qubits), then existing algorithm- and application-based benchmarks are limited to running smaller algorithmic circuits. This only enables, at best, *indirect* assessment of how far the prototype Q' is from successfully running a utility-scale circuit c . SVB addresses this question directly by running “snippets” from the target circuit c on the available prototype Q' .

B. Circuit Snipping

The central idea of SVB is to run circuits of various shapes that are snipped out from our target circuit c , which could be very large. There is, however, no unique way to do this and no natural *and* precise set of criteria for judging between two different snipping algorithms—the best choice will depend on what is to be done with the SVB data (e.g., predicting the full circuit’s fidelity). The primary factor that complicates how to do circuit snipping is multi-qubit gates: if we want to select a width- w subcircuit from within a width- w_c circuit c and we randomly select w qubits from c to retain, there may be many gates between those qubits and qubits outside that set. If $w \ll w_c$ it will typically be the vast majority of those multi-qubit gates. For systems with a connectivity graph that is low-degree and local, there is a simple approach to solving this problem: pick *connected* subsets of qubits, as shown in the schematic in the lower left of Fig. 1 (which shows a circuit for a linear chain of qubits), with the constructed circuit snippet then dropping any multi-qubit gates between those qubits and qubits outside that set. This is the approach we take, and below we describe it more precisely. Other approaches may be necessary for systems with high connectivity, and exploring snipping algorithms for that setting is left for future work.

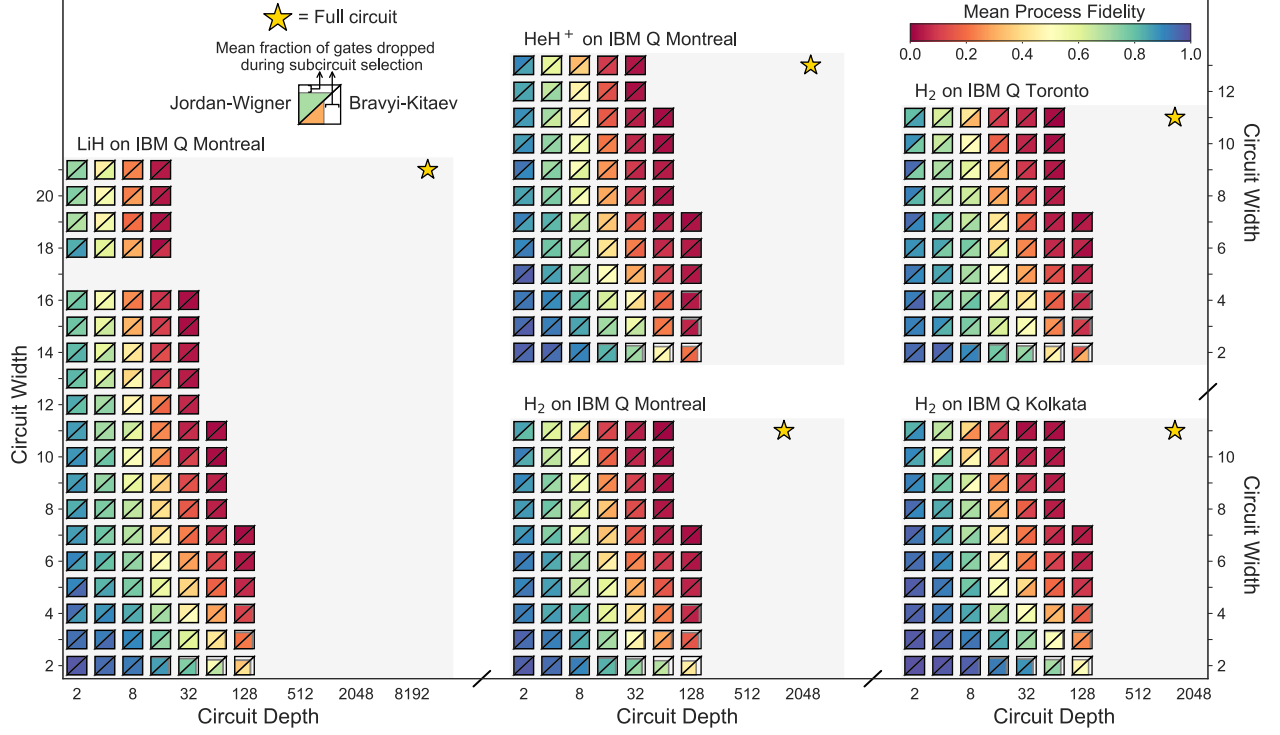


FIG. 2. **Benchmarking IBM Q system’s performance on Hamiltonian simulation algorithms using SVB.** Volumetric benchmarks obtained by applying SVB to benchmark three IBM Q systems’ performance on ground-state energy problems for three different molecules (LiH, HeH⁺ and H₂) using two versions of a block-encoded Hamiltonian simulation algorithm. This data was collected in 2022. Each plot shows the measured performance of one system (IBM Q Montreal, IBM Q Toronto, or IBM Kolkata) on a ground-state energy algorithm for one molecule, using two different versions of the algorithm: a Jordan-Wigner (upper triangles) and Bravyi-Kitaev (lower triangles) encoding of the molecular orbitals into the qubits. In each case, we indicate the shape of the target circuit with a star in the upper right. In all cases, we observe little difference between the performance of the two versions of the algorithm. In all cases we also see that these systems were far from being capable of running the target circuit with low error (note the logarithmic scale on the horizontal axis).

Since SVB is intended to measure performance on circuits that may be too big to run on available hardware, we must consider adapting a circuit c that was compiled for a hypothetical quantum computer Q with n qubits to an available smaller prototype Q' that contains $n' < n$ qubits. We say that an n' -qubit subset of Q is *equivalent* to Q' if Q' offers the same operations and connectivity graph as that subset, up to relabeling of qubits. To implement SVB on a prototype Q' for Q , there must be at least one subset of Q that is equivalent to Q' and ideally there are many such subsets. For example, if Q is a 100×100 square grid of qubits and Q' is a 10×10 square grid of qubits, then there are 8281 different 10×10 grids of qubits embedded in Q —each of which is equivalent to Q' .

The input to our circuit snipping algorithm is a circuit c compiled for Q , with width w_c (the number of qubits) and depth d_c (the number of layers), a quantum computer Q' with n' qubits on which we will execute our snippets, and a circuit shape (w, d) for our snippets with $w \leq n'$, $w \leq w_c$, and $d \leq d_c$. We assume that c acts on all of the qubits in Q to simplify the description. With these inputs, we select a circuit snippet as follows:

1. Uniformly sample a starting circuit layer from the first $d_c - d + 1$ layers in c , and create a depth d circuit c' from

this and the following $d - 1$ layers in c .

2. Randomly sample an n' -qubit subset of Q , denoted q_n , that is equivalent to Q' .
3. Randomly choose a set of w qubits (q_w) from q_n and which are connected in Q [48], and then create circuit $c_{w,d}$ of shape (w, d) from c' by discarding all the qubits in c' except those in q_w along with any multi-qubit gates between a qubit in q_w and qubits outside of q_w [49].

The random sampling in steps 2 and 3 is not necessarily *uniform*, because uniform sampling over all valid qubit subsets may be both difficult (when $w \gg 1$) and suboptimal. For example, sampling qubit subsets uniformly does not generally give each qubit equal probability of being included in a shape (w, d) subcircuit. We use the following simple approach: to select q_w we first select a single qubit from c uniformly at random, and then we grow our set of selected qubits by randomly selecting a neighbour of our qubits to add to the set, until we have selected w qubits.

Our snippet-sampling algorithm drops multi-qubit gates between qubit sets that cross the boundary of the selected subset. We tested the size of this effect—i.e., the fraction of the gates discarded by the algorithm—and found it to be small

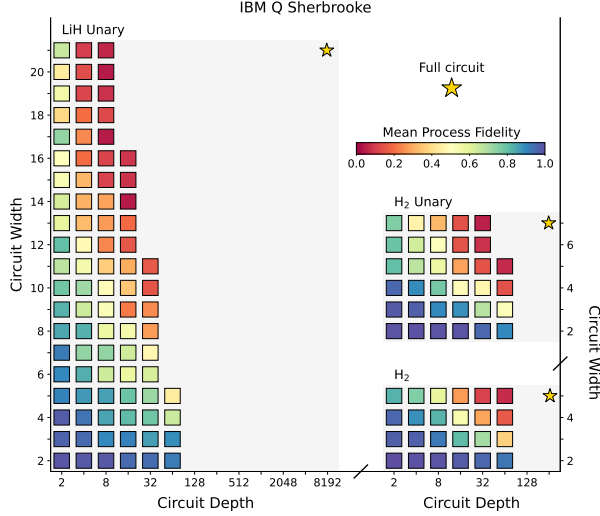


FIG. 3. **Benchmarking IBM Q Sherbrooke's performance on ground-state energy problems using SVB.** Volumetric benchmarks obtained by applying SVB to benchmark IBM Q Sherbrooke's performance on algorithms for ground-state energy problems: a block-encoded Hamiltonian simulation algorithm for LiH using a unary encoding, and for H₂ with and without a unary encoding. This data was collected in 2024. As in the IBM Q results from 2022 (Fig. 2), IBM Q Sherbrooke is not capable of executing any of the target circuits with low error.

in the circuits used for this paper, except when $w = 2$. This is shown in Fig. 2, where the size of each triangle (the data points) shows the fraction of gates that were dropped.

