

# Application-Oriented Performance Benchmarks for Quantum Computing

Thomas Lubinski,<sup>1,2</sup> Sonika Johri,<sup>3</sup> Paul Varosy,<sup>4</sup> Jeremiah Coleman,<sup>5</sup> Luning Zhao,<sup>3</sup> Jason Necaie,<sup>6</sup> Charles H. Baldwin,<sup>7</sup> Karl Mayer,<sup>7</sup> and Timothy Proctor<sup>8</sup>

(Quantum Economic Development Consortium (QED-C) collaboration)\*

<sup>1</sup>*Quantum Circuits Inc, 25 Science Park, New Haven, CT 06511*

<sup>2</sup>*QED-C Technical Advisory Committee on Standards and Performance Benchmarks Chairman*

<sup>3</sup>*IonQ Inc, 4505 Campus Dr, College Park, MD 20740, USA*

<sup>4</sup>*Department of Physics, Colorado School of Mines, Golden, CO 80401, USA*

<sup>5</sup>*Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ, 08544, USA*

<sup>6</sup>*D-Wave Systems, Burnaby, British Columbia, Canada, V5G 4M9, Canada*

<sup>7</sup>*Quantinuum, 303 S. Technology Ct, Broomfield, CO 80021, USA*

<sup>8</sup>*Quantum Performance Laboratory, Sandia National Laboratories, Albuquerque, NM 87185, USA and Livermore, CA 94550, USA*

(Dated: January 3, 2022)

In this work we introduce an open source suite of quantum application-oriented performance benchmarks that is designed to measure the effectiveness of quantum computing hardware at executing quantum applications. These benchmarks probe a quantum computer's performance on various algorithms and small applications as the problem size is varied, by mapping out the fidelity of the results as a function of circuit width and depth using the framework of volumetric benchmarking. In addition to estimating the fidelity of results generated by quantum execution, the suite is designed to benchmark certain aspects of the execution pipeline in order to provide end-users with a practical measure of both the quality of and the time to solution. Our methodology is constructed to anticipate advances in quantum computing hardware that are likely to emerge in the next five years. This benchmarking suite is designed to be readily accessible to a broad audience of users and provides benchmarks that correspond to many well-known quantum computing algorithms.

CONTENTS		VI. Highlights and Impressions	15
I. Introduction	2	VII. Summary and Conclusions	18
II. Background	3	Code Availability	19
A. Component-Level Specifications	3	Acknowledgement	19
B. Quantum Volume	4		
C. Volumetric Benchmarks	4	A. Selected Algorithms and Applications	20
III. Application-Oriented Benchmarks	5	1. Shallow Simple Oracle Based Algorithms	20
A. The Benchmarking Suite	5	2. Quantum Fourier Transform	21
B. Volumetric Positioning	5	3. Grover's Search Algorithm	22
C. Circuit Depth	6	4. Phase and Amplitude Estimation	23
D. Circuit Fidelity	7	5. Hamiltonian Simulation	23
IV. Benchmark Results and Analysis	7	6. Monte Carlo Sampling	24
A. Benchmark Results on Quantum Simulator	8	7. Variational Quantum Eigensolver	25
B. Benchmark Results on Quantum Hardware	8	8. Shor's Order Finding	27
C. Comparison to Generic Benchmarks	12	B. Advances in Quantum Computers	28
V. Measuring Execution Time	13	1. Mid-Circuit Measurements	28
A. Total Execution Time	14	2. Circuit Parameterization	28
B. Quantum Execution Time	14	3. Multiply-Controlled Gates	29
		4. Close Classical/Quantum Integration	29
		5. Qubit Connectivity Improvements	30
		References	31

\* This work was sponsored by the Quantum Economic Development Consortium (QED-C) and was performed under the auspices of the QED-C Technical Advisory Committee on Standards and Performance Benchmarks. The authors acknowledge many committee members for their input to and feedback on the project and this manuscript.

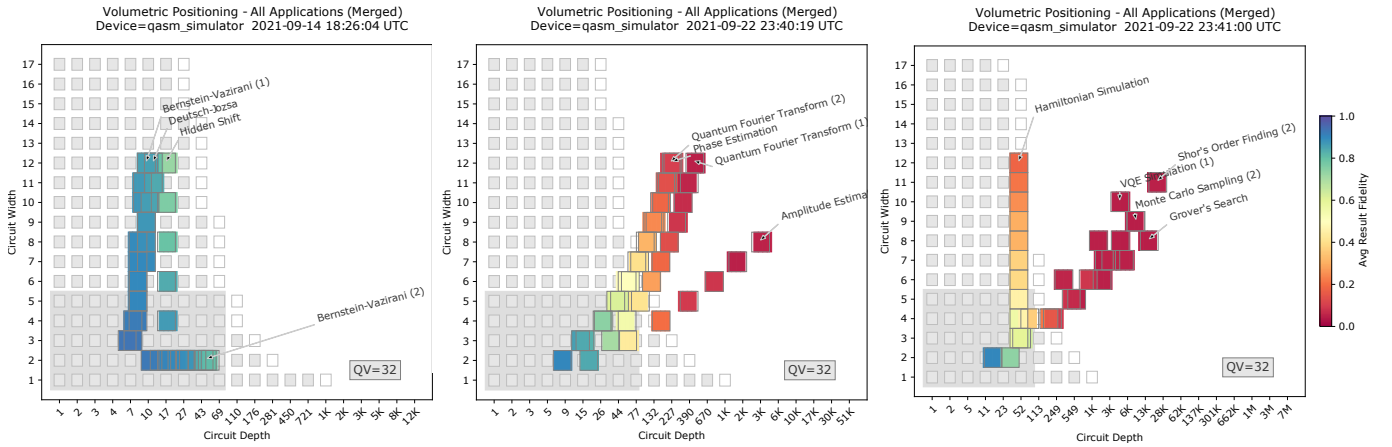


FIG. 1. **Quantum Application-Oriented Performance Benchmarks.** The results of executing our quantum application-oriented performance benchmarking suite on a simulator of a noisy quantum computer, with results split into benchmarks based on three loose categories of algorithm: tutorial, subroutine, and functional. For each benchmark, circuits are run with a variety of widths, corresponding to the problem size, here ranging from 2 to 12 qubits. The result fidelity, a measure of the result quality, is computed for each circuit execution, and is shown as a colored square positioned at the corresponding circuit’s width and normalized depth. Results for circuits of equal width and similar depth are averaged together. The results of the application-oriented benchmarks are shown on top of a ‘volumetric background’ (grey-scale squares) which extrapolates from a device’s quantum volume (here, 32) to predict the region in which a circuit’s result fidelity will be above  $1/2$  (the grey squares).

## I. INTRODUCTION

Over the past decade, quantum computers have evolved from exploratory physics experiments into cloud-accessible hardware, heralding their transformation into early-stage commercial products. A broad audience of users has now begun to explore and interact with quantum computers, inspiring research into practical quantum algorithms [1] and garnering attention from the academic, government, and business communities. With recent demonstrations of quantum advantage [2, 3], it seems increasingly likely that quantum computers could soon outperform their classical counterparts for practically relevant tasks. Unlike contemporary classical computers, though, the capabilities of current quantum computers are limited primarily by errors, not by their size or speed.

Quantum computers can experience a wide range of complex errors. Many of these errors result from noise in or miscalibrations of the lowest-level components in the system — quantum gate operations and the qubits they act on. Component-level benchmarks and characterization protocols, such as randomized benchmarking [4, 5] or gate set tomography [6], can provide insight into the type and magnitude of these errors. These tools are critical for experimental efforts to improve hardware. But users seeking to run an application on a quantum computer are typically concerned not with low-level details, but rather how likely their application is to execute successfully. Extrapolating low-level performance metrics to predict the performance of a specific application is challenging, and it is often not possible to do accurately [7, 8].

There has therefore been an increasing focus on benchmarks that concisely summarize the high-level performance of a quantum computer [8–10]. A notable example is the quantum volume benchmark [10], which is designed to probe the effective

useful size of a specific quantum computer and summarize it in one number: the quantum volume. The quantum volume has been widely adopted and reported by the community. However, due to the complexity of errors in quantum hardware, neither a device’s quantum volume nor any other single metric is likely to accurately predict its performance on all applications [7, 8]. There is thus a pressing need for a diverse set of application-oriented metrics and benchmarks that test the performance of quantum computers on practically relevant tasks. Such benchmarks will enable hardware developers to quantify their progress in commercially relevant ways, and will make it possible for end users to more accurately predict how the available hardware will perform on their application.

In this paper, we introduce an extensible suite of quantum application-oriented performance benchmarks, complementing other recent methods [11–17] that use, e.g., small chemistry problems or basic quantum circuits as benchmarks. Demonstrated in Figure 1, each benchmark in our suite is derived from an algorithm or application and specifies a scalable family of quantum circuits. The benchmarks are designed to evaluate the capability of quantum hardware to successfully execute a meaningful computational task, and are intended to reflect likely use cases for quantum computers. The benchmarking suite is available as a public, open-source repository with extensive documentation [18]. Validated implementations of our benchmarks are supplied in multiple common quantum programming languages, including Qiskit [19], Cirq [20], Braket [21] and Q# [22], and can be rapidly deployed on nearly all major cloud quantum computers.

Our benchmarking suite is designed to measure proxies for the quality of and the time to solution for each application. In analyzing the results of these benchmarks, we adopt the framework of volumetric benchmarking [23], as shown in Figure 1. This approach, which generalizes the quantum volume,

displays the result quality for each application as a function of the problem size (circuit width) and the computation's length (circuit depth). To do so, we define a normalized measure of result quality that makes results from different applications comparable, and we define a normalized, device-independent measure of circuit depth that removes complications associated with diverse native gate sets and dissimilar device connectivity. The resulting analysis enables easy comparison of how various quantum devices execute the same computational tasks.

The remainder of this paper is structured as follows. Section II provides background on the fundamentals guiding this work. Section III introduces our application-oriented benchmarks and our methodology. In Section IV we present results from executing subsets of the benchmark suite on quantum simulators and quantum hardware. We discuss the challenges of measuring execution time in Section V. In Section VI we discuss the limitations of and opportunities for improvements to our benchmarking suite, and we conclude in Section VII. Appendix A contains a detailed description of each of our application-oriented benchmarks. In Appendix B we analyze the adaptations to our suite that will be required to address near-term technological enhancements such as mid-circuit measurements, parameterized gate operations, and hybrid classical-quantum computations.

## II. BACKGROUND

In this section we review the methods on which our work is based. To place our work in context, in Section II A we begin with a brief discussion of the component-level performance metrics that are typically available to users of the current generation of hardware. Throughout this paper we will compare a quantum computer's performance on our benchmarks with its quantum volume, so in Section II B we review the quantum volume metric. To present results from our benchmarks we will primarily use the framework of volumetric benchmarking, so in Section II C we review this methodology and explain how it provides the underpinning for the analysis of our application benchmarks.

### A. Component-Level Specifications

There are many techniques for estimating component-level performance of quantum computing devices, e.g., randomized benchmarking [4, 5] and gate set tomography [6]. Hardware providers use these techniques to calibrate and test their systems, and they often publish performance metrics extracted from the results of these experiments. The measures of performance that providers report to users of their systems typically include one- and two-qubit gate fidelity, state preparation and measurement (SPAM) fidelity, and measures of how long each qubit can retain quantum information (T1 and T2 coherence times). For example, Figure 2 shows the contents of the text file that the Amazon Braket cloud service [24] presents to users of the IonQ Quantum Processing Unit (QPU), and Figure 3 shows similar information that Google [25] provide for its We-

```

1 {
2   "braketSchemaHeader": {
3     "name": "braket.device_schema.ionq.ionq_provider_properties",
4     "version": "1"
5   },
6   "fidelity": {
7     "1Q": {
8       "mean": 0.99717
9     },
10    "2Q": {
11      "mean": 0.9696
12    },
13    "spam": {
14      "mean": 0.9961
15    }
16  },
17  "timing": {
18    "T1": 10000,
19    "T2": 0.2,
20    "1Q": 0.000011,
21    "2Q": 0.00021,
22    "readout": 0.000175,
23    "reset": 0.000035
24  }
25 }

```

FIG. 2. An example of component-level performance metrics. The component-level performance metrics provided by Amazon Braket for the IonQ Quantum Processing Unit.

Metric	Symbol	Low <sup>1</sup>	Typ <sup>2</sup>	High <sup>3</sup>	Units
Single-qubit gate error rate	e1	0.1	0.1	0.2	% error per gate
Two-qubit gate error rate ( $\sqrt{\text{ISWAP}}$ )	e2 ( $\sqrt{\text{ISWAP}}$ )	0.7	0.9	1.9	% error per gate
		0.8	1.4	3.3	% error per gate
Readout error  0⟩	ero	0.5	1.1	2.6	% error
		1	2	3	% error
Readout error  1⟩	eri	3	5	9	% error
		3	7	9	% error

FIG. 3. An example of component-level performance metrics. The component-level performance metrics provided by Google for its Weber device.

ber device (Sycamore class). Other quantum computing cloud services, such as IBM Quantum Services [26], provide similar types of information about their machines.

Component-level performance metrics provide a lot of information about a machine, but they have at least two important limitations. Firstly, component-level performance metrics are typically not sufficient to accurately predict a device's performance on an algorithm [7, 8]. This is because summary metrics cannot capture the full complexity of errors in quantum computing components (e.g., individual gates and readouts), and not all sources of error are even apparent when testing a single component (e.g., crosstalk [27, 28]). Second, component-level metrics are challenging for non-specialists to interpret, i.e., it is often difficult for a prospective quantum computer user to extrapolate from component-level metrics to predict how their application will perform.

## B. Quantum Volume

In this work we compare results from our benchmarks to predictions extrapolated from a device’s quantum volume, and so we now review the quantum volume. IBM introduced the quantum volume (QV) metric [10] in recognition of the need for a simple way to quantify and compare the capabilities of quantum computing devices. The quantum volume is a single metric that is influenced by multiple factors contributing to the overall performance of a quantum computer, including its number of qubits, systemic errors, device connectivity, and compiler efficiency. Several vendors and users have welcomed this as a step towards quantifying what a quantum computer can do [29]. IBM publishes the quantum volume for each machine that it makes available to external users through IBM Quantum Services [26], as shown in Figure 4. Note that a device with more qubits does not always have a larger quantum volume.