C. Estimating subcircuit performance

SVB estimates how some quality metric varies with circuit snippet shape, and so it is necessary to specify a quality metric and a method for estimating it. In this paper, we use *process fidelity* [2], because it can be efficiently estimated and plausibly extrapolated to larger circuits. In this subsection, we motivate the choice of process fidelity and explain how to estimate and extrapolate it.

1. Definition and properties of process fidelity

The process fidelity F between an n -qubit unitary U and a noisy implementation described by a transfer matrix Λ is given by [2]

$$F = \langle \Psi | (\mathbb{I} \otimes \mathcal{E}) [|\Psi\rangle\langle\Psi|] | \Psi \rangle, \quad (1)$$

where $\mathcal{E} = \Lambda \mathcal{U}^\dagger$ is the circuit's error map (i.e., ideally $\mathcal{E} = \mathbb{I}$), $\mathcal{U}[\rho] = U\rho U^\dagger$ is the superoperator representation of the ideal unitary, \mathbb{I} is the n -qubit identity superoperator, and $|\Psi\rangle$ is any pure state that is maximally entangled between the two 2^n -dimensional Hilbert spaces on which $\mathbb{I} \otimes \mathcal{E}$ acts. Because process fidelity is a property of a dynamical map, it does not

depend on or account for state preparation and measurement (SPAM) errors.

Process fidelity quantifies how accurately the quantum dynamical transformation Λ (implemented by running c on a real-world processor) simulates the ideal evolution U , if/when the qubits getting transformed are (1) initially entangled with other qubits and (2) not going to be measured immediately [2]. This is precisely the scenario that applies in SVB, because the snippets whose fidelity are being measured are subroutines of a larger target circuit.

2. Predicting a circuit's process fidelity from subcircuit fidelities

Using the process fidelity as a quality metric in SVB enables us to extract *effective error rates* from SVB data, and then make principled predictions for the full circuit's process fidelity. We propose an extrapolation formula that predicts the full circuit's process fidelity, and we use it to define a *capability coefficient* quantifying what fraction of the full circuit the benchmarked device can execute, as well as a *scalability coefficient* that quantifies whether the device's performance degrades on larger circuits (and if it does, how much).

Extrapolation is based on a simple principle: process fidelities are *approximately multiplicative*. Consider a circuit c of shape (w_c, d_c) with a simple error model: local Pauli stochastic errors with a rate of ϵ and process fidelity $F = 1 - \epsilon$. For this error model, the fidelity of the circuit is given by [50, 51]

$$F_c \approx (1 - \epsilon)^{w_c d_c} = F^{w_c d_c}, \quad (2)$$

to within, typically, only a small correction [52]. More generally, if each location (i, j) in the circuit (i.e., the i th qubit at layer j) has local Pauli stochastic errors with error rate ϵ_{ij} then

$$F_c \approx \prod_{ij} F_{ij}, \quad (3)$$

where $F_{ij} = 1 - \epsilon_{ij}$. This is the widely-used formula for approximating a circuit's fidelity by multiplying together the fidelities of its constituent gates, shown schematically in the top left of Fig. 4. It is often approximated further, using a first-order expansion, as $F_c = 1 - \sum_{ij} \epsilon_{ij}$.

For general errors, a circuit's fidelity is not well-approximated by Eq. (3). Coherent errors on different gates can add or cancel, crosstalk errors can mean that a gate does not have a single, well-defined fidelity—a gate's errors can depend on the context of which other gates are being applied and may be correlated—and non-Markovian errors can complicate matters even further. This has been widely observed in experiments (e.g., see [5]) and is one of the motivations for using SVB (or other holistic benchmarks) instead of simply measuring gate fidelities and using them to predict a circuit's fidelities. We can, however, generalize the ideas behind Eq. (3) to circuit snippets of arbitrary shape.

We can approximate F_c by splitting c into disjoint regions of any size, as shown in the diagram in Figure 4. That is, the circuit is split into disjoint blocks each of shape $w \times d$ (or disjoint mixed-shape blocks) with $w \leq w_c$ qubits and $d \leq d_c$

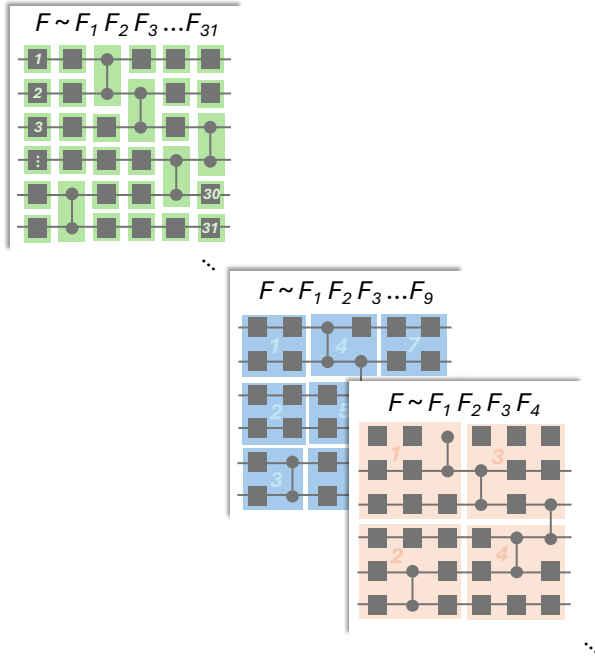


FIG. 4. **Estimating circuit fidelities from subcircuit fidelities.** The key idea underpinning SVB is that a large circuit’s fidelity (F) can be estimated by estimating the fidelities of subcircuits from that circuit, then multiplying them together. We illustrate the idea by subdividing a circuit of shape $(6,6)$ into disjoint subcircuits of shape $(2,2)$ [middle] and $(3,3)$ [lower right], and estimating F by the product of those subcircuits’ fidelities (F_i). The fidelities of larger subcircuits can capture complex effects such as crosstalk and coherent addition or cancellation of error, which would not be captured by simply multiplying together the fidelities of all gates in the circuit (illustrated by the division of the same circuit into blocks of shape $(1,1)$ and $(2,1)$ in the top left). Two-qubit gates that connect different subcircuits cause practical and conceptual problems, as discussed in the main text. In SVB we randomly sample subcircuits of a given shape, rather than picking a disjoint subdivision of the circuit and exhaustively estimating the fidelities of all those circuits, as shown here.

layers, generalizing the 1×1 and 2×1 blocks used in Eq. (3) (shown in the top left of Fig. 4). Letting $F_{w,d}^{(i)}$ be the fidelity of the i^{th} block, and letting $N_{w,d}$ denote the total number of blocks, then

$$F_c \approx \prod_{i=1}^{N_{w,d}} F_{w,d}^{(i)}. \quad (4)$$

As the size of the blocks grows, their error processes are, in typically circuits, likely to combine multiplicatively, as though they are stochastic. This follows from three general observations: (1) except in highly structured cases, the coherent errors in each large block’s error process are unlikely to be close to maximally adding or cancelling between blocks (which requires those coherent errors to be commuting), meaning that the error processes approximately combine as though they are stochastic; (2) a growing fraction of crosstalk errors between nearby qubits will act *within* each block rather than across blocks; and (3) a growing fraction

of non-Markovian (time-correlated) errors will similarly act within each block and thus be captured by each block’s process fidelity. We therefore conjecture that as $w \rightarrow w_c$ and $d \rightarrow d_c$, the multiplicative approximation will become increasingly accurate. (Note that when $(w,d) = (w_c,d_c)$ the equality holds trivially). If this is true, we can estimate F_c by dividing it into smaller blocks and estimating each block’s process fidelity. This conjecture is foundational to SVB.

3. Predicting the target circuit’s process fidelity with SVB data

SVB does not actually divide c into disjoint blocks and measure each one’s fidelity. For utility-scale circuits, the number of blocks would be prohibitive large. Instead, it samples a relatively small number K of snippets with each shape (w,d) , for some range of shapes, and estimates each snippet’s process fidelity $F_{w,d,k}$ for $k = 1, \dots, K$. We then use the geometric mean of those snippets’ observed process fidelity, in lieu of Eq. (4), to estimate $\prod_{i=1}^{N_{w,d}} F_{w,d}^{(i)}$ and thus F_c . Specifically, we estimate F_c as

$$\widehat{F}_c = \text{GM}[F_{w,d}]^{\frac{w_c d_c}{wd}} \quad (5)$$

where

$$\text{GM}[F_{w,d}] \equiv (F_{w,d,1} F_{w,d,2} \cdots F_{w,d,K})^{\frac{1}{K}}. \quad (6)$$

Using the geometric mean fidelity of the snippets with shape (w,d) , we define an *effective error per quop* in the context of c :

$$\epsilon_{w,d} = 1 - \text{GM}[F_{w,d}]^{\frac{1}{wd}}. \quad (7)$$

The estimate or prediction for F_c derived from the shape (w,d) snippets is thus

$$\widehat{F}_c = (1 - \epsilon_{w,d})^{w_c d_c}. \quad (8)$$

If the benchmarked system’s errors are very well-behaved (e.g., local and gate-independent depolarizing errors), then the effective error rate will be nearly independent of snippet shape—there will be some ϵ for which $\epsilon_{w,d} \approx \epsilon$ for all (w,d) —and every snippet of the same shape will have approximately the same fidelity. Such a system has a single, well-defined error per quop ϵ in the context of circuit c , which can be easily extracted from SVB experiments. Generally, however, we expect the observed $\epsilon_{w,d}$ to vary with (w,d) , e.g. because of crosstalk and/or coherent errors. SVB data can be used to probe (and perhaps understand) this dependence.

4. Estimating process fidelity

The SVB protocol is *scalable* only if each step in the protocol can be performed with only polynomial resources even when $w \gg 1$. Several methods for estimating a circuit snippet’s process fidelity are both scalable and reliable (“system robust” [1]), each with different regimes of applicability

[2, 50, 51, 53]. We chose to use *mirror circuit fidelity estimation* (MCFE) [50], which has broad applicability. MCFE estimates a circuit c 's process fidelity by running circuits sampled from three ensembles of *mirror circuits* constructed from c . For the experiments in this paper, each circuit snippet was mapped to many different circuits (typically between a few hundred and a few thousand) that were run to estimate that snippet's fidelity. See Refs. [2, 50] for further details on MCFE.

Any experimental procedure for estimating a circuit's fidelity will result in uncertainty ("error bars") in its estimate. Estimating the effective error per quop (Eq. (7)) requires estimating the geometric mean of K circuits of the same shape (w, d), and when the snippet's fidelities are close to zero the uncertainty on this estimate will be particularly large. In our analysis of experimental data below, we do not compute an effective error per quop at any circuit shape where we cannot clearly distinguish one or more of the estimated fidelities from zero (we use a somewhat arbitrary criteria of the fidelity estimate being below 7%, to account for both statistical uncertainty and systematic errors in MCFE estimates).

III. DEMONSTRATION

We now use SVB to create a scalable benchmark for a subroutine relevant to quantum chemistry calculations, and use it to benchmark IBM Q devices.

A. The benchmarked Hamiltonian simulation subroutine: LCU circuits

One prospective application of quantum computers is calculating the ground-state or low-lying excitations of molecular or solid-state systems [54, 55]. A particular approach to this application involves sampling from the eigenspectrum of first-principles electronic structure Hamiltonians using quantum phase estimation [56]. This requires preparing specific encoded eigenstates of those Hamiltonians [57, 58] and applying unitaries that encode their eigenspectra to those states [59, 60]. Block encoding is a common subroutine that might be used for either purpose [58, 59]. It is ubiquitous in contemporary quantum algorithms [47, 61]. We demonstrate SVB for target circuits that implement a single application of the block encoding of a second-quantized electronic structure Hamiltonian, for different molecules. While this is an algorithmic primitive (subroutine) rather than a full algorithm, it still yields target circuits too large to fit on present-day quantum computers.

We do not expect complete applications that use these block encodings to be feasible without fault-tolerant quantum computation (FTQC) facilitated by quantum error correction (QEC) [62]. Even single applications of block encodings for minimal molecular Hamiltonians may not be feasible without QEC. Nevertheless, because these subroutines are key components of some of the most efficient quantum simulation algorithms [63, 64], it is compelling to track progress towards

their implementation as quantum hardware continues to mature. We leave the adaptation of SVB to benchmarking *logical* circuits for future work.

The block encodings that we analyze with SVB are implemented using the *linear combinations of unitaries* (LCU) method [65]. We consider second-quantized Hamiltonians for three molecules: molecular hydrogen (H_2), the helium hydride cation (HeH^+), and lithium hydride (LiH). These Hamiltonians are discretized using small Gaussian basis sets: the STO-3G basis for H_2 and HeH^+ , and the STO-6G basis with an active space of 4 orbitals for LiH [66, 67]. The choice of molecules and basis sets means that all of our circuits use 21 or fewer qubits, although we note that SVB can be applied to circuits of any width. We also considered two different fermion-to-qubit mappings for encoding these Hamiltonians: Bravyi-Kitaev [68] and Jordan-Wigner [69]. Further implementation details of the requisite LCU oracles are given in Appendix B.

B. LCU benchmarking experiments on IBM Q

We ran SVB experiments using LCU circuits on various IBM Q systems, choosing a variety of combinations of the algorithmic parameters (see above) that define an LCU circuit instance. The precise details of how a high-level description of an LCU subroutine circuit was generated, optimized, compiled for a specific IBM Q system (creating our target c), and then turned into an SVB experiment, can be found in Appendix B.

We conducted two separate sets of experiments, one in 2022 and the other in 2024. In our 2022 experiments, we ran SVBs for the three different molecules considered herein (H_2 , HeH^+ and LiH) on the same system (IBM Q Montreal), and we ran an H_2 SVB experiment with identical algorithmic parameters on two other IBM Q systems (IBM Q Toronto and IBM Q Kolkata). In each case, we ran an experiment with the Bravyi-Kitaev encoding and one with the Jordan-Wigner encoding. In our 2024 experiments, we ran SVBs for H_2 and LiH on a single system (IBM Q Sherbrooke). In this experiment, we studied a smaller LCU circuit for H_2 by using a tapered Bravyi-Kitaev encoding [70]. We also considered an alternative implementation of that LCU circuit using unary iteration [59] (see Appendix B).

C. Summary volumetric plots

Figures 2 and 3 summarize the results of these two experiments using volumetric plots. Each panel in each figure shows the results for a single device and LCU instance (labelled by the molecule and, in some cases, additional algorithmic parameters). In each panel, we show the shape of the target circuit (yellow stars)—i.e., the fully-compiled LCU circuit implementing the LCU unitary for that molecule. The data points (squares) show how the average process fidelities of the circuit snippets executed vary with their shape. In all cases,