A quantum computer’s quantum volume is defined in terms of its average performance on a set of random circuits, which we refer to as the quantum volume circuits. An  $n$ -qubit quantum volume circuit is a square circuit consisting of  $n$  layers of Haar-random  $SU(4)$  unitaries between  $\lfloor n/2 \rfloor$  randomly selected qubit pairs (where  $\lfloor n/2 \rfloor$  rounds  $n/2$  down to the nearest integer). A device’s quantum volume is defined to be  $2^n$  with  $n$  being both the width (the number of qubits) and the depth (the number of layers) of the largest quantum volume circuits that can be executed ‘successfully’ on a specified quantum device. So a quantum computer that can successfully execute 5 layer quantum volume circuits on 5 qubits, but not 6 layer quantum volume circuits on 6 qubits, would have a quantum volume of  $2^5$  or 32. A device is considered to implement  $n$ -qubit quantum volume circuits successfully if, on average, it produces more ‘heavy outputs’ than a set threshold of  $2/3$  with two-sigma confidence. Quantum volume is defined as the exponential of the circuit depth as this is intended to represent the challenge of simulating the same circuit on a classical computer. Note, however, that arguably no single number can summarize the many dimensions that make up a quantum computer’s performance characteristics [7, 30]. For example, the quantum volume is not guaranteed to reliably predict the performance of wide and shallow circuits, or deep and narrow circuits.

## C. Volumetric Benchmarks

To present results from our benchmarks we will primarily use the framework of volumetric benchmarking [23], which we now review. Volumetric benchmarking is a framework for constructing and visualizing the results of large, flexible families of benchmarks. It consists of (1) choosing a family of circuits to use as a benchmark, (2) selecting circuits from this family and executing them on the hardware to be benchmarked, and then (3) plotting the observed performance of the hardware on these circuits as a function of the circuits’ width and depth. The results are plotted in the depth  $\times$  width ‘volumetric space’, as illustrated in Figure 5. The data points (here, grey-scale squares) show the result quality for the tested circuits of that

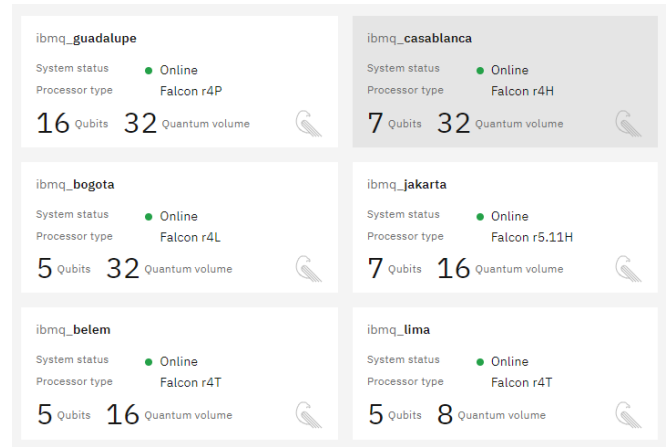


FIG. 4. **Quantum volume examples.** The number of qubits and the quantum volume for a selection of IBM Q machines, accessed via IBM Quantum Services.

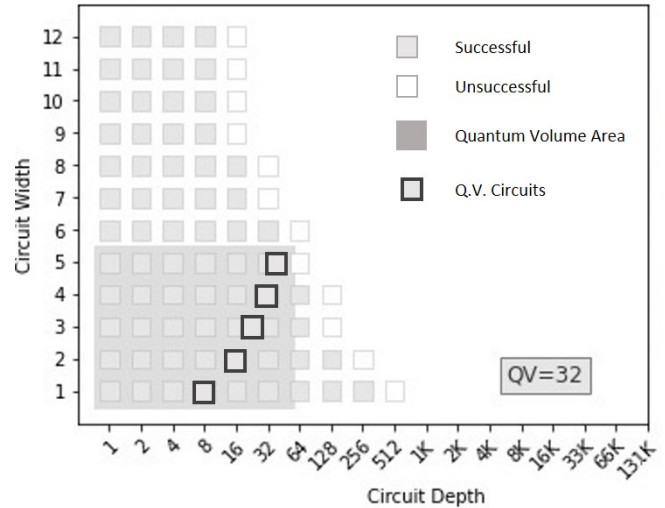


FIG. 5. **An example of volumetric benchmarking.** Volumetric benchmarking results for a hypothetical 12-qubit quantum computer, whereby a quantum computer’s performance on some family of circuits is mapped out as a function of circuit width and depth. This example uses a binary measure of performance: grey and white squares show the circuit shapes at which the test circuits succeeded and failed, respectively, according to this metric. The location of the quantum volume circuits, that would be used to extract the quantum volume, are shown in bold. Here and throughout, ‘depth’ for all circuits is defined with respect to a particular standardized gate set, and in this gate set the quantum volume circuits are not square.

shape (width and depth), and in this example the metric of result quality takes one of two values: ‘success’ (grey squares) or ‘fail’ (white squares). Note that Figure 5 also contains the location of the quantum volume circuits (black squares) and a quantum volume region (grey rectangle) that we explain in Section III B.

Volumetric benchmarking can be used to estimate and report the performance of both wide shallow circuits and narrow deep circuits, in addition to square circuits like those tested in the

quantum volume benchmark. Furthermore, volumetric benchmarking is not restricted to a single kind of circuit (e.g., the specific random circuits of the quantum volume benchmark). This makes volumetric benchmarking an ideal framework within which to place application-oriented benchmarks.

### III. APPLICATION-ORIENTED BENCHMARKS

In this section we introduce our application-oriented benchmarking suite, and we explain how we analyze the data and present the results. In Section III A we introduce the benchmarking suite and methodology. In Section III B we explain how we present the results of our benchmarking over a volumetric background. This method for presenting our results relies on normalized definitions for circuit depth and result fidelity, and so we introduce these definitions in Section III C and Section III D.

#### A. The Benchmarking Suite

The algorithms and applications on which our benchmarks are based are summarized in Table I. Each of our benchmarks is based on one of these algorithms, but an algorithm does not itself define a unique benchmark. In some cases, we developed more than one benchmark based on an algorithm. Complete specifications for each benchmark are provided in Appendix A.

Application-Oriented Benchmarks	
Tutorial	Deutsch-Jozsa Bernstein-Vazirani Hidden Shift
	Quantum Fourier Transform Phase Estimation Amplitude Estimation
Subroutine	Grover's Search Hamiltonian Simulation Monte Carlo Sampling Variational Quantum Eigensolver Shor's Period Finding
Functional	

TABLE I. The algorithms on which our application-oriented benchmarking suite is based. They are separated into three categories: tutorial, subroutine, and functional.

We refer to our benchmarks as ‘application-oriented’, as the suite consists of both quantum algorithms that are a key component of many applications (e.g., the Quantum Fourier Transform) and simple applications (e.g., Monte Carlo Sampling). The benchmarks fall loosely into three categories, as shown in Table I. The ‘tutorial’ category is drawn from simple examples, often found in tutorials, that can be implemented with shallow circuits. The ‘subroutine’ category consists of algorithms or code modules commonly used as a part of larger, functional applications. The ‘functional’ category are closer to

complete applications, but they do not necessarily correspond exactly to a complete application.

Each of our application benchmarks consists of (1) choosing a range of problem sizes at which to probe performance, (2) generating a set of circuits for each problem size, that consists of the algorithm’s circuits or some simple adaptation of those circuits with randomly sampled algorithm inputs, and (3) executing the circuits and measuring key components of performance, notably the result fidelity and the execution time. We implement this methodology using Algorithm 1:

---

#### Algorithm 1 Benchmark Execution Loop

---

```

1: target  $\leftarrow$  backend_id
2: initialize_metrics()
3: for num_qubits  $\leftarrow$  min, max_qubits do                                 $\triangleright N_1$ 
4:   for circuit_id  $\leftarrow$  1, max_circuits do                             $\triangleright N_2$ 
5:     execute_classical()
6:     circuit  $\leftarrow$  create_circuit(circuit_id)
7:     circuit  $\leftarrow$  compile_load_circuit(circuit)
8:     for params  $\leftarrow$  1, max_params do                                 $\triangleright N_3$ 
9:       execute_quantum(target, circuit, params, shots)
10:    end for
11:    collect_circuit_metrics()
12:  end for
13: collect_group_metrics()
14: end for

```

---

The algorithm defines three nested loops. The outer loop (labelled  $N_1$ ) is a sweep over the range of problem sizes, which define the circuit width. The middle loop ( $N_2$ ) represents multiple interleaved executions of classical and quantum execution routines. In all our existing benchmarks the classical routines consist solely of circuit construction and compilation. Each iteration of the middle loop constructs a quantum circuit, which can be a parameterized circuit. The inner loop ( $N_3$ ) consists of executing that circuit, for some range of parameter values if it is a parameterized circuit.

#### B. Volumetric Positioning

Volumetric benchmarking provides an intuitive and practical framework for visualizing the results of our quantum application benchmarks. We now explain how we use volumetric benchmarking plots to summarize the results of our benchmarks. Each of our application-oriented benchmarks is associated with a family of circuits that is parameterized by the input problem size, i.e., each input size defines a circuit ensemble. Each application’s circuit family has a particular ‘footprint’ that represents the way in which the shape of the circuits — i.e., their width and depth — varies as a function of the input size. We can therefore report the performance of our application benchmarks in the depth  $\times$  width ‘volumetric space’. In this work, for each application a fixed input size always corresponds to a single circuit width. In contrast, an application’s circuits can have a range of depths for each input size. For example, the algorithms in the ‘tutorial’ category use a varied-depth oracle circuit that encodes the problem in-



stance. We therefore use the circuits’ average depth to define that ensemble’s volumetric position, and we report average performance over the circuit ensemble.

To illustrate this idea, we executed our benchmarking suite on a quantum simulator. The lower plot of Figure 6 shows the result fidelity for a single benchmark plotted in the volumetric space. Figure 1 shows the result fidelity for each of our benchmarks, plotted in the volumetric space and separated into the three categories of benchmark. Each benchmark consists of running its circuits for a range of problem sizes, which closely correspond to the circuits’ widths. Each data point (small rectangles) summarizes the results for the circuits of a particular problem size. The rectangle’s location indicates the circuits’ width and average depth, and the color indicates the average quality of the result for that circuit shape. We quantify the quality of the result using the average result fidelity, defined in Section III D. The width and depth of benchmarking circuits for different applications can be similar, resulting in overlap on the volumetric plots. Whenever this is the case the result fidelities from the overlapping circuits for different applications are averaged together. We use this averaging in Figure 1, and throughout this paper except where stated.

Throughout this work, the results of our benchmarks are placed on top of what we call a ‘volumetric background’, shown by the grey-scale rectangles in Figure 1. A volumetric background means a prediction for the result fidelity, as a function of circuit width and depth, obtained from some method that is independent of our application oriented benchmarks. In Figure 1 and throughout most of this paper, we use a volumetric background that is extrapolated from the corresponding device’s quantum volume. For visual clarity, we use a binary ‘success’ (grey squares) or ‘fail’ (white squares) background, with ‘success’ corresponding to a result fidelity of at least  $1/2$  [31]. We use this type of volumetric background throughout this paper, except where stated.

Predicting a circuit’s result fidelity from a processor’s quantum volume requires an extrapolation. We use the following methodology: a width  $w$  and depth  $d$  circuit is predicted to have a result fidelity above  $1/2$  if  $wd < (\log_2(V_Q))^2$  where  $V_Q$  is the quantum volume [10]. That is, a circuit is predicted to succeed if it’s depth  $\times$  width is no larger than the largest successful quantum volume circuits. Note that here, as throughout, we use the standardized circuit depth (see Section III C) and with this definition for the circuit depth quantum volume circuits are not square. Furthermore, note that we do not have a measured quantum volume for all devices that are tested in this paper. In all such cases, the quantum volume is approximated by visual inspection from the other benchmark data.

No volumetric background is likely to accurately predict the performance of all circuits, including our application oriented benchmarks. This is because a circuit’s performance is not only a function of its shape, i.e., its volumetric position, as it also depends on the type of errors present and whether those errors change with time or with number of qubits. A volumetric background inferred from a quantum volume involves particularly large extrapolations, as discussed in Section IV C. For this reason, we also include a ‘quantum volume region’ (the large grey square in Figure 1), alongside any volumetric background

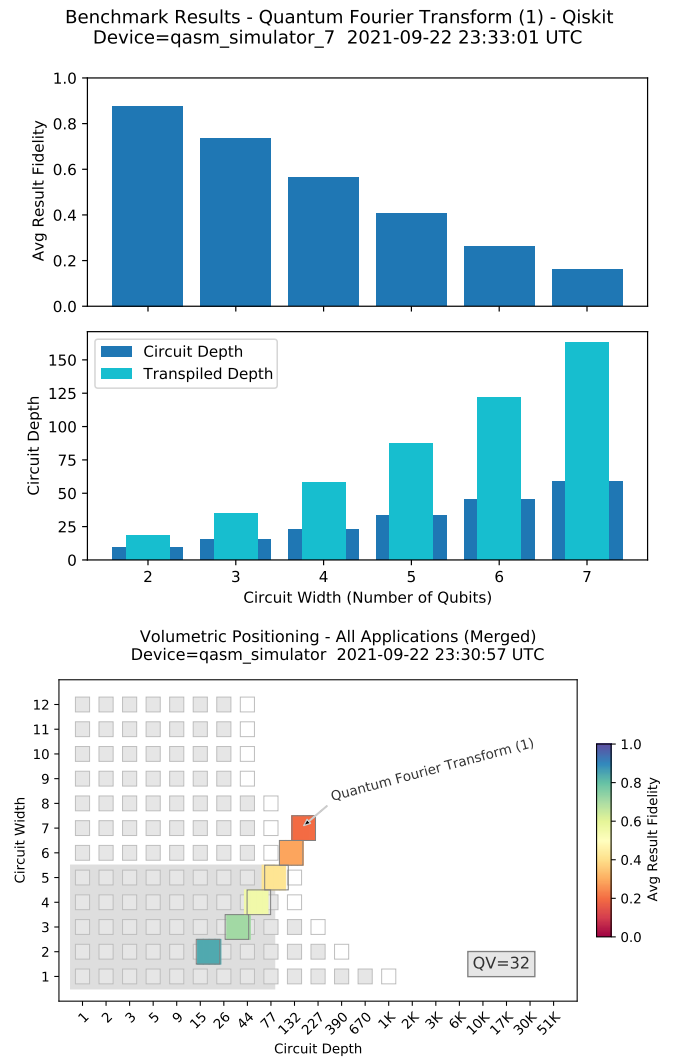


FIG. 6. **Demonstrating our methodology.** Demonstrating our methodology by executing our Quantum Fourier Transform (1) benchmark up to 7 qubits on a noisy quantum computer simulator. The upper plot shows the average result fidelity as a function of circuit width. The middle plot shows both the circuit’s normalized depth and the circuit’s depth when written in the high-level API used to define our benchmarks, as a function of circuit width. The lower plot presents these data on a volumetric background, in order to visualize the ‘footprint’ and result fidelity of the benchmark circuits.

extrapolated from a quantum volume. This quantum volume region encompasses all circuit shapes that are smaller than the largest successful quantum volume circuit, and so circuits within this region will have high result fidelity under weaker assumptions.