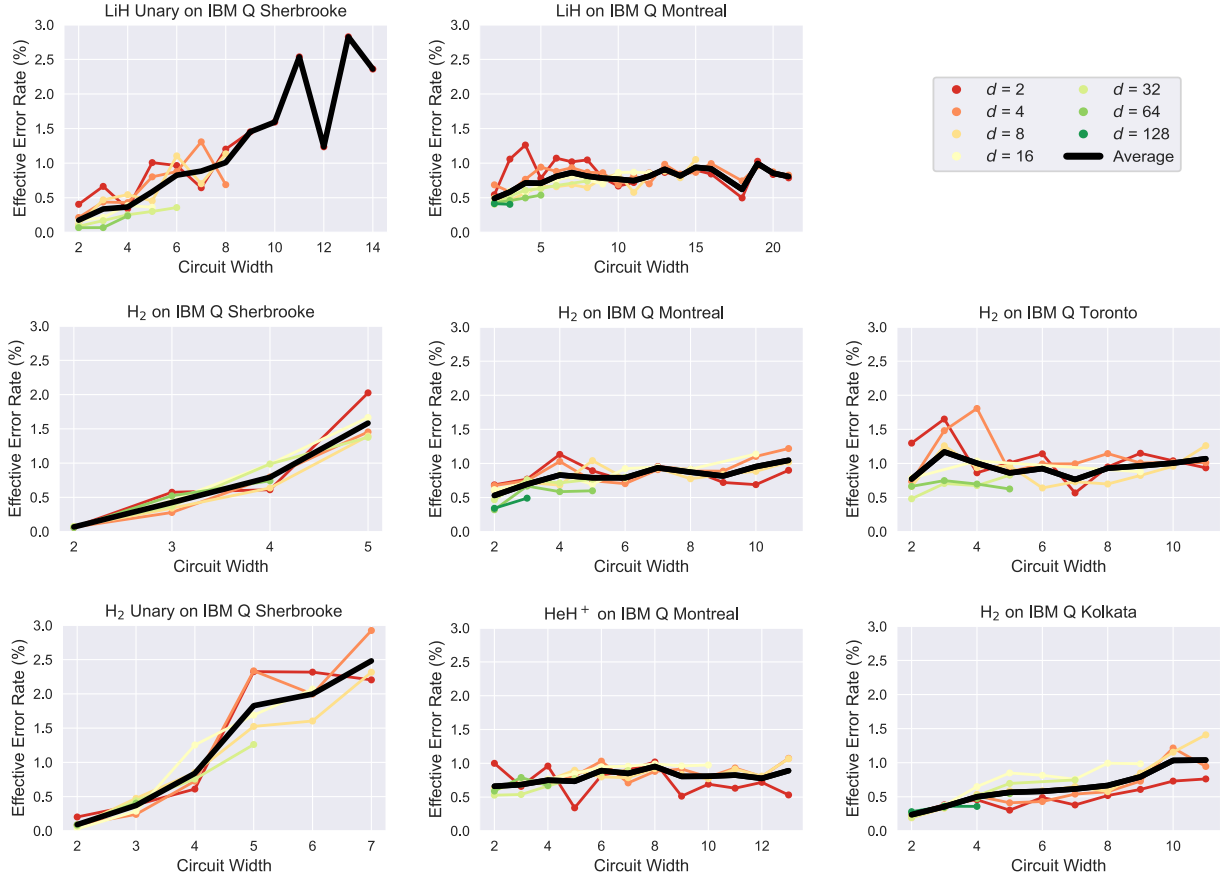


FIG. 5. **Effective error rates for IBM Q systems estimated from SVB.** The effective error rates observed in each SVB experiment (each panel) as a function of the circuit snippet’s width and depth. In an SVB experiment we measure the process fidelities of circuit snippets of various shapes (w, d) , and for each (w, d) we can turn those measured process fidelities into an effective error rate per quop $\epsilon_{w,d}$. We show $\epsilon_{w,d}$ as a function of width for each d (colored points and lines) and the average over d (ϵ_w , black lines). We observe that ϵ_w grows with w , i.e., the effective error rate increases in circuits on more qubits, but its growth varies substantially between systems. In particular, although IBM Q Sherbrooke (left column) has a substantially lower effective error rate in the low-width ($w = 2$) circuits than the other three systems (other columns), its effective error rate grows rapidly with width.

we find that the benchmarked systems are far from successfully implementing the target circuits—even for the example with smallest target circuit, corresponding to the H_2 molecule using a tapered Bravyi-Kitaev encoding (labelled as H_2 Tapered). These plots provide a qualitative summary of how far each system is from implementing the corresponding LCU circuit. We show how to quantify this “distance” to success in Section III D.

SVB enables experimental exploration of the impact of algorithmic parameters on circuit fidelities. Our first experiments were designed to explore one such parameter, the choice between Bravyi-Kitaev and Jordan-Wigner encodings. Figure 2 shows results for LCU circuits with a Bravyi-Kitaev encoding (lower triangles), and with a Jordan-Wigner encoding (upper triangles). For all systems, we observe only small differences between the process fidelities of the snippets for the Bravyi-Kitaev and Jordan-Wigner encodings. This is not surprising, but was also not a given; the target circuits defined by the two encodings have structural differences. Because this analysis showed negligible differences between the

two encodings, we combined the data from *both* encodings for the subsequent analysis. This initial comparison demonstrates a general method for using SVB to compare a system’s performance on different algorithms (or variants of the same algorithm) for the same problem.

D. Summarizing performance with effective quops

We now show how to use SVB results to concisely quantify how close a system is to implementing an algorithm, demonstrated with these example SVB experiments. At each snippet shape (w, d) we compute the effective error per quop ($\epsilon_{w,d}$) using Eq. (7). These error rates enable us to predict the target circuit fidelity and quantify how far the benchmarked device is from being able to execute the target circuit, as a percentage of that circuit’s quops.

Figure 5 shows how $\epsilon_{w,d}$ varies with snippet width (w) for each experiment, showing both the average over depths (ϵ_w , black lines) and $\epsilon_{w,d}$ for each depth d (colored lines). The

Algorithm	System	Width	Depth	F_0	F
H ₂	Toronto	11	1277	10 ⁻⁴⁷	10 ⁻⁶⁶
H ₂	Kolkata	11	1277	10 ⁻¹⁴	10 ⁻⁶⁴
H ₂	Montreal	11	1277	10 ⁻³²	10 ⁻⁶⁴
HeH ⁺	Montreal	13	2480	10 ⁻⁹³	10 ⁻¹²⁵
LiH	Montreal	21	12090	10 ⁻⁵⁴²	10 ⁻⁸⁹⁴
LiH	Sherbrooke	21	7979	10 ⁻¹²⁷	10 ⁻¹⁷³⁸
H ₂ Tapered	Sherbrooke	5	263	10 ^{-0.4}	10 ⁻⁹
H ₂ Unary	Sherbrooke	7	252	10 ^{-0.7}	10 ⁻¹⁹

TABLE I. **Predicted target circuit fidelities.** For each algorithm and system that we benchmarked, we use the measured fidelities of circuit snippets to predict the fidelity with which that system can run the target algorithmic circuit c . When we use the fidelities of the narrowest circuits ($w = 2$) to predict c 's fidelity we obtain the “optimistic” estimates F_0 , which we expect will typically be similar to predictions obtained from measuring one- and two-qubit gate error rates with RB. When we instead use the measured fidelities of wide snippets (in all cases here, as wide as the target circuit) we obtain fidelity predictions F that we conjecture will be much more predictive of c 's process fidelity. In all cases, we see that F is many orders of magnitude smaller than F_0 , and in the cases of H₂ on Sherbrooke we observe $F_0 > 0.1$ whereas F is so small that it would require an infeasible number of circuit executions to obtain useable data.

dependence of the effective error per quop on width reveals both spatial variation in qubit error rates and crosstalk errors in that system. In all cases, we find that ϵ_w grows with w , but its growth varies substantially from system to system. It increases by an order of magnitude with all three SVBs run on IBM Q Sherbrooke, from around 0.2% at $w = 2$ to between around 1.5% and around 2% at w_{\max} (w_{\max} denotes the largest width, which varies between algorithms, and is, in our experiments, equal to the target circuit's width). This dependence of ϵ_w on w shows that context-independent gate error rates (as, e.g., measured using RB) will not be able to accurately predict the target circuit's fidelity F_c . We conjecture that SVB will enable more accurate predictions of F_c by measuring the error per quop in a context that closely mimics execution of the target circuit.

To quantify the effect of the context-dependent error rates, we predicted F_c from the observed effective error rates at the lowest width ($w = 2$) and at the largest width ($w = w_{\max}$). We denote these predictions by F_0 and F respectively. They are shown in Table I for all our experiments. In all cases, F is smaller than F_0 by orders of magnitude (the smallest discrepancy is approximately eight orders of magnitude). For the smallest target circuit (H₂ Tapered) $F_0 = 0.40 \approx 10^{-0.4}$ but $F = 10^{-9}$. The LCU circuit is a subroutine in useful quantum algorithms, where it is typically used many times, so even a fidelity of 0.40 is far too small to obtain reasonable fidelities in a Hamiltonian simulation algorithm. However, an $O(1)$ fidelity for a complete algorithmic circuit might be sufficient to obtain utility, e.g., with the application of error mitigation [71]. In contrast, a fidelity of $O(10^{-9})$ is unlikely to be con-

sist with obtaining utility. This highlights the importance of realistic estimates of a target circuit's fidelity, as enabled by SVB.

To quantify how far a system is from successfully running the target circuits, we can express our effective error per quops as an *effective quops*,

$$Q_{w,d} = 1/\epsilon_{w,d}. \quad (9)$$

This is the maximum number of operations that can be implemented while achieving a fidelity of $O(1)$. The effective quops could depend strongly on both snippet width and depth, but we conjecture that *in practice* $\epsilon_{w,d}$ will asymptote to a constant value ϵ_c for sufficiently large w and d (see Section II C)—and that “sufficiently large” widths and depths will be much smaller than the target circuit's width and depth, except for target circuits where the width or depth is small. If this conjecture holds, a single *effective quops in context* can be defined for a target circuit c as

$$Q_c = 1/\epsilon_c. \quad (10)$$

Our experiments did not probe sufficiently large w and d to test this conjecture, but we can examine how $Q_{w,d}$ varied with shape in our experiments. We observed relatively little dependence on d (see Fig. 5, noting that we do not compute $\epsilon_{w,d}$ when we cannot distinguish one or more fidelities from zero), but there is significant dependence on w . We can take advantage of the relatively small variance with respect to d to define $Q_w = 1/\epsilon_w$, where ϵ_w is the average of $\epsilon_{w,d}$ over all depths d (black lines in Fig. 5). Since Q_w depends strongly on w in most of the experiments we performed, we define each system's *observed capability* as $Q_C = 1/\epsilon_{w_{\max}}$.