### C. Circuit Depth

We will primarily present our results in the volumetric space, meaning that a circuit’s shape, i.e., its depth and width, specifies where results from that circuit will appear in our plots.

Here we introduce and motivate the normalized definition for circuit depth that we use. A circuit's depth is the number of layers of logic gates it contains. However, any specific quantum computation can be implemented with many different circuits, perhaps defined over different gate sets, that have different depths. A quantum circuit described in a high-level API such as Qiskit can contain gates that are not directly implementable on a particular device. In order to run it, it must be converted, or transpiled, to the set of primitive quantum gates that the hardware actually executes. This transpiled circuit's depth — which we call the ‘physical depth’ — is often larger than the depth of the original circuit. For example, a single multiply-controlled NOT gate needs to be decomposed into many one- and two-qubit gates in order to run it on most quantum hardware.

A circuit's physical depth varies across devices (because, e.g., it depends on a device's connectivity). So, if we use the physical depth to define ‘depth’ in a volumetric plot, a circuit's position on the plot will be device-dependent. From some perspectives this device-dependent volumetric positioning might be desirable (see Section IV C). However it arguably complicates comparisons between devices. Therefore, in this work we chose to use a device-independent normalized definition of circuit depth, by defining a specific, device-independent basis (a gate set) to which the circuit is transpiled. The middle plot of Figure 6 shows an example of the depth of circuits from our Quantum Fourier Transform (1) benchmark when (1) written in a high-level API and (2) after the circuits have been transpiled to our standard basis, i.e., the normalized depth of the circuit. In this example, and in most of the benchmarks, the transpiled depth is larger than the circuit depth when written in a high-level API.

The basis that we selected consists of arbitrary single qubit rotations around the 3 Pauli axes, a CNOT gate, and a measurement gate. We assume that CNOT gates can be applied between any pair of qubits (i.e., all qubit pairs are connected), and gates on distinct qubits can be applied in parallel. This gate set, denoted by [‘rx’, ‘ry’, ‘rz’, ‘cx’] in Qiskit, is a widely used universal basis [32] that is hardware-agnostic. Note, however, that this (or any other) basis choice is unavoidably subjective. Each benchmark circuit is transpiled to this basis to compute a normalized circuit depth. When executed on hardware, the circuit is (necessarily) transpiled to the basis specific to that hardware, which will often include the addition of qubit SWAP gates so as to respect the topology of that hardware. Throughout this paper, ‘circuit depth’ always refers to our normalized definition for circuit depth unless otherwise stated.

#### D. Circuit Fidelity

One of the main performance metrics measured by our benchmarks is the quality of the result. In this work we quantify result quality using a normalized version of fidelity, that we now introduce. As with many metrics of result quality, the normalized fidelity is computed, for a circuit  $C$ , by comparing  $P_{\text{output}}(x)$  and  $P_{\text{ideal}}(x)$ , where  $P_{\text{output}}(x)$  is the output distribution over bit strings ( $x$ ) observed when running a cir-

cuit  $C$  on a particular device, and  $P_{\text{ideal}}(x)$  is the distribution that would be sampled from if  $C$  was run on a perfect quantum computer. There are many ways to quantify the difference between  $P_{\text{output}}(x)$  and  $P_{\text{ideal}}(x)$ . In this work we use a quantity that is based on the classical Hellinger distance [33] or the classical fidelity  $F_s$  [19, 34], defined by

$$F_s(P_{\text{ideal}}, P_{\text{output}}) = \left( \sum_x \sqrt{P_{\text{output}}(x)P_{\text{ideal}}(x)} \right)^2. \quad (1)$$

The reason that we do not use  $F_s$  to quantify result quality is that  $F_s$  is non-zero even when the output of the circuit is completely random, i.e., when  $P_{\text{output}} = P_{\text{uni}}$  where  $P_{\text{uni}}$  is the uniform distribution. This is particularly inconvenient for comparisons across different benchmarks, because the value of  $F_s(P_{\text{ideal}}, P_{\text{uni}})$  depends on  $P_{\text{ideal}}$  (it depends on the sparsity of  $P_{\text{ideal}}$ ). We therefore introduce a normalized fidelity defined by:

$$F(P_{\text{ideal}}, P_{\text{output}}) = \frac{F_s(P_{\text{ideal}}, P_{\text{output}}) - F_s(P_{\text{ideal}}, P_{\text{uni}})}{1 - F_s(P_{\text{ideal}}, P_{\text{uni}})}. \quad (2)$$

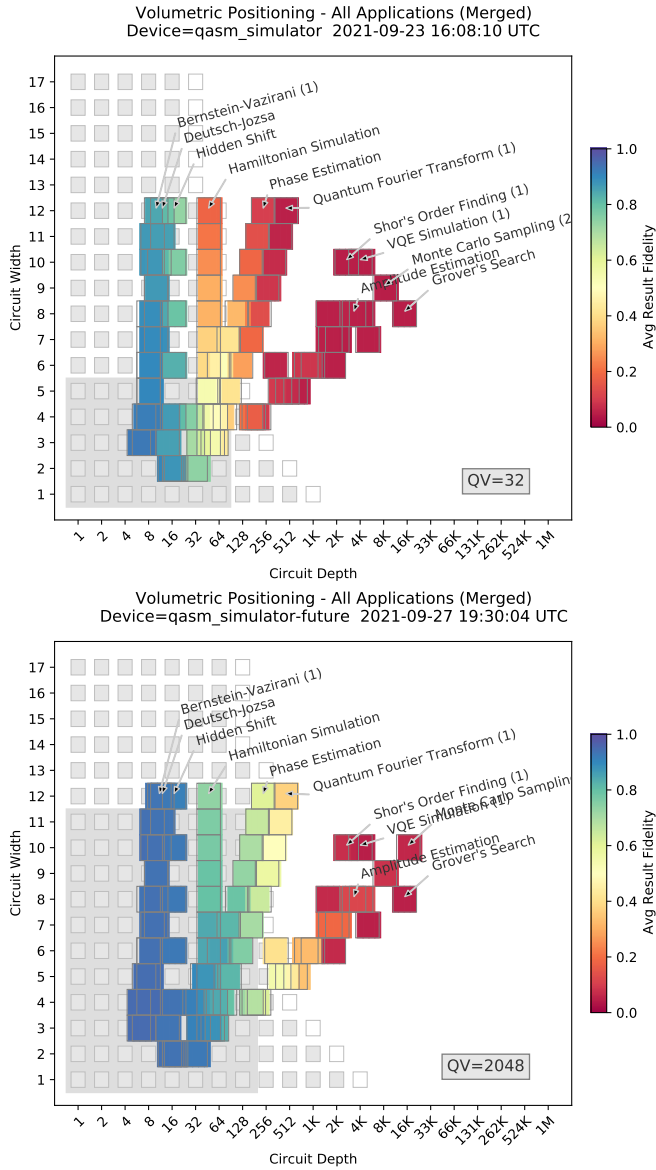
This normalized fidelity has the property that  $F(P_{\text{ideal}}, P_{\text{uni}}) = 0$ . Note that  $F$  generalizes the notion of ‘polarization’ introduced in [8], by extending it beyond the special case of definite outcome circuits. Throughout this work, ‘result fidelity’ refers to our normalized fidelity  $F$ . Typically, we average this fidelity over a circuit ensemble, which we denote by  $\bar{F}$ .

Figure 6 shows how we present the average result fidelity of a set of benchmark circuits, along with their widths and normalized depths, using the example of the Quantum Fourier Transform (1) benchmark run on simulation of a noisy quantum computer. The benchmark sweeps over a range of circuit widths and computes the average result fidelity for the circuits of that width, the average depth of the original circuits, and the average standard depth, i.e., the depth of the circuits after transpilation to our standard basis. This fidelity and depth data are shown, versus circuit width, in the upper two plots of Figure 6. The lower plot in Figure 6 shows the same data plotted on a volumetric background, which is the primary method that we use to visualize our results.

## IV. BENCHMARK RESULTS AND ANALYSIS

In this section we present results from executing our application oriented benchmarks on both a quantum simulator (Section IV A) and a selection of physical devices (Section IV B). Our benchmark suite is designed to execute nearly identically across multiple quantum programming environments, including Qiskit, Cirq, and Braket. All of the benchmarks have been implemented using the Qiskit API. Some, but not all, have been implemented using the Cirq and Braket APIs. In the sections below, all execution on a quantum simulator is done using the Qiskit version of our benchmarking suite, as it provides rich circuit analysis features. Execution on all of the hardware was also performed using the Qiskit version of our benchmarks, with the exception of the experiments on Rigetti Computing's hardware, which used the Braket version of our benchmarks.

## A. Benchmark Results on Quantum Simulator



**FIG. 7. Demonstrating our benchmarks on a quantum simulator.** Results from running our suite of application-oriented benchmarks on a quantum simulator (colored squares) on top of a volumetric background (grey-scale squares) extrapolated from the quantum volume. The simulator uses all-to-all qubit connectivity and two different error rate scales (details in main text) that result in quantum volumes of 32 and 2048, as shown in each plot. Each volumetric background is extrapolated from the quantum volume using a simple heuristic (see Section III B) that is designed to predict the regions in which a circuit's result fidelity will be above  $1/2$ .

Execution of the entire suite of benchmark programs results in a data file that stores all of the metrics collected for the tested device. Plotting all of the accumulated average result fidelity data on a single volumetric background produces charts like the ones shown in Figure 7, which were generated by running our benchmarks on the Qiskit Aer simulator. These simulations

used all-to-all connectivity between the qubits, and a simple error model in which each  $n$ -qubit gate was subject to  $n$ -qubit depolarizing errors, for  $n = 1, 2$ . For the simulation that generated the upper plot of Figure 7, the one- and two-qubit gate error rates were 0.003 and 0.03, respectively. These error rates are similar to those found in some current devices (although note that current devices suffer more complex, structured forms of error than pure depolarization). The quantum volume was determined to be 32, by executing the quantum volume protocol directly on this simulator, and this was used to generate the volumetric background (see Section III B). For the lower plot of Figure 7, the one- and two-qubit gate error rates were 0.0005 and 0.005, respectively, resulting in a quantum volume of 2048.

For both plots in Figure 7, there is close agreement between the average result fidelities of all the application benchmarks. There is also close agreement between the application benchmarks and the volumetric background. This is expected here due to the simplicity of the error model and the use of all-to-all connectivity. However, for more complex errors (e.g., coherent errors) the exact structure of a circuit can impact its performance, as errors can, e.g., coherently add or cancel. This means that circuits of the same shape can have very different performance, potentially resulting in significant differences between a device's performance on different applications. Similarly, if there is limited qubit connectivity, after a circuit is transpiled to a device's primitive gates, there can be an increase in the circuit depth that is large and strongly dependent on both the application and the device's connectivity. This can cause a sharper decrease in performance with increasing circuit width than anticipated by a volumetric background extrapolated from the quantum volume (using our simple heuristic), and this result fidelity decrease can be application dependent. We observe some of these effects in our experimental results.

## B. Benchmark Results on Quantum Hardware

In this section, we present the results of executing our application-oriented benchmarks on various providers' hardware. This section is not intended to perform or highlight direct comparisons between different systems, but rather to demonstrate our benchmarking suite. Quantum computing hardware is changing rapidly, and we did not have access to the latest hardware from all providers. Any comparisons would be quickly outdated, and therefore potentially misleading or erroneous. Here we present a few representative samples of results and highlight several conclusions that we can draw from these results.