It is instructive to compare each system's observed capability to the capability that would be predicted from running only circuits of width $w = 2$, which we call *predicted capability*: $Q_0 = 1/\epsilon_2$. We expect Q_0 to be similar to what RB error rates would predict, except that Q_0 is more context-specific because it is extracted from narrow snippets of the target circuit instead of random Clifford circuits. Table II tabulates the observed and predicted capabilities for each experiment. Observed capabilities range from $Q_C \sim 40$ up to $Q_C \sim 120$. In almost all cases they are much smaller than the corresponding predicted capability. We quantify this gap with the *scalability coefficient*: Q_C/Q_0 , shown in Table I (as a percentage). The scalability coefficient ranges from 74% down to 4%.

Finally, we use our results to quantify how close these systems are to accurately executing the target circuits. Each target circuit requires a certain number of quops, $Q_T = w_c d_c$ [72]. Since we have measured the number of quops that each system can execute (in context), we can concisely summarize how close each system is to executing the target circuit with the ratio Q_C/Q_T , which we call the *capability coefficient*. These systems achieve between 0.03% and 5% of the required quops needed to implement these LCU circuits. The capability coefficient provides a simple way to track a sequence of prototypes' performance on an algorithm even when that algorithm can only be run with vanishing chance of success. The smallest capability coefficient we observed (0.03%) corresponds to

Algorithm	System	Algorithm Size (quops, Q_T)	Predicted Capability (quops, Q_0)	Observed Capability (quops, Q_C)	Scalability Coefficient (Q_C/Q_0 , %)	Capability Coefficient (Q_C/Q_T , %)
H ₂	Toronto	14000	130	94	72%	0.7%
H ₂	Kolkata	14000	420	96	23%	0.7%
H ₂	Montreal	14000	190	96	50%	0.7%
HeH ⁺	Montreal	32000	150	112	74%	0.3%
LiH	Montreal	254000	203	124	60%	0.05%
LiH Unary	Sherbrooke	169000	570	42	7%	0.03%
H ₂ Tapered	Sherbrooke	1300	1480	63	4%	5%
H ₂ Tapered Unary	Sherbrooke	1800	1100	40	4%	2%

TABLE II. **Summarizing system performance with observed quops.** The observed capability (Q_C) in quops, of each system on each LCU instance, from which we can concisely summarize how close the system is to successfully running the target circuit by computing the *capability coefficient* (right hand column), which is the observed quops as a percentage of the number of quops in the target circuit (Q_T). The observed capability is computed from the effective error rates (Fig. 5) observed in each SVB experiment, and we compare it to the capability predicted from the lowest width circuits (Q_0). The *scalability coefficient* Q_C/Q_0 quantifies the effect of complex errors (e.g., crosstalk) in degrading system performance on wide circuits.

executing the target circuit with a fidelity of around 10^{-1700} (see Table I), which could not feasibly be measured by just running that target circuit.

IV. CONCLUSION

Quantum computer benchmarks can enable measuring progress towards quantum utility, but only if they are scalable, reliable, and measure well-motivated metrics. In this work, we have introduced subcircuit volumetric benchmarking (SVB), which is a method for creating benchmarks that meet these criteria out of any quantum algorithm or computational problem. The central idea of SVB is to run subcircuits snipped out from a potentially very large circuit that implements an algorithm or algorithmic subroutine of interest. By snipping out circuits of various shapes, we can measure how complex and arbitrary kinds of errors like crosstalk, coherent, or non-Markovian errors impact execution of the target circuit—even on systems that are far too small or too noisy to run the target circuit with reasonable fidelity. SVB data enables summarizing performance using effective error rates, and can even be used to define an *observed quops* that can be compared directly to the size of the target circuit. This simple idea enables predicting the fidelity with which a larger or better (but otherwise comparable) quantum computer would run the target circuit. How close *this* system is to successfully running the target circuit can be concisely summarized by the capability coefficient, which is the observed quops divided by the number of quops in the target circuit (and, herein, expressed as a percentage).

Quantum utility will likely require FTQC architectures [73] to obtain the large number of quops (low error rates) that appear necessary for solving practical problems with quan-

tum algorithms [1, 31–33]. We demonstrated SVB with circuits run directly on physical qubits—known as “NISQ computing”—but SVB can equally well be applied to circuits run on fault-tolerant logical qubits protected by error correction. SVB could be used as part of a framework for comparing NISQ computing and FTQC solutions to the same computational problem: a NISQ compilation and an FTQC compilation for a problem could be defined, the effective quops of a system (running in NISQ and FTQC modes) measured for each compilation, and the capability coefficient computed for each. However, FTQC and NISQ compilations of algorithms are *very* different [73] (e.g., because universal gates in FTQC require techniques such as magic state distillation or cultivation) and the error processes that will occur on logical qubits in real FTQC systems remain to be seen.

ACKNOWLEDGMENTS

We gratefully acknowledge useful conversations with Anand Ganti, Lucas Kovalsky, Andrew Landahl, Alicia Magann, Benjamin Morrison, Jake Nelson, Corey Ostrove, Shivesh Pathak, Mohan Sarovar, and Noah Siekierski. This material was funded in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Quantum Testbed Pathfinder Program and in part by the U.S. Department of Energy, National Nuclear Security Administration, Advanced Simulation and Computing program. S.K.S. and A.D.B. acknowledge support from the Department of Energy (DOE) Office of Fusion Energy Sciences “Foundations for quantum simulation of warm dense matter” project. T.P. acknowledges support from an Office of Advanced Scientific Computing Research Early Career Award. Sandia National Laboratories is a multi-program laboratory managed and operated by National Technology and 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-NA-0003525. This research used IBM Quantum resources of the Air Force Research Laboratory. All statements of fact, opinion or

conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of the U.S. Department of Energy, or the U.S. Government, or IBM, or the IBM Quantum team.