### Rigetti Computing

One of the first quantum computers available to the public was provided by Rigetti Computing. Rigetti's devices are based on superconducting qubit technology, and they can be accessed using the Amazon Braket Service [21]. Here, we show results obtained from executing a subset of our benchmarks on Rigetti



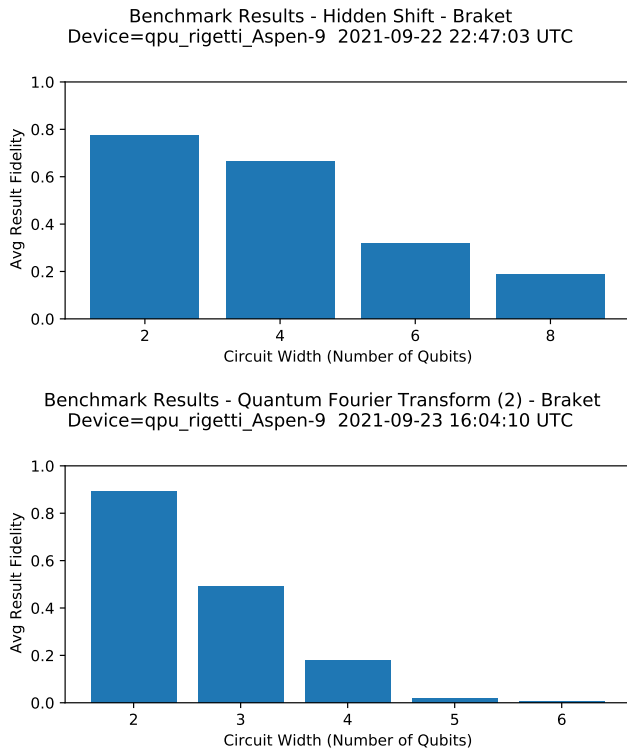


FIG. 8. **Demonstrating two benchmarks on Rigetti Aspen-9.** The result fidelity observed when running our Hidden Shift benchmark (upper plot) and Quantum Fourier Transform (2) benchmark (lower plot) on Rigetti Aspen-9. For each benchmark, we show the average result fidelity as a function of circuit width (the number of qubits).

Aspen-9, which is a 32-qubit machine [35], using the Amazon Braket version of our benchmarks.

Figure 8 shows the result fidelity as a function of the circuit width (the number of qubits) obtained when running the Hidden Shift and Quantum Fourier Transform (2) benchmarks on Rigetti Aspen-9. The Hidden Shift benchmark was executed on the first 8 qubits of Aspen-9. The Quantum Fourier (2) Transform benchmark was executed on only the first 6 qubits of Aspen-9, because, as shown in Figure 8, the result fidelity became negligible at around 5-6 qubits.

The result fidelity decreases as the circuit width increases, as expected. There are numerous factors that contribute to the precise form of the result fidelity decay that we observe, including the qubit connectivity. This is because, in general, these benchmarks use controlled rotation gates between non-connected qubits, which must be implemented using multiple two-qubit gates between pairs of connected qubits.

This is a fundamental consequence of limited qubit connectivity, but note that none of our benchmarks are tailored to utilize the subset of qubits in a device that have the best connectivity for the given application. Instead, the results represent what a user would experience when executing applications using the qubits 1 through  $n$ , where  $n$  is the circuit width. More sophisticated embeddings of our application benchmarks into a device's physical qubits are possible, will likely improve the performance we observe in devices with limited connectivity,

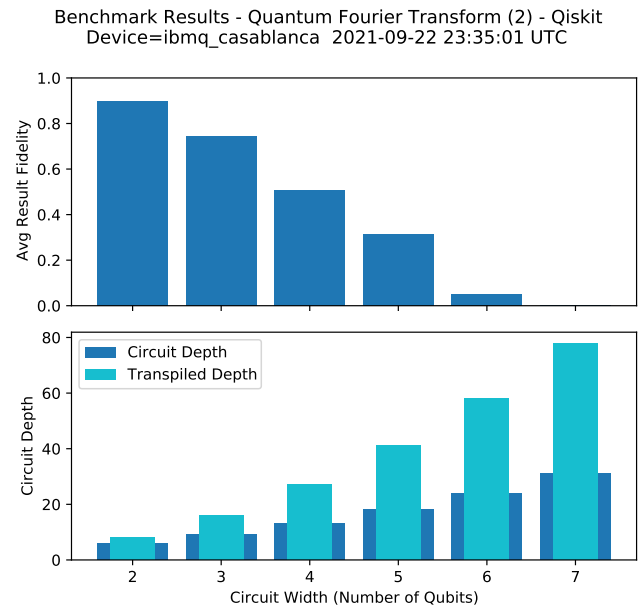


FIG. 9. **Demonstrating our QFT benchmark on IBM Q Casablanca.** Results from running the Quantum Fourier Transform (2) benchmark, the second of our two QFT-based benchmarks, on IBM Q Casablanca. The upper and lower plots show the average result fidelity and circuit depth, respectively, as a function of circuit width (the number of qubits).

and would be interesting future work.

### IBM Quantum

IBM Quantum provides a large selection of quantum computing devices through IBM Quantum Services [26]. IBM Q's devices are based on superconducting qubit technology. Below we present results from several of those machines. Note that there are larger and more recent IBM Q systems, including systems that have larger quantum volumes than those that we tested.

Figure 9 shows results from running the Quantum Fourier Transform (2) benchmark on IBM Q Casablanca, which is a 7 qubit device. We show the average result fidelity and average circuit depth as a function of the circuit width (the number of qubits). The lower plot of the figure shows both the normalized circuit depth and the depth of the circuits when written in a high-level API. Note that neither of these depths are the depth of the circuits after they have been transpiled to the device's native operations (i.e., the circuit's physical depth), which will typically be larger.

Figure 10 shows the results of running a subset of the benchmarking suite on IBM Q Casablanca and IBM Q Guadalupe. IBM Q Casablanca and IBM Q Guadalupe are 7 and 16 qubit devices, respectively, that both have a quantum volume of 32. Some of these benchmarks use circuits with a depth of well over 1000 (e.g., Grover's Search benchmark), and we observed low result fidelities for all such circuits. All application circuits whose volumetric positions are in the region

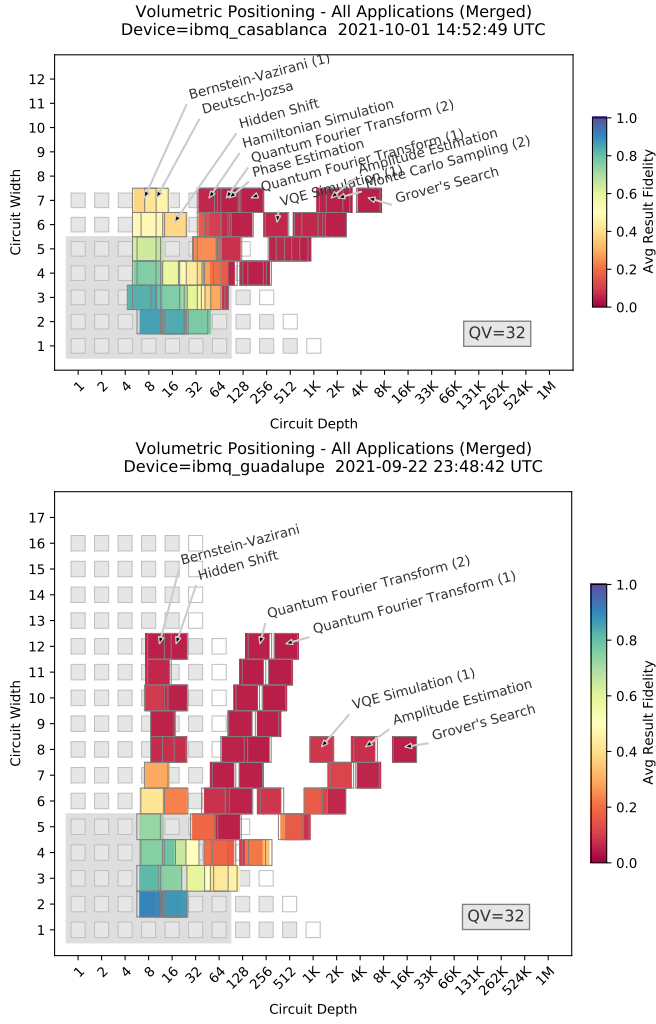


FIG. 10. **Benchmarking results on IBM Q Casablanca and Guadalupe.** The result fidelities obtained when running subsets of our benchmarking suite on IBM Q Casablanca and IBM Q Guadalupe. The volumetric backgrounds (grey squares) extrapolated from each device’s quantum volume (32 in both cases) do not accurately predict the performance of wide circuits on IBM Q Guadalupe.

predicted to have low result fidelity by the volumetric background (white squares and blank region) do have low result fidelities. However, there are application circuits with low result fidelities (e.g., red squares) whose volumetric positions fall within the high result fidelity region of the volumetric background (grey squares). We observe that performance drops off with increasing width much more quickly than predicted by the volumetric background. This is likely primarily because this volumetric background is extrapolated from the quantum volume using a simple heuristic that does not account for a device’s connectivity. This could potentially be addressed with a more sophisticated heuristic, or by instead using measured volumetric backgrounds (see Section IV C).

Mid-circuit measurement and qubit reset to make qubits reusable is a recent quantum computing enhancement, and

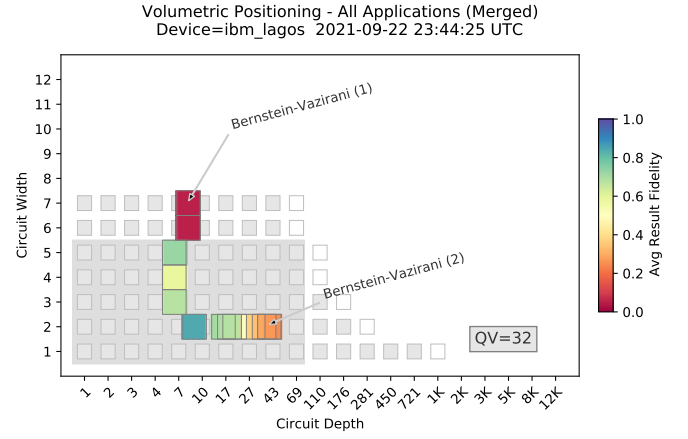


FIG. 11. **Benchmarking mid-circuit measurements.** Results from running our Bernstein-Vazirani (1) and Bernstein-Vazirani (2) benchmarks on IBM Q Lagos. These benchmarks are based on two different implementations of the Bernstein-Vazirani algorithm: version (1) uses  $n + 1$  qubit circuits for  $n$ -bit inputs, whereas version (2) uses mid-circuit measurements to reuse one of the qubits, and therefore needs only 2 qubits for any size input. Both benchmarks use the same random 2-6 bit algorithm inputs. We observe higher result fidelity when using mid-circuit measurements to reduce the width of the circuits.

IBM has introduced this feature for some of their systems. Our benchmark suite has been designed so that it can be adapted to include circuits that utilize mid-circuit measurements, as we now briefly demonstrate. As we review in Appendix B 1, the effect of this advance is that certain applications can be implemented with circuits over fewer qubits, as qubits can be reused. Figure 11 shows the results of executing two benchmarks, constructed from the Bernstein-Vazirani algorithm, on IBM Q Lagos, which is a 7 qubit device. In both versions of the benchmark the algorithm’s input is an  $n$  bit integer, and on IBM Q Lagos  $n$  ranges from 2 to 6. In the Bernstein-Vazirani (1) benchmark, an  $n$  bit integer input uses an  $n + 1$  qubit circuit, so the circuit widths range from 3 to 7. In the Bernstein-Vazirani (2) benchmark, an  $n$  bit integer input uses only 2 qubits for all  $n$ , one of which is repeatedly reset and reused with the aid of mid-circuit measurements. This results in width 2 circuits for all  $n$ . In Figure 11 we observe that the result fidelity drops precipitously when the width of the circuit exceeds 5, which places it outside the quantum volume region (the grey rectangle). In the Bernstein-Vazirani (2) benchmark, the circuits’ widths do not exceed 2, and we observe a larger result fidelity for the same problem instances. This illustrates the potential value of mid-circuit measurements for increasing an algorithm’s result fidelity.

### Quantinuum

Quantinuum System Model H1.1 was released commercially in 2020 [36]. This quantum computing system currently encodes up to 12 qubits in trapped atomic ions using the quantum charged coupled architecture [37]. The qubits are fully connected by using ion transport operations to rearrange the ions

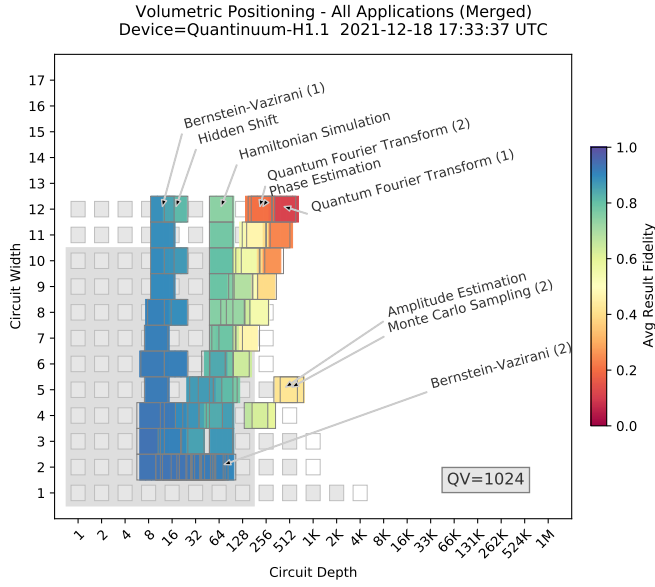


FIG. 12. **Benchmark results on Quantinuum’s H1.1.** The result fidelities obtained when running our benchmarking suite on H1.1, which is a 12-qubit device. Each benchmark was run with 500 shots and at most five random circuits per qubit number. For all wide and shallow circuits the result fidelity is high. Quantinuum has measured the quantum volume of this device to be 1024 [40], and this value is used to construct the volumetric background region. The volumetric background is broadly predictive of the performance of our algorithmic benchmarks.

to interact any pair of qubits. The system has an average single-qubit gate fidelity of 99.99(1)% and average two-qubit gate fidelity of 99.70(5)%, as measured using standard randomized benchmarking [38]. The system is also capable of applying high-fidelity mid-circuit measurement and reset to any qubit [39].

Figure 12 shows the results of executing a subset of our application-oriented benchmarks on the Quantinuum H1.1. For these tests the code was slightly edited to more easily run with the Quantinuum API but the qiskit version is also supported. These results show that the Quantinuum system can run 12-qubit, moderate-depth circuits that contain interactions between many pairs of qubits and still obtain moderate to high fidelity results. Furthermore, these results are broadly consistent with the component-level fidelities measured by one- and two-qubit randomized benchmarking. The Bernstein-Vazirani (1), Hidden-Shift, and Hamiltonian Simulation benchmarks all return high-fidelity results ( $\bar{F} \geq 0.77$ ) for all qubit numbers up to the 12 qubit system size limit. Benchmarks that require deeper circuits, such as the Quantum Fourier Transform (1) and (2) benchmarks and the Phase Estimation benchmark, still produce results that are distinguishable from random outputs. The smallest fidelity for the QFT (1) was  $\bar{F} = 0.07$ , which was obtained for the case of  $n = 12$ . Overall, we observe that the volumetric background extrapolated from a quantum volume of 1024 [40] provides a broadly, but not entirely, accurate prediction of the performance of our algorithmic benchmarks.

The H1.1 can implement mid-circuit measurement and re-

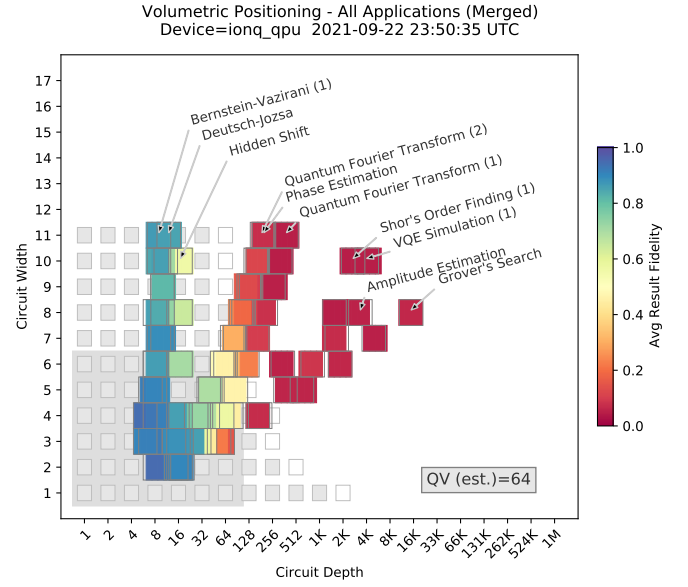


FIG. 13. **Benchmark results on IonQ’s cloud-accessible system.** The result fidelities obtained when running our benchmarking suite on IonQ’s current cloud-accessible system, which features 11 qubits. For all wide and shallow circuits the result fidelity is high. IonQ does not publish a quantum volume for this device, and we did not measure it directly. We use a quantum volume of 64 to generate the volumetric background, as this value for the quantum volume is broadly consistent with the results of our benchmarks.

set operations, which we benchmarked using the Bernstein-Vazirani (2) benchmark. We see that while both versions of our Bernstein-Vazirani benchmark achieve high fidelity results ( $\bar{F} > 0.929$ ), version (2) performs slightly better for larger register size. The Bernstein-Vazirani (2) benchmark can also be implemented for larger algorithm input sizes ( $n > 12$ ) as it only uses two qubits for any input, although we did not do this in our experiments, whereas the Bernstein-Vazirani (1) benchmark is limited to  $n \leq 12$  due to the machine’s current restriction to 12 qubits.

### *IonQ*

IonQ is a provider of quantum computers that are available via Amazon Braket [24], Microsoft Azure Quantum [22] and Google Cloud [41]. IonQ systems support most leading quantum programming environments, and we executed our benchmarks on the IonQ system using the Qiskit provider module supplied by IonQ. Their hardware utilizes trapped ytterbium ions where qubits are encoded in two states of the ground hyperfine manifold. The IonQ system currently on the cloud consists of 11 qubits that are fully connected.

Figure 13 shows results from executing a subset of the application-oriented benchmarks on the IonQ system. IonQ does not publish a quantum volume metric for this device, and we chose not to run the quantum volume circuits on any of the tested hardware. In Figure 13 we use a volumetric background extrapolated from a quantum volume of 64. We choose this

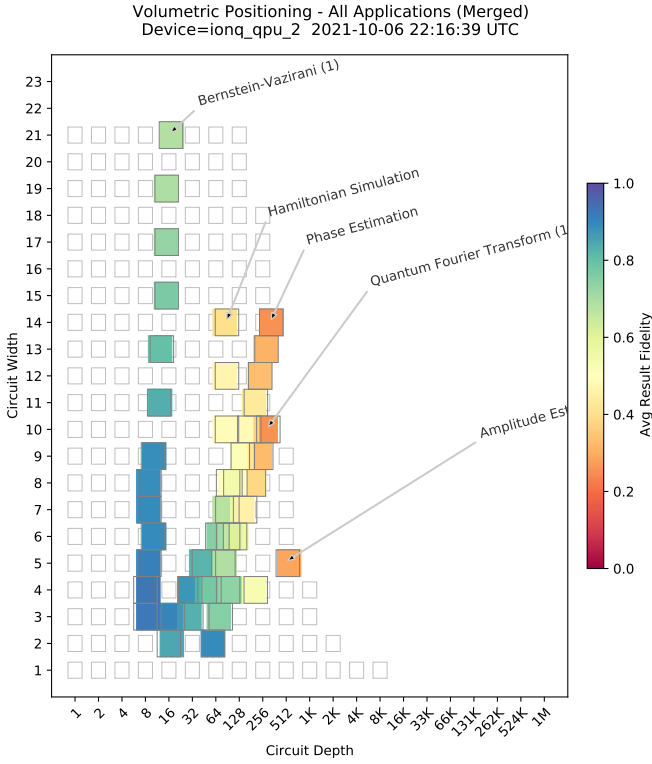


FIG. 14. **Benchmark results on IonQ’s latest system.** The result fidelities obtained when running our benchmarking suite on up to 21 qubits of IonQ’s latest system. We observe a significant performance improvement in comparison to the previous generation of IonQ hardware. IonQ does not publish a quantum volume for this device, and we did not measure it directly.

value as it maximizes the agreement between the results of the benchmarks and that volumetric background, and as such it is the most reasonable approximation to the device’s true quantum volume given this data. In Figure 13 we observe high-fidelity results for the Bernstein-Vazirani (1), Deutsch-Jozsa and Hidden Shift benchmarks for all qubit numbers, up to the current system limit of  $n = 11$ . For benchmarks that use deeper circuits we observe a steeper decrease in result fidelity with  $n$ , and the observed performance is broadly, but not entirely, consistent with the predictions of the volumetric background.

IonQ has recently introduced its next-generation quantum hardware [42]. This hardware is purported to feature an order of magnitude reduction in gate error rates in comparison to IonQ’s current system on the cloud. Figure 14 shows the results from running a subset of our benchmarking suite on this device, including benchmarks from all 3 categories in Table I. We observe a significant increase in performance in comparison to the previous generation hardware, as is apparent by comparing Figs. 13 and 14. For instance, the Bernstein-Vazirani (1) algorithm is run for up to 21 qubits, and the result fidelity for the largest circuit is 70%. This can be compared with the result fidelity for the largest circuit for the Bernstein Vazirani (1) benchmark on the cloud quantum computer, which is 78% for 11 qubits. Note that the maximum width and depth of the Hamiltonian Simulation and Phase Estimation bench-

mark circuits here exceeds that in all other IonQ experiments, as here we tested up to 14 qubits.

### Compatible Hardware

Our application-oriented benchmarking suite is designed to be applicable to any gate-based quantum computer, and our implementation of that suite is applicable to any platform that can be interfaced with Qiskit, Cirq, Braket, or Q#. Table II contains a non-exhaustive list of hardware that that we did not test in this work but that is compatible with our implementation of our benchmarking suite. Note that support for different quantum hardware devices by these programming environments will evolve over time.

Hardware	Programming Environment
Google QCS	Cirq
Alpine Quantum Technologies	Qiskit
	Cirq

TABLE II. The Application-Oriented Benchmark Suite introduced in this work can be executed on other quantum computing hardware, using the programming environments specified.

### C. Comparison to Generic Benchmarks

In this work we have compared results from our application-oriented benchmarks with a volumetric background. As introduced in Section III B, a volumetric background is a prediction for the result fidelity of circuits, as a function of circuit shape (i.e., volumetric position), that is constructed from the results of some independent benchmark or performance data. So far in this paper, we have considered only volumetric backgrounds that are extrapolated from a single performance metric: the quantum volume.

Our simulated and experimental results show that this extrapolated volumetric background can provide reasonably accurate predictions of application performance (particularly with all-to-all connectivity devices), but this is not always the case. There are many reasons why we should not expect our volumetric background extrapolated from the quantum volume to always accurately predict performance, including: (1) the extrapolation uses a very simple heuristic that does not account for device connectivity (see Section III B), and (2) we are typically extrapolating the quantum volume to predict the performance of circuits containing significantly more qubits than were used in any of the quantum volume circuits.

This highlights an interesting research direction: developing methods for constructing volumetric backgrounds that better predict the performance of these (and other) application-oriented benchmarks. To illustrate this idea and highlight some challenges, below we briefly explore one possibility: creating a volumetric background by running ‘randomized mirror



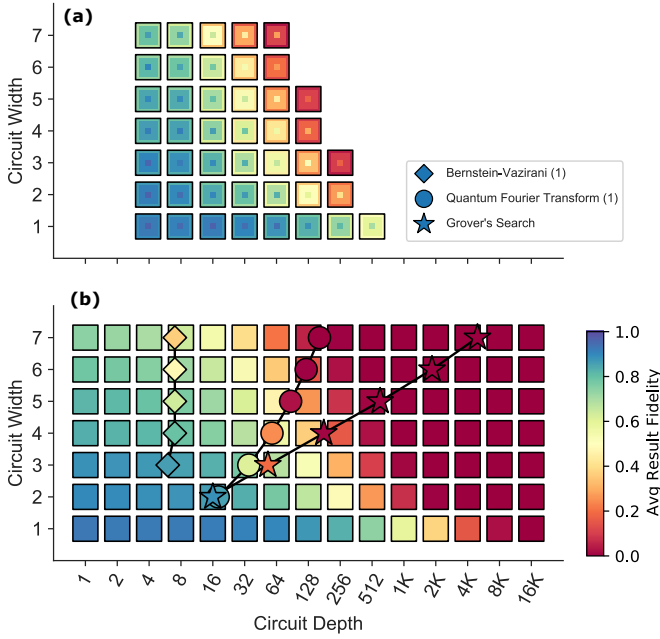


FIG. 15. **Comparing mirror circuit and application-oriented benchmarks.** (a) Results from running randomized mirror circuits [8] on IBM Q Casablanca. At each circuit shape 40 circuits were sampled and run. The maximum, average, and minimum result fidelity that was observed at each circuit shape is shown, in concentric squares. (b) The average result fidelity of the mirror circuits is extrapolated to arbitrary depths to create a volumetric background (squares), and compared to the results from running a subset of the application benchmarks on IBM Q Casablanca (diamonds, circles and stars). We observe good correlation between the different benchmarks at low width, but an increasing discrepancy between the volumetric background and the applications as the circuit width increases. This is likely primarily due to the different frequency with which gates between non-connected qubits are used in the four different benchmarks. Mirror circuit benchmarks that mimic the frequency with which algorithms require gates between non-connected qubits might reduce this discrepancy, but this is not explored herein.

circuits’ [8]. For this illustrative comparison, we focus on a single device, IBM Q Casablanca, and the particular class of randomized mirror circuits used in [8]. Figure 15 (a) shows the result fidelities obtained when running randomized mirror circuits on IBM Q Casablanca.

We use the data in Figure 15 (a) to construct a volumetric background by (1) using the average result fidelity for randomized mirror circuits of a given shape as our predictor for the performance of other circuits of this shape, and (2) extrapolating this to all circuit depths using the theory for randomized mirror circuits presented in [43]. This volumetric background is shown in Figure 15, alongside the average result fidelities of three of our application benchmarks: the Bernstein-Vazirani (1) benchmark, the Quantum Fourier Transform (1) benchmark, and the Grover’s Search benchmark. To facilitate a more precise comparison to the results of the application-oriented benchmarks, here we (1) do not average together the results for application-oriented benchmarks with similar volumetric positions, and (2) we use a color-scale volumetric background.

The general trend of all three benchmarks and the volumetric background are unsurprisingly in agreement: result fidelity decreases with increasing circuit width and depth. However, there are substantial differences between the rates at which the performance of each type of circuit drops off with increasing circuit size.

There are multiple possible reasons for this discrepancy. Perhaps the most important is again related to device connectivity. The randomized mirror circuits that we ran contain two-qubit gates only between connected qubits. In contrast, the application-oriented benchmarks can contain two-qubit gates between any pair of qubits. Because depth is defined with respect to an all-to-all connectivity gate set (see Section III C), there can be a large discrepancy between the physical depth of the randomized mirror circuits and the physical depth of the application circuits, at a given normalized circuit depth. This discrepancy will typically increase with the number of qubits (i.e., circuit width), and so this effect could explain the faster result fidelity decrease with circuit width that we observe for the application benchmarks, in comparison to the predictions from the randomized mirror circuit data.

The discrepancies between the results of our application-oriented benchmarks and both the volumetric backgrounds inferred from randomized mirror circuit data and from a device’s quantum volume are indicative of the challenges associated with predicting algorithm performance from generic benchmarks. One way to potentially remove the discrepancy we observe in Figure 15 is to use physical depth to define a circuit’s volumetric position, which would mean that an application’s volumetric ‘footprint’ will be device-dependent (something we choose to avoid in this work). We leave an investigation of this to future work. Alternatively, application-agnostic benchmarks that are designed to reflect the qubit connectivity implicitly used by a particular application could be developed (e.g., this is possible within the framework of randomized mirror circuits). However, developing new application-agnostic benchmarks is also beyond the scope of this work so we do not pursue this further here.

## V. MEASURING EXECUTION TIME

Most existing quantum computing benchmarks focus on measuring result quality, but the time taken to run an application is also an important metric of performance [44–47]. In particular, the time to solution is a crucial metric when comparing quantum and classical computations, e.g., comparing quantum and classical compute times was central to recent demonstrations of quantum advantage [2, 3]. Our benchmarking suite is designed to quantify various aspects of the time to solution, as we explain below. In Section V A we discuss how to measure and subdivide the total execution time. In Section V B we then turn to the component of execution time that our benchmarking suite currently focuses on: the time taken to execute the quantum circuits.