-
- [1] T. Proctor, K. Young, A. D. Baczewski, and R. Blume-Kohout, Benchmarking quantum computers, *Nat. Rev. Phys.* **7**, 105 (2025).
 - [2] A. Hashim, L. B. Nguyen, N. Goss, B. Marinelli, R. K. Naik, T. Chistolini, J. Hines, J. P. Marceaux, Y. Kim, P. Gokhale, T. Tomesh, S. Chen, L. Jiang, S. Ferracin, K. Rudinger, T. Proctor, K. C. Young, R. Blume-Kohout, and I. Siddiqi, A practical introduction to benchmarking and characterization of quantum computers, *arXiv [quant-ph]* (2024), *arXiv:2408.12064 [quant-ph]*.
 - [3] T. Lubinski, S. Johri, P. Varosy, J. Coleman, L. Zhao, J. Necaise, C. H. Baldwin, K. Mayer, and T. Proctor, Application-Oriented performance benchmarks for quantum computing, *IEEE Transactions on Quantum Engineering* **4**, 1 (2023).
 - [4] M. Amico, H. Zhang, P. Jurcevic, L. S. Bishop, P. Nation, A. Wack, and D. C. McKay, Defining standard strategies for quantum benchmarks, *arXiv preprint arXiv:2303.02108* (2023).
 - [5] T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, Measuring the capabilities of quantum computers, *Nat. Phys.* **18**, 75 (2021).
 - [6] R. Blume-Kohout and K. C. Young, A volumetric framework for quantum computer benchmarks, *Quantum* **4**, 362 (2020).
 - [7] A. Erhard, J. J. Wallman, L. Postler, M. Meth, R. Stricker, E. A. Martinez, P. Schindler, T. Monz, J. Emerson, and R. Blatt, Characterizing large-scale quantum computers via cycle benchmarking, *Nat. Commun.* **10**, 5347 (2019).
 - [8] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, Validating quantum computers using randomized model circuits, *Phys. Rev. A* **100**, 032328 (2019).
 - [9] K. Chen, W. Fang, J. Guan, X. Hong, M. Huang, J. Liu, Q. Wang, and M. Ying, VeriQBench: A benchmark for multiple types of quantum circuits, *arXiv preprint arXiv:2206.10880* 1048550/*arXiv.2206.10880* (2022).
 - [10] Tomesh, Gokhale, Omole, Ravi, Smith, Vizslai, Wu, Hardavellas, Martonosi, and Chong, SupermarQ: A scalable quantum benchmark suite, in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Vol. 0 (2022) pp. 587–603.
 - [11] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, Experimental comparison of two quantum computing architectures, *Proc. Natl. Acad. Sci. U. S. A.* **114**, 3305 (2017).
 - [12] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. C. Pienti, M. Chmielewski, C. Collins, K. M. Hudek, J. Mizrahi, J. D. Wong-Campos, S. Allen, J. Apisdorf, P. Solomon, M. Williams, A. M. Ducore, A. Blinov, S. M. Kreikemeier, V. Chaplin, M. Keesan, C. Monroe, and J. Kim, Benchmarking an 11-qubit quantum computer, *Nat. Commun.* **10**, 5464 (2019).
 - [13] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, Full-stack, real-system quantum computer studies: architectural comparisons and design insights, in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19 (Association for Computing Machinery, New York, NY, USA, 2019) pp. 527–540.
 - [14] H. Donkers, K. Mesman, Z. Al-Ars, and M. Möller, QPack scores: Quantitative performance metrics for application-oriented quantum computer benchmarking, *arXiv preprint arXiv:2205.12142* 1048550/*arXiv.2205.12142* (2022).
 - [15] J. R. Finžgar, P. Ross, J. Klepsch, and A. Luckow, QUARK: A framework for quantum computing application benchmarking, *arXiv preprint arXiv:2202.03028* 1048550/*arXiv.2202.03028* (2022).
 - [16] D. Mills, S. Sivarajah, T. L. Scholten, and R. Duncan, Application-Motivated, holistic benchmarking of a full quantum computing stack, *arXiv preprint arXiv:2006.01273* 1048550/*arXiv.2006.01273* (2020).
 - [17] T. Lubinski, J. J. Goings, K. Mayer, S. Johri, N. Reddy, A. Mehta, N. Bhatia, S. Rappaport, D. Mills, C. H. Baldwin, L. Zhao, A. Barbosa, S. Maity, and P. S. Mundada, Quantum algorithm exploration using Application-Oriented performance benchmarks, *arXiv preprint arXiv:2402.08985* 1048550/*arXiv.2402.08985* (2024).
 - [18] T. Lubinski, C. Coffrin, C. McGeoch, P. Sathe, J. Apanavicius, and D. E. Bernal Neira, Optimization applications as quantum performance benchmarks, *arXiv preprint arXiv:2302.02278* 1048550/*arXiv.2302.02278* (2023).
 - [19] J.-S. Chen, E. Nielsen, M. Ebert, V. Inlek, K. Wright, V. Chaplin, A. Maksymov, E. Pérez, A. Poudel, P. Maunz, and J. Gamble, Benchmarking a trapped-ion quantum computer with 29 algorithmic qubits, *arXiv preprint arXiv:2308.05071* 1048550/*arXiv.2308.05071* (2023).
 - [20] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz, A generative modeling approach for benchmarking and training shallow quantum circuits, *npj Quantum Information* **5**, 45 (2019).
 - [21] A. Li and S. Krishnamoorthy, QASMBench: A low-level QASM benchmark suite for NISQ evaluation and simulation, *arXiv preprint arXiv:2005.13018* 1048550/*arXiv.2005.13018* (2020).
 - [22] N. Quetschlich, L. Burgholzer, and R. Wille, MQT bench: Benchmarking software and design automation tools for quantum computing, *Quantum* **7**, 1062 (2023).
 - [23] Y. Dong and L. Lin, Random circuit block-encoded matrix and a proposal of quantum LINPACK benchmark, *Phys. Rev. A* **103**, 062412 (2021).
 - [24] S. Martiel, T. Ayrat, and C. Allouche, Benchmarking quantum coprocessors in an Application-Centric, Hardware-Agnostic, and scalable way, *IEEE Transactions on Quantum Engineering* **2**, 1 (2021).
 - [25] W. van der Schoot, D. Leermakers, R. Wezeman, N. Neumann, and F. Phillipson, Evaluating the q-score of quantum annealers, *arXiv preprint arXiv:2208.07633* 1048550/*arXiv.2208.07633* (2022).
 - [26] W. van der Schoot, R. Wezeman, N. M. P. Neumann, F. Phillipson, and R. Kooij, Q-score Max-Clique: The first quantum metric evaluation on multiple computational paradigms, *arXiv preprint arXiv:2302.00639* 1048550/*arXiv.2302.00639* (2023).
 - [27] A. Cornelissen, J. Bausch, and A. Gilyén, Scalable bench-

- marks for Gate-Based quantum computers, arXiv preprint arXiv:2104.10698 1048550/arXiv.2104.10698 (2021).
- [28] K. Georgopoulos, C. Emary, and P. Zuliani, Quantum computer benchmarking via quantum algorithms, arXiv preprint arXiv:2112.09457 1048550/arXiv.2112.09457 (2021).
 - [29] Y. Dong, K. B. Whaley, and L. Lin, A quantum hamiltonian simulation benchmark, npj Quantum Information **8**, 1 (2022).
 - [30] A. Chatterjee, S. Rappaport, A. Giri, S. Johri, T. Proctor, D. E. B. Neira, P. Sathe, and T. Lubinski, A comprehensive cross-model framework for benchmarking the performance of quantum hamiltonian simulations, IEEE Trans. Quantum Eng. **6**, 1 (2025).
 - [31] C. Gidney, How to factor 2048 bit rsa integers with less than a million noisy qubits, arXiv preprint arXiv:2505.15917 10.48550/arXiv.2505.15917 (2025).
 - [32] G. H. Low, R. King, D. W. Berry, Q. Han, A. E. De-Prince III, A. White, R. Babbush, R. D. Somma, and N. C. Rubin, Fast quantum simulation of electronic structure by spectrum amplification, arXiv preprint arXiv:2502.15882 10.48550/arXiv.2502.15882 (2025).
 - [33] N. C. Rubin, D. W. Berry, A. Kononov, F. D. Malone, T. Khattar, A. White, J. Lee, H. Neven, R. Babbush, and A. D. Baczewski, Quantum computation of stopping power for inertial fusion target design, Proc. Natl. Acad. Sci. U. S. A. **121**, e2317772121 (2024).
 - [34] J. Emerson, R. Alicki, and K. Życzkowski, Scalable noise estimation with random unitary operators, J. Opt. B Quantum Semiclassical Opt. **7**, S347 (2005).
 - [35] J. Emerson, M. Silva, O. Moussa, C. Ryan, M. Laforest, J. Baugh, D. G. Cory, and R. Laflamme, Symmetrized characterization of noisy quantum processes, Science **317**, 1893 (2007).
 - [36] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland, Randomized benchmarking of quantum gates, Phys. Rev. A **77**, 012307 (2008).
 - [37] E. Magesan, J. M. Gambetta, and J. Emerson, Scalable and robust randomized benchmarking of quantum processes, Phys. Rev. Lett. **106**, 180504 (2011).
 - [38] T. J. Proctor, A. Carignan-Dugas, K. Rudinger, E. Nielsen, R. Blume-Kohout, and K. Young, Direct randomized benchmarking for multiqubit devices, Phys. Rev. Lett. **123**, 030503 (2019).
 - [39] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, Detecting crosstalk errors in quantum information processors, Quantum **4**, 321 (2020).
 - [40] J. M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, J. A. Smolin, J. M. Chow, C. A. Ryan, C. Rigetti, S. Poletto, T. A. Ohki, M. B. Ketchen, and M. Steffen, Characterization of addressability by simultaneous randomized benchmarking, Phys. Rev. Lett. **109**, 240504 (2012).
 - [41] T. Proctor, S. Seritan, K. Rudinger, E. Nielsen, R. Blume-Kohout, and K. Young, Scalable randomized benchmarking of quantum computers using mirror circuits, Phys. Rev. Lett. **129**, 150502 (2022).
 - [42] R. Harper and S. T. Flammia, Learning correlated noise in a 39-qubit quantum processor, PRX quantum **4**, 040311 (2023).
 - [43] J. Hines and T. Proctor, Scalable Full-Stack benchmarks for quantum computers, arXiv preprint arXiv:2312.14107 1048550/arXiv.2312.14107 (2023).
 - [44] J. A. Smolin, G. Smith, and A. Vargo, Oversimplifying quantum factoring, Nature **499**, 163 (2013).
 - [45] G. H. Low and I. L. Chuang, Hamiltonian simulation by uniform spectral amplification, arXiv preprint arXiv:1707.05391 10.48550/arXiv.1707.05391 (2017).
 - [46] S. Chakraborty, A. Gilyén, and S. Jeffery, The power of block-encoded matrix powers: improved regression techniques via faster hamiltonian simulation, arXiv preprint arXiv:1804.01973 10.48550/arXiv.1804.01973 (2018).
 - [47] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe, Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics, in *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing* (2019) pp. 193–204.
 - [48] That is the qubits in q_w correspond to a connected sub-graph of Q 's connectivity graph.
 - [49] We also relabel the qubits in our snippet, according to the equivalence mapping from our subset of Q to Q' .
 - [50] T. Proctor, S. Seritan, E. Nielsen, K. Rudinger, K. Young, R. Blume-Kohout, and M. Sarovar, Establishing trust in quantum computations, arXiv preprint arXiv:2204.07568 1048550/arXiv.2204.07568 (2022).
 - [51] M. Seth, P. Timothy, F. Samuele, H. Jordan, B. Samantha, L. C. G. Govia, and M. David, When clifford benchmarks are sufficient; estimating application performance with scalable proxy circuits, arXiv [quant-ph] (2025), arXiv:2503.05943 [quant-ph].
 - [52] There are two sources of approximation. First, there is an $O(1/4^w)$ correction factor that is related to the difference between process fidelity and process polarization [2] (which, because we are typically interested in $w \gg 1$, we ignore for simplicity). Second, stochastic errors can cancel out within a circuit, and the exact rate of cancellation varies from circuit to circuit and depends on the exact biases in the errors. The impact of this effect on $F_{w,d}$ is at most $O(\epsilon^2 w^2 d^2)$ but, in practice, it causes a negligible correction to Eq. (2).
 - [53] S. Ferracin, S. T. Merkel, D. McKay, and A. Datta, Experimental accreditation of outputs of noisy quantum computers, Phys. Rev. A **104**, 042603 (2021).
 - [54] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya, *et al.*, Quantum chemistry in the age of quantum computing, Chemical reviews **119**, 10856 (2019).
 - [55] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, Quantum computational chemistry, Reviews of Modern Physics **92**, 015003 (2020).
 - [56] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, Simulated quantum computation of molecular energies, Science **309**, 1704 (2005).
 - [57] S. Pathak, A. E. Russo, S. K. Seritan, and A. D. Baczewski, Quantifying T-gate-count improvements for ground-state-energy estimation with near-optimal state preparation, Physical Review A **107**, L040601 (2023).
 - [58] D. W. Berry, Y. Tong, T. Khattar, A. White, T. I. Kim, G. H. Low, S. Boixo, Z. Ding, L. Lin, S. Lee, *et al.*, Rapid initial-state preparation for the quantum simulation of strongly correlated molecules, PRX Quantum **6**, 020327 (2025).
 - [59] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, Encoding electronic spectra in quantum circuits with linear T complexity, Physical Review X **8**, 041015 (2018).
 - [60] A. E. Russo, K. M. Rudinger, B. C. Morrison, and A. D. Baczewski, Evaluating energy differences on a quantum computer with robust phase estimation, Physical review letters **126**, 210501 (2021).
 - [61] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang, Grand unification of quantum algorithms, PRX quantum **2**, 040203 (2021).