### A. Total Execution Time

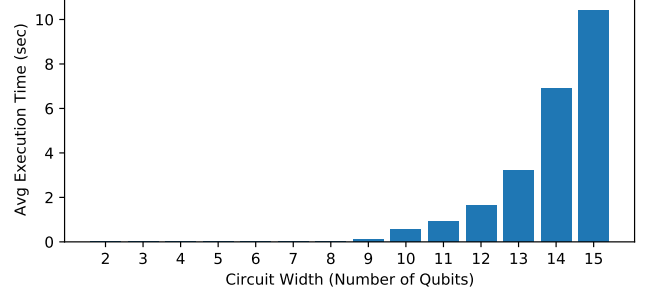
The total time required to execute a complete quantum application is evidently a critical aspect of a quantum computer’s performance. In order to understand a quantum computer’s performance in detail, the total execution time should ideally be divided into the times taken for distinct sub-tasks (e.g., circuit compilation), and an ideal benchmarking suite would measure each such time. In general, a quantum application is a hybrid quantum-classical computation consisting of running one or more quantum circuits interleaved with classical processing. We therefore suggest that the following non-exhaustive set of sub-tasks and associated compute times will be important to measure:

1. Compilation time ( $t_{\text{compile}}$ ) — the time spent compiling an application’s circuits, written in a high-level language (e.g., Q# or Qiskit), into the hardware’s native operations and language (e.g., OpenQASM, or perhaps the pulse-level description of the circuit).
2. Classical computation time ( $t_{\text{classical}}$ ) — the time spent implementing classical computations to, e.g., process data from quantum circuits already run (as in variational algorithms [48–50]).
3. Quantum execution time ( $t_{\text{quantum}}(N)$ ) — the time required for  $N$  shots of a circuit. This is the only time that we report on in this work.

An additional component in the total execution time when using shared-usage cloud-access quantum hardware is the queuing time, i.e., the time that circuits spend in a queue waiting to be executed. This is an artifact of shared-usage access models, and arguably its main importance here is that it can complicate reliable measurements of other aspects of the total execution time.

To illustrate the potential utility of measuring different components of total execution time, we consider one aspect of running variational quantum algorithms. These algorithms involve repeated execution of a parameterized circuit for different parameter values. There are two distinct ways in which these circuits, that differ only in the parameter values, can be implemented. Each such circuit can be recompiled from a high-level language description (incurring the  $t_{\text{compile}}$  cost each time), or a single parameterized circuit can be compiled with its parameters set and updated using low-level classical controls. These two methods for running parameterized circuits will result in different compute times, and avoiding recompilation could potentially result in substantial compute time savings. Our Variational Quantum Eigensolver benchmark has been designed to enable this comparison, by looping over multiple instances of a circuit with differing parameter values. Furthermore, we suggest that benchmarks that can reveal and quantify other potential improvements in compute time will be important tools for testing and improving quantum computers.

Benchmark Results - Quantum Fourier Transform (1) - Qiskit  
Device=qasm\_simulator\_15 2021-09-22 22:40:27 UTC



Benchmark Results - Quantum Fourier Transform (1) - Qiskit  
Device=ibmq\_guadalupe\_15 2021-09-22 22:40:30 UTC

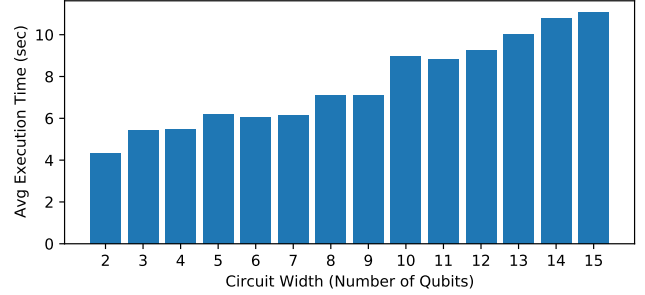


FIG. 16. **Measuring quantum execution time.** The average quantum execution time observed when running the Quantum Fourier Transform (1) benchmark up to 15 qubits, with  $N = 1000$  shots, on a Qiskit Aer simulation of a noisy quantum computer (upper plot) and IBM Q Guadalupe (lower plot).

### B. Quantum Execution Time

The aspect of execution time that our current benchmarking suite focuses on is quantum execution time  $t_{\text{quantum}}(N)$ . This is the time taken to run  $N$  ‘shots’ of a quantum circuit (as the results from quantum circuits are probabilistic in nature, they are each typically run many times, with an individual circuit execution referred to as a ‘shot’). The quantum execution time depends on both the circuit (e.g., it will increase with the circuit depth), and on the number of shots  $N$ . In our experiments we use  $N = 1000$ , and we do not study the dependence of  $t_{\text{quantum}}$  on  $N$ . Note that is important to use the same value of  $N$  when comparing different devices.

The quantum execution time is not directly accessible to a quantum computer user, but cloud quantum computers typically return a quantity that is analogous to  $t_{\text{quantum}}(N)$ , following the execution of the quantum circuit. This time is recorded by our benchmarking suite, and it is what we report. Note, however, that the precise meaning of the timing information returned by a provider’s API is not always clear to the end user (and is possibly different for different providers). This should be kept in mind when comparing quantum execution time results between platforms.

Figure 16 shows the average time to execute the circuits of the Quantum Fourier Transform (1) benchmark, as a func-

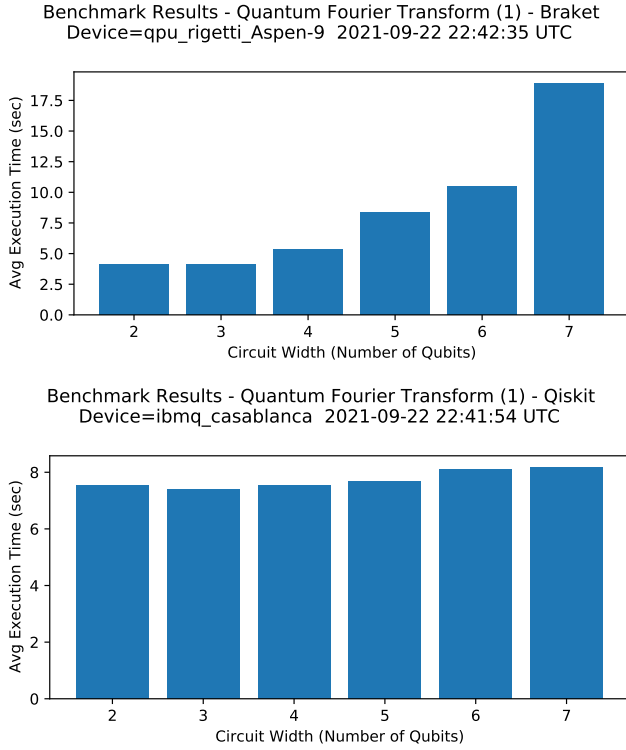


FIG. 17. **System differences in quantum execution time.** We observe significant differences in the quantum execution time between different systems. This example shows the average quantum execution time observed when running the Quantum Fourier Transform (1) benchmark on Rigetti Aspen-9 (upper plot) and IBM Q Casablanca (lower plot). Note the different minimum execution times and the different rates of increase with circuit width.

tion of circuit width, for a quantum simulator and for IBM Q Guadalupe. The execution time increases exponentially on the simulator, and linearly on this quantum hardware. Note that the exponential growth in the time to simulate these circuits does not imply that these circuits are classically hard to simulate [51], only that this classical simulator exhibits exponential simulation time scaling on these circuits.

Figure 17 shows results from running the same Quantum Fourier Transform (1) benchmark on Rigetti Aspen-9 and IBM Q Casablanca. The average execution time is shown as a function of the circuit width (the number of qubits) up to 7 qubits. On IBM Q Casablanca, the execution time is approximately independent of circuit width, suggesting a minimal execution time of  $\sim 7$  seconds for any circuit. The increase in execution time is small as circuit width and depth increase (although note that this trend should not be naively extrapolated to larger widths and depths). In contrast, on Rigetti Aspen-9 the minimum execution time is about 4 seconds and there is a steep increase in execution time as circuit width increases (we conjecture that this is because all gates are serialized on Aspen-9, meaning that gates on distinct qubits that could be applied in parallel are applied in serial). We found that there is significant variation in the execution time profile across the many different devices that we tested.

## VI. HIGHLIGHTS AND IMPRESSIONS

A set of applications for a quantum computer does not itself constitute a benchmarking suite. In this work, we have converted applications into benchmarks, and in doing so we have made a variety of choices with associated strengths and weaknesses that we discuss in this section. In particular:

- We quantify result quality with average result fidelity. This enables application-independent data analysis, but the average result fidelity is not always a natural measure of an application’s performance.
- Our benchmarks closely correspond to the applications from which they are constructed — where possible, they consist of simply running that application’s circuits. Therefore, some (but not all) of our benchmarks require classical simulations that are exponentially expensive in the number of qubits.
- Our benchmarks test a quantum computer’s ability to implement algorithms for particular classes of input problem. The chosen input problems are not guaranteed to be representative of real-world use-cases.
- Our implementation of these benchmarks is designed to be compatible with most widely-used quantum computing APIs. However, differences between these APIs limit our ability to achieve this.

Below, we expand on these points, discuss how they impact our results, and propose pathways to improving our benchmarks. Technical details that are relevant only to individual benchmarks within our suite are largely relegated to Appendix A.

### *Limitations of the Average Result Fidelity*

In this work we chose to analyze the output of our benchmarks using the result fidelity, defined in Eq. (2). For each circuit that we run, we calculate its result fidelity — which is a measure of the closeness of  $P_{\text{output}}$  to  $P_{\text{ideal}}$  — and then we average the result fidelities for all the circuits of a fixed input size  $n$ . This is not the only possible metric for result ‘quality’. We could use an alternative measure of the closeness of  $P_{\text{output}}$  to  $P_{\text{ideal}}$ . Furthermore, we do not need to average our chosen closeness measure over all the results for each  $n$ , e.g., we could look at worst-case performance. Any single choice will have limitations that must be taken into account when interpreting a benchmark’s result.

To illustrate the limitations of both fidelity and averaging, we use one of our benchmarks as an example: the Deutsch-Jozsa benchmark. The Deutsch-Jozsa algorithm is an  $n + 1$  qubit oracle algorithm that computes whether  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is a balanced or constant function using a single call to an oracle that implements  $f$ . If the function is constant the first  $n$  qubits should all return 0, and otherwise at least one of the qubits should return 1. In our benchmark we randomly choose the oracle: with 50% probability  $f$  is a balanced function, and with

50% probability  $f$  is constant. The balanced function that we use is the parity function, in which case all the qubits should return 1. Now consider two quantum computers,  $Q_A$  and  $Q_B$ , that always return the bit strings 0000... and 1000..., respectively. In both cases, if we run the Deutsch-Jozsa algorithm we always get the correct *algorithm* output for one of the two inputs, and the incorrect output for the other, i.e., arguably  $Q_A$  and  $Q_B$  perform equally badly on this algorithm. But we find that  $\bar{F}_s = 0.5$  for  $Q_A$  and  $\bar{F}_s = 0$  for  $Q_B$ , where  $\bar{F}_s$  is  $F_s$  averaged over the balanced and constant oracle. This is because for  $Q_B$ ,  $F_s(P_{\text{output}}, P_{\text{ideal}}) = 0$  for both inputs (as  $P_{\text{ideal}}(111\dots) = 1$  for the balanced function) [52].

This example illustrates two points. First, the fidelity between  $P_{\text{ideal}}$  and  $P_{\text{output}}$  for a particular algorithm input does not directly correspond to the probability of obtaining the correct result on that input. In this example,  $Q_B$  always returns the correct algorithm result on the balanced function input, even though  $F_s(P_{\text{ideal}}, P_{\text{output}}) = 0$ . To avoid this effect we would need an algorithm-specific metric for result quality. We suggest that the result fidelity should be complemented (but not replaced) with such metrics. Second, quantifying performance using only an average over problem instances can obscure important performance information. We suggest that it will be important to also study, e.g., worst-case performance over problem instances.

Similar effects to those discussed above were apparent in our experiments. Figure 18 shows the results of running the Deutsch-Jozsa benchmark on (a) a quantum simulator with depolarizing errors, and (b) IBM Q Guadalupe. In the simulation we observe that the average result fidelity decays towards zero as the number of qubits increases (note that our normalized result fidelity  $F$  is designed so that it decays towards zero in the presence of depolarizing errors, unlike  $F_s$ ). In contrast, the average result fidelity for IBM Q Guadalupe decays quickly to  $\sim 0.5$  with increased circuit width, and then decays slowly. The constant oracle uses a shallow  $O(1)$  depth circuit containing no two-qubit gates, and its correct result is 000.... In contrast, the balanced oracle uses  $O(n)$  two-qubit gates and its correct result is 111.... The constant oracle can therefore be expected to perform much better than the balanced oracle on IBM Q Guadalupe, which has high-fidelity one-qubit gates [26] and which (like all superconducting systems) experiences substantial T1 decay that adds a bias towards returning the 000... bit string. This strong asymmetry in the performance on the constant and balanced oracles would explain the results of Fig. 18, and it illustrates that average result fidelities must be interpreted and compared with care.

#### Impact of Compiler Optimization Techniques

Implementation of these benchmarks on hardware systems (results shown in Section IV B) is intentionally done using default settings for the transpilation and execution of the circuits. This is how most users work with any programming system, even a quantum one. Many of the quantum computing programming packages offer compilation options that will modify the circuits before execution to account for device characteristics such as qubit connectivity or timing parameters. While

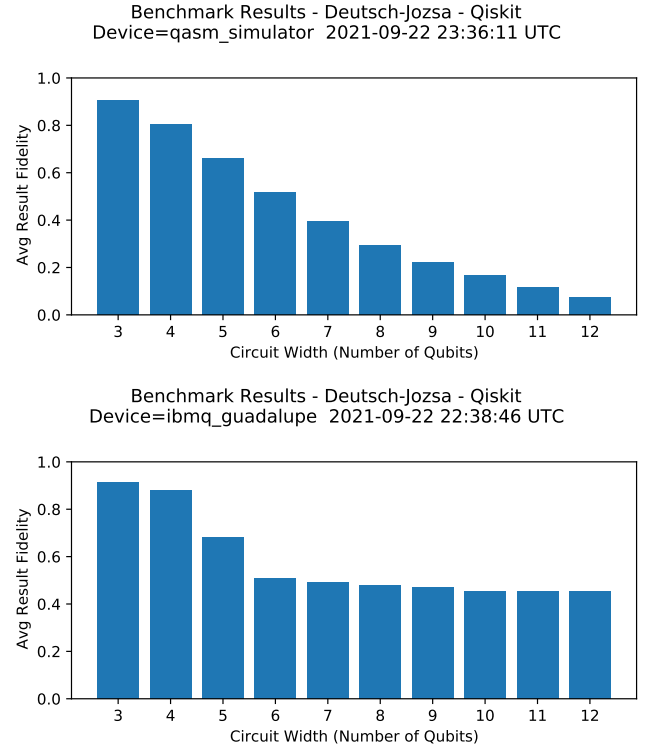


FIG. 18. **Demonstrating the limitations of average result fidelity.** Results from running the Deutsch-Jozsa benchmark on a simulator (upper plot) with depolarizing errors and IBM Q Guadalupe (lower plot). Unlike in the simulation, the average result fidelity on IBM Q Guadalupe decays quickly to  $\sim 0.5$  with increasing circuit width but then decays slowly. This suggests a strong asymmetry in the result fidelities for the two problem instances — the balanced and unbalanced function oracles — and, as discussed in the main text, this asymmetry is consistent with known noise mechanisms in IBM Q Guadalupe.

using these parameters can result in better performance, it is often difficult for users to know which parameters should be used or what values to choose for each target device. Providers also may elect to deliberately exclude automatic optimization techniques for fear of interfering with some users' custom optimizations. The result can be that users fail to obtain the best possible performance from a vendor's hardware by default.

Figure 19 illustrates the impact of a simple compiler optimization and layout strategy as applied to execution on the IBM Q Guadalupe device. Note that the fidelity obtained for circuits that are wide and shallow is significantly improved using this strategy because it optimizes the layout of the circuit for the qubit connectivity of the device. In this case, the Qiskit execution method was configured to use `optimization_level = 3`, which provides the highest circuit optimization but at the expense of longer execution time of the classical circuit optimizer [19]. Additionally, the execution was configured to use `layout_method = 'sabre'` and `routing_method = 'sabre'`, which apply information about the connectivity of the target device to layout the circuit and route interactions between qubits optimally.

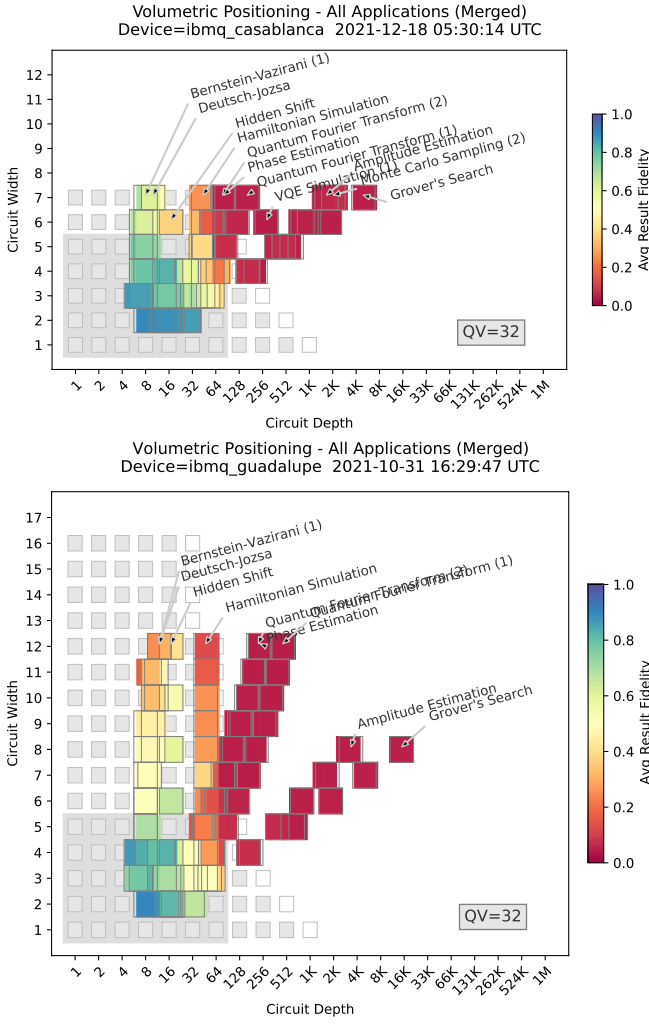


FIG. 19. **Optimized Benchmarking results on IBM Q Casablanca and Guadalupe.** The result fidelities obtained when running a subset of our benchmarking suite on the IBM Q Casablanca and Guadalupe systems using the compiler optimization strategy defined in the text. There is a noticeable improvement in fidelity results for wide circuits, as compared with the results shown in Figure 10.

On the H1.1 system we experimented with using arbitrary angle two-qubit gates to implement the QFT (1) and (2) tests to increase the fidelity as shown in Fig. ?? . In trapped-ion systems, two-qubit gates are typically implemented with the Mølmer-Sørensen interaction. In most cases, the resulting gate is produced with a fixed laser power that ideally results in the unitary  $\exp[-i(\pi/4)XX]$ , which is equivalent to CNOT up to local unitaries. A controlled- $Z(\theta)$  operation, which appears in many of the application oriented benchmarks, requires two CNOTs. The Mølmer-Sørensen gate can also be implemented with an arbitrary angle  $\exp[-i(\phi/2)XX]$  by varying the laser power as a function of  $\phi$ . The resulting gate is equivalent to a controlled- $Z(\theta)$  gate up to local unitaries. This modification halves the number of two-qubit interactions required to implement controlled- $Z(\theta)$  (for most  $\theta$ ) and may further decrease two-qubit errors that scale with  $\phi$ . From the QFT (1) and (2)

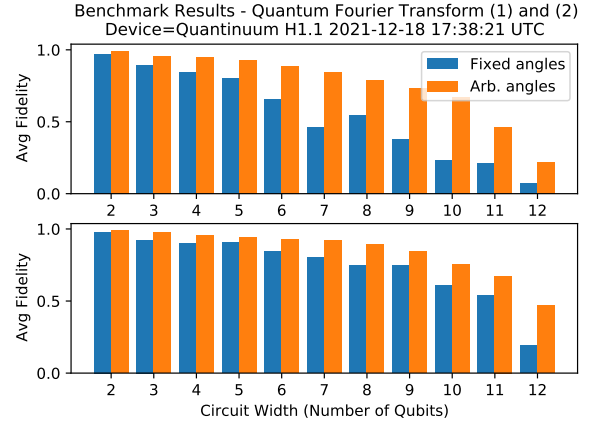


FIG. 20. **Optimized Benchmarking results on Quantinuum's H1.1.** The result fidelities obtained when running a subset of our benchmarking suite on H1.1 using arbitrary angles. For both QFT benchmarks the number of two-qubit interactions is roughly cut in half and the errors are further reduced by decreasing the angle of two-qubit rotations.

benchmarks plotted in Fig. 20, we can verify that the arbitrary angle gates are yielding a significant improvement. For example, for QFT (1) without arbitrary angle gates the fidelity for  $n = 12$  qubits was 0.07 but with the arbitrary angle gates it had a  $3\times$  improvement to 0.216.

#### Exponential Cost to Evaluate the Result Fidelity

To calculate the result fidelity for a circuit we first compute  $P_{\text{ideal}}$  on a classical computer. Our benchmarks are built on quantum circuit families with a variable number of qubits  $n$ , and in general computing  $P_{\text{ideal}}$  can be exponentially expensive in the number of qubits  $n$ . This is the case for our Hamiltonian Simulation and Variational Quantum Eigensolver benchmarks (but not for, e.g., our Quantum Fourier Transform benchmarks). As currently constructed, these benchmarks are therefore only feasible to run for small  $n$ . Although our existing benchmarks scale to sufficiently many qubits for the purposes of our experiments, in the long-term it will be essential to adapt our benchmarks so that they scale to hundreds or thousands of qubits.

There are a variety of existing methodologies that could be adopted to make our benchmarks scalable. The benchmarks could be altered so that they use only problem instances for which the solution can be efficiently computed. However, it is critical that a quantum computer's performance on these benchmarks is indicative of its performance on practically useful problems, and so benchmarks based on this idea must be designed carefully. Alternatively, there exist general-purpose tools converting circuits into similar circuits for which  $P_{\text{ideal}}$  is efficient to compute [8, 53], and methodologies like these could be used to adapt our benchmarks so that they can be applied to hundreds or thousands of qubits.

#### Benchmarks with Representative Performance

A quantum computer’s performance on an application-based benchmark will ideally reflect its ability to run that application. However, a benchmark built from an application is not guaranteed to have this property. For example, a benchmark might only test a quantum computer’s performance on an algorithm for a subset of problem instances that (1) are not representative of potential use-cases for the algorithm, and (2) are particularly easy to implement with high fidelity, e.g., because they require only very low depth circuits.

Assessing whether a benchmark is representative of an algorithm’s potential use-cases is particularly challenging for an algorithm that is necessarily a subroutine, i.e., it can only perform a classically intractable computation when it is part of a large quantum computation. For example, the quantum Fourier transform cannot alone solve any useful problems (when acting on product states and followed by a measurement of the qubits it can be efficiently simulated on classical computer), but it is an important component in multiple applications, such as Shor’s algorithm. Arguably, a benchmark built from a subroutine, like the quantum Fourier transform, should measure how well it will perform as a subroutine within some larger circuit [54]. This requires quantifying how well the algorithm performs on entangled input states. Our benchmarks are not currently designed to accurately measure the performance of a subroutine on arbitrary input states. Adaptions of our methods using, e.g., the techniques introduced in [8], to this type of performance assessment of subroutines would be interesting future work.

### *Benchmarking Hybrid Algorithms*

Some quantum algorithms are hybrid algorithms that contain both classical and quantum subroutines, e.g., the Variational Quantum Eigensolver, Shor’s algorithm, and the Quantum Approximate Optimization Algorithm. The performance of these algorithms depends on the performance of both the classical and quantum subroutines, as well as how they interact. In this work, we focused on and benchmark only variations of the quantum subroutines. For the Variational Quantum Eigensolver and Shor’s algorithm we benchmark the quantum subroutine of ansatz evaluation and order finding respectively. Focusing on the quantum subroutine ensures all our benchmarks can be implemented on systems that currently do not have the classical co-processing and feedback capabilities required to run hybrid algorithms. It also means that the results can be directly compared to the other benchmarks in our suite. However, to assess quantum hardware’s ability to implement hybrid algorithms it will ultimately be important to develop additional, complementary benchmarks that include both the quantum and classical subroutines of an algorithm.

### *Issues with Different Quantum Computing APIs*

Quantum applications are being implemented using a variety of APIs, e.g., Qiskit, Cirq, Braket, and Q#. These APIs have many features in common, and it is possible to implement our benchmarking circuits in a broadly consistent fashion on

each of these platforms. However, certain constraints exist that make it challenging to accomplish complete consistence across all APIs.

Our benchmarking circuits are defined in a high-level and algorithm-specific gate set, and then transpiled (1) to a standard gate set that we use to define each circuit’s ‘depth’ [see Section III C], and (2) to the native operations of a particular quantum computer in order to run the circuits. There are two challenges to consistently implementing this methodology across different APIs. Firstly, the permitted gates differ between APIs. For example, in the Amazon Braket API, there is not direct support for controlled subcircuits which are used in our definitions of the Monte Carlo Sampling and Amplitude Estimation benchmarks. These benchmarks are therefore not currently available in the Braket implementation of our suite. Second, we define a standardized notion of circuit depth by compiling to a standard gate set. However, the depth of the resultant circuit will depend on the performance of the compilation algorithm, which will be API dependent. The majority of our experiments to date have used the Qiskit API, limiting cross-API comparisons. An API-independent definition of circuit depth will likely be necessary to enable consistent result comparisons across different APIs.

In addition to issues related to circuit compiling, a variety of other API issues exist that make it challenging to consistently implement our benchmarking suite and to gather all of the data of interest. These issues include a requirement in Cirq to include topology information in the circuit definition, the limited availability of the transpiled circuit and detailed metrics of the execution time in Braket and Cirq, and minor issues relating to circuit visualizations. For our work, the Qiskit platform provided the best overall set of features for implementing this suite of benchmarks and generating the analysis described in this manuscript.

### *Future Evolution of our Benchmarks*

In this section we have identified a variety of limitations with our current benchmarking suite, suggesting that it will need significant revisions. Indeed, there is evidence from classical computing that a suite of successful benchmarks will evolve significantly over time as they exhibit weaknesses in use and as hardware evolves. A good example is the evolution of the SPEC (Standard Performance Evaluation Corporation) benchmark suite [55, 56], one of the most successful approaches used for classical computers. Starting in the late 1980s with 10 benchmarks, this suite went through at least 5 major revisions over the next two decades, stabilizing in 2006 with 25 benchmarks, 6 of which have some resemblance to those of the early suite. For this reason, we expect that the first version of our suite of benchmark programs, that we have presented in this paper, will be enhanced substantially over time.

## **VII. SUMMARY AND CONCLUSIONS**

Quantum computing is an emerging technology, and existing quantum computers experience a wide range of complex errors.



To understand the capabilities of current and near-term quantum hardware the research community therefore needs a robust set of performance benchmarks and metrics. In this work we introduced a suite of quantum application-oriented performance benchmarks and demonstrated them on a selection of cloud-accessible quantum computers. Our benchmarking suite is based on a variety of well-known quantum algorithms, and it is designed to probe the effectiveness of quantum hardware at executing applications.