- [62] J. S. Nelson and A. D. Baczewski, Assessment of quantum phase estimation protocols for early fault-tolerant quantum computers, *Physical Review A* **110**, 042420 (2024).
- [63] G. H. Low and I. L. Chuang, Hamiltonian simulation by qubitization, *Quantum* **3**, 163 (2019).
- [64] R. King, G. H. Low, R. Babbush, R. D. Somma, and N. C. Rubin, Quantum simulation with sum-of-squares spectral amplification, *arXiv preprint arXiv:2505.01528* 10.48550/arXiv.2505.01528 (2025).
- [65] A. M. Childs and N. Wiebe, Hamiltonian simulation using linear combinations of unitary operations, *Quantum Information and Computation* **12**, 901 (2012).
- [66] C. Hempel, C. Maier, J. Romero, J. McClean, T. Monz, H. Shen, P. Jurcevic, B. P. Lanyon, P. Love, R. Babbush, A. Aspuru-Guzik, R. Blatt, and C. F. Roos, Quantum Chemistry Calculations on a Trapped-Ion Quantum Simulator, *Phys. Rev. X* **8**, 031022 (2018), 1803.10238.
- [67] O. G. Maupin, A. D. Baczewski, P. J. Love, and A. J. Landahl, Variational Quantum Chemistry Programs in JaqalPac, *Entropy* **23**, 657 (2021).
- [68] S. B. Bravyi and A. Y. Kitaev, Fermionic quantum computation, *Annals of Physics* **298**, 210 (2002).
- [69] P. Jordan and E. Wigner, Über das Paulische Äquivalenzverbot, *Zeitschrift für Physik* **47**, 631 (1928).
- [70] S. Bravyi, J. M. Gambetta, A. Mezzacapo, and K. Temme, Tapering off qubits to simulate fermionic hamiltonians, *arXiv preprint arXiv:1701.08213* 10.48550/arXiv.1701.08213 (2017).
- [71] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. van den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, and A. Kandala, Evidence for the utility of quantum computing before fault tolerance, *Nature* **618**, 500 (2023).
- [72] We do not distinguish between idle operations and gates, because, in real systems, idle operations are often as noisy as gates and cannot be *a priori* assumed to have very low error.
- [73] E. T. Campbell, B. M. Terhal, and C. Vuillot, Roads towards fault-tolerant universal quantum computation, *Nature* **549**, 172 (2017).
- [74] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, Transformation of quantum states using uniformly controlled rotations, *Quantum Inf. Comput.* **5**, 467 (2005).
- [75] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. d. Silva, A divide-and-conquer algorithm for quantum state preparation, *Sci. Rep.* **11**, 6329 (2021).
- [76] Q. Sun, Libcint: An efficient general integral library for gaussian basis functions, *Journal of computational chemistry* **36**, 1664 (2015).
- [77] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, *et al.*, Pyscf: the python-based simulations of chemistry framework, *Wiley Interdisciplinary Reviews: Computational Molecular Science* **8**, e1340 (2018).
- [78] Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, *et al.*, Recent developments in the pyscf program package, *The Journal of chemical physics* **153**, 10.1063/5.0006074 (2020).
- [79] J. R. McClean, N. C. Rubin, K. J. Sung, I. D. Kivlichan, X. Bonet-Monroig, Y. Cao, C. Dai, E. S. Fried, C. Gidney, B. Gimby, *et al.*, Openfermion: the electronic structure package for quantum computers, *Quantum Science and Technology* **5**, 034014 (2020).
- [80] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, *(tket): a retargetable compiler for nisq devices*, *Quantum Science and Technology* **6**, 014003 (2020).
- [81] E. Nielsen, K. Rudinger, T. Proctor, A. Russo, K. Young, and R. Blume-Kohout, Probing quantum processor performance with pygsti, *Quantum science and Technology* **5**, 044002 (2020).
- [82] A. Cross, A. Javadi-Abhari, T. Alexander, N. De Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, *et al.*, Openqasm 3: A broader and deeper quantum assembly language, *ACM Transactions on Quantum Computing* **3**, 1 (2022).
- [83] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, *et al.*, Quantum computing with qiskit, *arXiv preprint arXiv:2405.08810* 10.48550/arXiv.2405.08810 (2024).

Appendix A: Circuit compilation

SVB takes, as its input, a circuit c that has been *fully compiled* for a particular quantum computer Q . In this appendix we provide additional details on what this means, and discuss it in the context of a quantum algorithm or application (which is the primary setting in which we anticipate SVB will be employed). A circuit c is fully compiled for Q if each qubit in c is assigned to a physical qubit in Q and if c contains only the native gates of Q , including only interacting qubits that are coupled in Q 's connectivity graph. There is some ambiguity about precisely what a quantum computer's "native gates" are (e.g., arbitrary single qubit gates are implementable by two rotations by $\pi/2$ around the X axis together with three rotations around the Z axis, which are typically implemented virtually, so an arbitrary single qubit gate might be considered a native operation). These somewhat arbitrary choices impact the exact meaning of a fully compiled circuit, and then impact the depth of a fully compiled circuit. This, however, has no important consequences for SVB—as the depth of a circuit snippet from c will typically be interpreted relatively to c 's depth—and so we do not need to be pedantic about precisely how the native gate set is defined.

The reason why we require that c is fully compiled is that compiling of the circuit snippets SVB samples from c is not permitted (except for trivial compilations, like replacing an X gate with two $X_{\pi/2}$ gates). Unless c is already fully compiled, this would prevent the execution of those snippets. The precise constraint we set on the compilations of our snippets can be formalized using the concept of *compilation barriers*: each layer of c is separated by a compilation barrier, and those barriers persist in our snippets. We demand no compilation of our snippets because otherwise it is possible that compilations could reduce the depth or change other characteristics of those snippets (e.g., adding ancillary qubits), making the fidelities of those snippets less likely to be predictive of the fidelity of Q .

To create a fully compiled c for SVB there is typically a multi-step process, which is illustrated in the top row of Fig. 1. SVB is, we anticipate, most interesting for the case of quantifying progress towards solving some computational problem with a quantum computer. Therefore, we must first select a problem instance and a quantum algorithm for solving this problem. Then all tunable parameters of the algorithm must be chosen (e.g., the number of Trotter steps in an algorithm

that uses Trotterization). These steps then typically produce one or more “high-level” circuits: circuits that are expressed in terms of subroutines that cannot be directly implemented on most or all quantum computing hardware (e.g., the quantum Fourier transform, or n -qubit Toffoli gates). This circuit then needs to be compiled to be executable on the particular quantum computer Q in question, which is a step that is often broken down into multiple stages, such as qubit mapping/routing. Note, however, that SVB can be separately applied to circuits created with different choices at one or more stages (e.g., different algorithm choices for the same problem) and, by doing so, the performance with different choices can be compared.

Appendix B: Hamiltonian simulation SVBs

In this appendix we provide additional details of the Hamiltonian simulation SVBs presented in the main text.

1. Hamiltonian block encoding circuits

We briefly review the block encoding framework [45–47] and, in particular, the LCU approach to this block encoding [65], which is what we use in our demonstration of SVB.

In general, a unitary U_A is an (α, m, ϵ) -block encoding of A if

$$\left\| A - \alpha \left(|0\rangle^{\otimes m} \otimes I_n \right) U_A \left(|0\rangle^{\otimes m} \otimes I_n \right) \right\| \leq \epsilon, \quad (\text{B1})$$

where α serves to scale A such that the norm of $U_A = 1$, n and m are the sizes of the system and auxiliary registers, respectively, and ϵ bounds the error of the block encoding. Suppose that the operator A can be expanded as a linear combination of L unitaries,

$$A = \sum_{l=0}^L c_l U_l, \quad (\text{B2})$$

where L necessarily sets the size of auxiliary register such that $L < 2^m - 1$. We can define a pair of state preparation unitaries U_{PREP} and U_{UNPREP} and a select unitary U_{SEL} as

$$U_{\text{PREP}}|0\rangle^{\otimes m} = \sum_{l=0}^L a_l |l\rangle, \quad (\text{B3})$$

$$U_{\text{UNPREP}}|0\rangle^{\otimes m} = \sum_{l=0}^L b_l |l\rangle, \quad (\text{B4})$$

$$U_{\text{SEL}} = \sum_{l=0}^L |l\rangle\langle l| \otimes U_l, \quad (\text{B5})$$

such that

$$\sum_{l=0}^L \left| a_l^* b_l - \frac{c_l}{\|c\|_1} \right| < \epsilon_1. \quad (\text{B6})$$

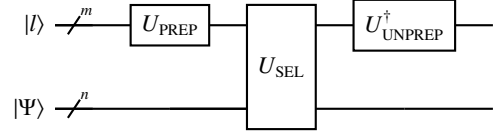


FIG. 6. General form of an LCU circuit.

Model System	System Qubits (n)	Number of LCU Terms (L)	Auxiliary Qubits ($2m - 1$)
H ₂	4	15	7
HeH ⁺	4	27	9
LiH	8	105	13

TABLE III. Number of qubits and terms needed for the LCU encoding of each model system. Auxiliary qubit counts include those needed for unary iteration.

The LCU encoding given by

$$U_A = (U_{\text{UNPREP}}^\dagger \otimes I_n) U_{\text{SEL}} (U_{\text{PREP}} \otimes I_n) \quad (\text{B7})$$

serves as a $(\|c\|_1, m, \epsilon_1)$ -block encoding of A . The circuit for this LCU encoding is depicted in Fig. 6.

In the case where all $c_l \geq 0$, the state preparation pair can be reduced to a single unitary with real coefficients

$$a_l = b_l = \sqrt{\frac{c_l}{\|c\|_1}}. \quad (\text{B8})$$

Alternatively, in the case where some $c_l < 0$, the state preparation pair unitaries must have coefficients a_l and b_l such that their inner product yields the requisite negative sign. One simple option is to set $a_l = -b_l$, which corresponds to flipping the sign of certain rotation angles in U_{UNPREP} .

An LCU circuit can be expressed in terms of many multi-controlled gates, as shown in the example of Fig. 7. The prepare circuit can be improved by using an efficient decomposition of uniformly controlled rotations that only consist of R_Y (a rotation around the Y axis) and CNOT gates [74]. The select circuit can be improved through a procedure known as “unary iteration” where additional qubits are introduced to store partial computations of the state in the auxiliary register [59]. For an LCU with m auxiliary qubits, unary iteration introduces $m - 1$ additional qubits but results in a massive reduction in depth of the final compiled circuit, relative to a naive implementation like the one in Fig. 7. Further implementation details for the prepare oracle are described below.

a. Angles for State Preparation

State preparation pair unitaries using uniformly controlled rotations such as the one shown in Fig. 7 are defined by their R_Y rotation angles θ_i^j , where i and j denote the controlling state and number of preceding qubits, respectively. Each angle

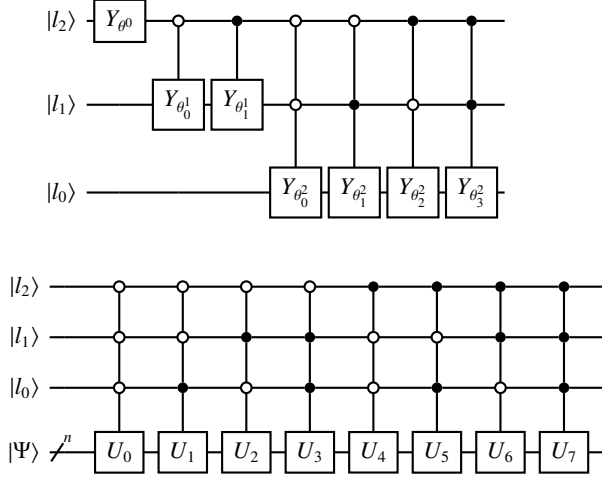


FIG. 7. **Prepare (top) and select (bottom) circuits.** An uncompiled LCU circuit consists of a prepare circuit and a select circuit. Here we show these circuit for an 8-term LCU circuit. The uncompiled prepare circuit (top) consists of controlled rotations with $Y_{\theta^j_i} \equiv R_Y(\theta^j_i)$ where i and j denote the controlling state and number of preceding qubits, respectively. The select unitary (bottom) consists of multi-controlled unitaries conditioned on the state of the auxiliary register.

can be computed [74] as

$$\theta^j_i = 2 \arcsin \left(\frac{\sqrt{\sum_{k=0}^{2^j-1} |a_{(2i+1)2^j+k}|^2}}{\sqrt{\sum_{k=0}^{2^{j+1}-1} |a_{i2^{j+1}+k}|^2}} \right), \quad (\text{B9})$$

where a_i are the coefficients of the prepared state from Eq. (B3). For m auxiliary qubits, the indexing quantities can vary as $j = 0, 1, \dots, m-1$ and $i = 0, 1, \dots, 2^{m-j}-1$.

However, one can also compute the angles given in Eq. (B9) using a binary trie where each node corresponds to a bit string and left/right children append a 0/1 to the bit string, respectively. Recent work on parallel implementations of state preparation unitaries [75] has utilized similar data structures, and we use their terminology of a “state tree” to denote a binary trie that stores coefficient information. Leaves hold the final coefficients, internal nodes hold partial norms of their chil-

dren needed to calculate the R_Y angles, and the root node necessarily holds a norm of 1. Each node’s location provides a bit string of the necessary control values for the multi-controlled- R_Y gates.

The rotation angles θ^j_i can now be calculated by navigating the state tree according to the bit string representation of i , which should reside on the j^{th} layer of the state tree. The current node’s partial norm will take the place of the denominator in Eq. (B9) while the right child’s partial norm takes the place of the numerator. Calculating all rotation angles needed for a full state preparation using the state tree is more efficient than using Eq. (B9) because there will be fewer partial norm summations.

The state tree can also be used to easily compute which angles will need to be flipped between U_{PREP} and U_{UNPREP} to satisfy Eq. (B6). The needed sign can be constructed in a bottom-up approach using the following rules: 1) each leaf takes the sign of its coefficient, 2) a node’s angle must be flipped in U_{UNPREP} if the signs of its children don’t match, and 3) a node inherits the sign of its right child.

2. Implementation details

The LCU coefficients were computed with PySCF [76–78]. OpenFermion [79] was used to perform the Jordan-Wigner and Bravyi-Kitaev mappings, utilizing OpenFermion-PySCF to interface between the two packages. The LCU circuits were then expressed in PyTket [80] using high-level operations such as multi-controlled Pauli strings before being converted to the TK1 and CNOT gate set. PyTket’s full peephole optimization pass was applied to the circuit before the noise-aware mapping and routing passes were used to embed the qubits and enforce the correct CNOT connectivity for the target IBM Q device. The circuit was rebased once more into the U3 and CNOT gate set, which only requires a minor adjustment to the angles in the one-qubit gates, before being passed to PyGSTi [81]. Subcircuits of each target width and depth were sampled from the full circuit before the circuit mirroring procedure was carried out on each subcircuit. The U3 gates were decomposed into $\pi/2$ rotations around the X axis (IBM’s “sx” gate) and R_Z gates, converted into QASM [82], and submitted to IBM Q using Qiskit [83].