Each of our application benchmarks tests a quantum computer’s performance on variants of that application’s quantum circuits. They are designed to map out performance as a function of the input problem’s size. We measure performance both in terms of result fidelity and the quantum execution time, providing users with proxies for both the quality of and the time to solution. However, as we have discussed in detail, our benchmarks have a variety of limitations which suggest that future revisions will be required — mirroring the development of classical computer benchmarks [55, 56].

One limitation of our benchmarking suite is that our measurements of execution time are currently rudimentary. We primarily quantify compute time in terms of the time required to execute each quantum circuit. This is rudimentary because the definition of this time can vary by provider, and it neglects the classical co-processing inherent in quantum applications (e.g., circuit compilation time) that must be quantified for an accurate measure of the total compute time. We suggest that further research is required into how to delineate and consistently measure each aspect of total compute time.

Our primary method for summarizing a quantum computer’s result quality on our benchmarks is volumetric benchmarking [8, 23] plots, which display performance as a function of the problem input size (circuit width) and the quantum computation’s length (circuit depth). This framework enables transparent comparisons between applications, as well as comparisons to other performance metrics. In this work we compared performance on our benchmarks to predictions extrapolated from the quantum volume [10] metric. Our experimental results demonstrate that, on the tested devices, the quantum volume is broadly predictive of application performance. However, the reliability of extrapolating the quantum volume to predict application performance varies between devices. A more quantitative and comprehensive comparison between performance on application-oriented benchmarks and application-agnostic benchmarks (e.g., the quantum volume benchmark [10] or mirror circuit benchmarks [8]) would constitute interesting future work.

The performance of the hardware that we tested suggests that substantial advances are required for quantum computers to outperform their classical counterparts on useful tasks. For example, our volumetric plots clearly communicate that an improvement in the quantum volume from, e.g., 32 to 2048, will not make it possible to successfully execute Shor’s algorithms on useful problem instances — which, as is well-known, will almost certainly require large-scale quantum computers running fault tolerant quantum error correction. However, the potential for advances in algorithms, in quantum computing hardware, and in the software stack mean that quantum advantage could

be achievable in the near-term. As new algorithms are developed they can be added to our suite, and our benchmarks can quantify the performance gains provided by technological advances. Therefore, our suite is well-suited to tracking progress towards useful quantum advantage.

Cloud-accessible quantum computers are attracting a wide audience of potential users, but the challenges in understanding their capabilities is a significant barrier to adoption of quantum computing. Our benchmarking suite makes it easy for new users to quickly assess a machine’s ability to implement applications, and our volumetric visualization of the results is designed to be simple and intuitive. We therefore anticipate that our benchmarking suite will encourage adoption of quantum computing, and further economic development within the industry.

## CODE AVAILABILITY

The code for the benchmark suite introduced in this work is available at <https://github.com/SRI-International/QC-App-Oriented-Benchmarks>. Detailed instructions are provided in the repository. The source code for both quantum volume and volumetric benchmarking are contained in their respective public repositories [57, 58].

## ACKNOWLEDGEMENT

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. IBM Quantum. <https://quantum-computing.ibm.com/>, 2021. We acknowledge IonQ for the contribution of results from hardware not yet generally available. We acknowledge Quantinuum for contributing the results from their commercial H1.1 hardware. Timothy Proctor’s contribution to this work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research through the Quantum Testbed Program. 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. 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.

## Appendix A: Selected Algorithms and Applications

In this appendix we examine the quantum programs that we selected for use as benchmarks, and describe how we turned those programs into benchmarks. Many of these programs are familiar to users of quantum computing from tutorial materials and academic papers. Generally, tutorials do not develop these programs in a way that makes them usable as benchmarks that are scalable to different problem sizes and that are applicable to all machines. Starting with tutorials and several advanced application examples, we identified the key adaptations required to turn them into practical benchmark programs.

In this appendix we highlight the challenges that were encountered when converting these algorithms and applications into benchmarks, and how those challenges were addressed. We evaluate how reliably the benchmarks perform in evaluating system capabilities, and how well the results of the benchmarks correlate with one another using the volumetric positioning framework discussed above.

Our benchmarks are not designed to evaluate trade-offs or design variations within the algorithms or applications themselves. This is because we use a very specific implementation of most algorithms. We use the benchmarks to provide insight into the hardware's performance, not to compare how well variants of an algorithm or application arrive at a solution to a problem.

In this section, all of numerical results presented are based on execution of the application circuits on a noisy simulator with a specific noise model. The noise model consists of one- and two-qubit depolarizing errors with error rates of 0.003 and 0.03, respectively.

### 1. Shallow Simple Oracle Based Algorithms

The first group of quantum programs that we converted into benchmarks consist of three shallow, simple 'oracle' algorithms.

- Deutsch-Jozsa Algorithm [59]
- Bernstein-Vazirani Algorithm [13, 60, 61]
- Hidden Shift Problem [13, 61, 62]

All of these algorithms share a simple structure in which the primary qubits of the circuit are initialized into a quantum superposition and a 'secret' integer value is embedded into the circuit using a special 'oracle' operation. The oracle uses quantum gates to manipulate the relative phases of the qubits to effectively 'hide' the integer, encoded as a string of ones and zeros, in the resulting quantum state. At the end of the circuit, the quantum state is transformed back into the computational basis and the secret integer value is revealed.

These algorithms are often used in tutorials to illustrate how a quantum computer is able to determine the secret integer encoded by an oracle function in a single execution of the oracle. In a classical computer, the oracle function would need

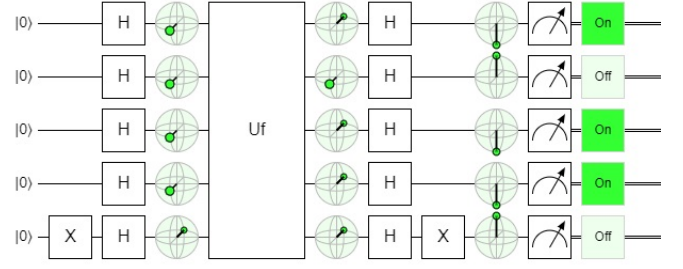


FIG. 21. Bernstein-Vazirani Circuit Diagram

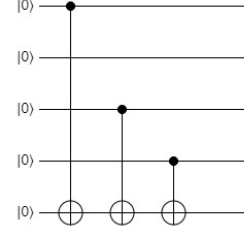


FIG. 22. Bernstein-Vazirani Oracle Function  $U_f$  - Encoding the Integer 13

to be executed repeatedly. In theory this demonstrates a potential advantage quantum computers could have over classical computers when operating on a specific class of problem.

Figure 21 shows a circuit diagram for the Bernstein-Vazirani algorithm (BV) as it is implemented on 5 qubits, one of which is used as an ancilla or accessory qubit. An example of an oracle circuit  $U_f$  encoding the integer value 13 (bitstring '1101') is shown in Figure 22. Each '1' bit is encoded using a controlled NOT gate on the 5<sup>th</sup> ancilla qubit.

To illustrate how these circuits work, we have inserted small Bloch sphere icons into the main circuit diagram before and after the oracle and again before the measurements. One can see how the oracle operates on the uniform superposition generated by Hadamard gates on the first 4 qubits to encode the secret integer into a quantum state. Each qubit that performs a controlled NOT operation will experience phase kickback that rotates the phase of its quantum state by  $\pi/2$ . The final set of Hadamard gates transforms the qubits back into the computational basis and the value encoded in the oracle is revealed upon measurement.

The main circuit of the Deutsch-Jozsa algorithm (DJ) is identical to the Bernstein-Vazirani algorithm, but it uses a slightly more complex oracle that encodes either a 'constant' or 'balanced' function. The oracle is very similar to the Bernstein-Vazirani — it also uses a sequence of controlled NOT gates — and so it is not shown here.

The Hidden Shift algorithm (HS) applies two oracle functions in a row, unlike the DJ and BV algorithms. The first application of the oracle is identical to the second application, except that it includes a hidden 'shift' — a secret integer that encodes a difference between the values returned from the two oracles (shown in Figure 23). The two oracles together result

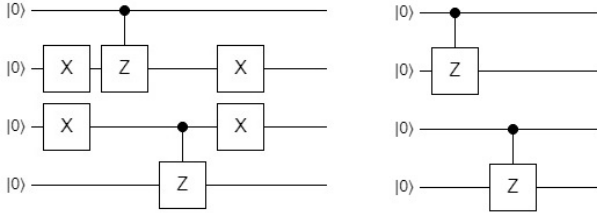
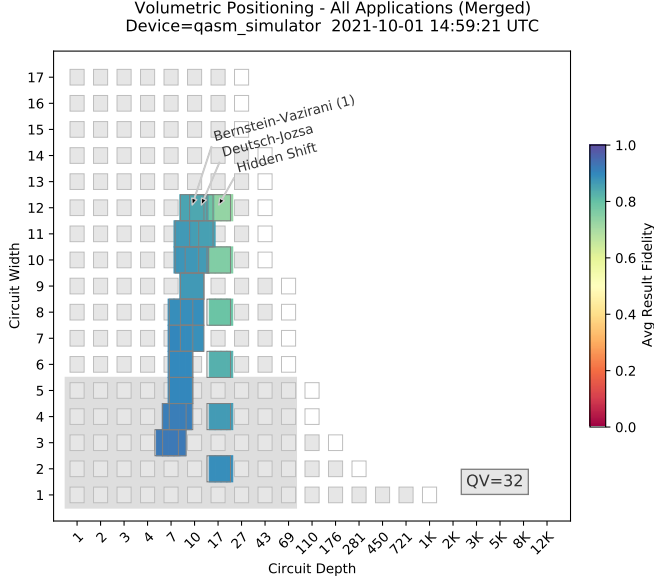
FIG. 23. Hidden Shift Oracle Functions -  $U_f$  and  $U_g$ 

FIG. 24. Benchmark Results for Shallow Simple Oracle Algorithms

in a circuit depth for the HS that is nearly double that of the BV and DJ algorithms. Due to the similarities between these algorithms, we group them together as the ‘tutorial’ category.

To implement each of these algorithms as a benchmark, each circuit of a specific width is executed multiple times with a random integer encoded in the oracle. In our data analysis, we choose to present results that are averaged over all circuits of the same width.

Figure 24 shows the results obtained from executing these 3 benchmarks on the noisy simulator, for 2 qubits up to 12 qubits. The scale of the depth axis for this plot has been stretched out so that differences in the depth of these shallow circuits can be seen clearly. When using the scale shown in other plots, results from these three circuits are difficult to distinguish.

Even for such simple circuits a great deal can be learned about using algorithms or applications as benchmarks. Several of the key observations are noted below. These will be relevant also in the analysis of the more complex examples that follow.

- Both the Deutsch-Jozsa and Bernstein-Vazirani Algorithms use an ancilla qubit to provide phase kickback to encode the secret integer, while the Hidden Shift does not. It uses multiple pairs of qubits entangled with a controlled Z gate and does not require an ancilla to encode the hidden shift. In order to define a benchmark on at

least 2 operational qubits, the DJ and BV benchmarks require a minimum of 3 qubits while the HS needs only 2 qubits.

- The Deutsch-Jozsa and Bernstein-Vazirani Algorithms are almost identical in circuit depth, differing only in the small difference in oracle components. They both increase in depth linearly with the number of qubits under test, as a larger number of qubits necessitates a corresponding increase in number of controlled gate operations. As expected from this, the fidelity degrades slowly with increasing circuit width.
- The Hidden Shift benchmark is different in that the circuit depth is constant. There are at most two X operations involved, one before and one after the CZ gate that entangles successive pairs of qubits, and as a result the circuit depth does not grow. A noticeable attribute of the Hidden Shift algorithm defined here is that successive pairs of qubits are completely independent of one another. As circuit width is increased successively by 2, there is added one additional clone of the first pair of qubits. A benchmark of 10 qubits is indistinguishable from 5 copies of a 2 qubit Hidden Shift benchmark.

## 2. Quantum Fourier Transform

The Quantum Fourier Transform (QFT) [63] and its inverse are used extensively as a component of many well-known quantum applications. We implemented two variations of a Quantum Fourier Transform (QFT) benchmark. The first version, which we refer to as the QFT (1) benchmark is shown in Figure 25 and is as follows. A random integer state (13 in the example shown in Figure 25) is initialized on the qubits under test. This is followed by a QFT circuit which encodes the integer value into the Fourier basis, through a series of Hadamard gates and controlled Z rotations. Then a set of simple rotation gates are applied that add 1 modulo  $2^n$  to the number encoded by the QFT. We then perform an inverse QFT sub-circuit, which converts the integer state back into the computational basis, and then all the qubits are measured. The resulting bit string should be the input plus 1 modulo  $2^n$  (in the example of Figure 25, 14, rather than the initial value of 13). This is a Loschmidt echo, or a form of mirror circuit [8], in which the QFT is undone by the inverse QFT that follows it. The inclusion of the +1 operation in the middle of the circuit prevents a transpiler from easily compiling this circuit down to an identity. Additionally, it adds sensitivity to errors that would be canceled by a perfect Loschmidt echo (a forwards and backwards evolution experiment) but that do impact the performance of the QFT. Note, however, that a randomly selected operation in the center of the circuit is required to guarantee sensitivity to all errors [8].

The circuits of our QFT (1) benchmark are displayed in Figure 25. The inverse QFT is not shown as it is a mirror image of the QFT circuit. Note also that this QFT implementation does not use SWAP gates (so it effectively reverses the order of the qubits) which simplifies its use in the benchmark programs.

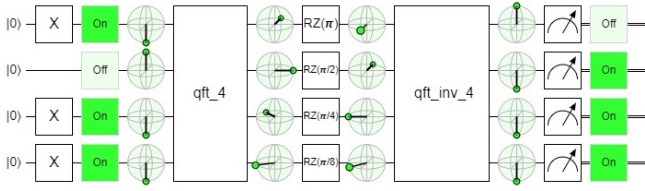


FIG. 25. Quantum Fourier Transform Benchmark Circuit (1)

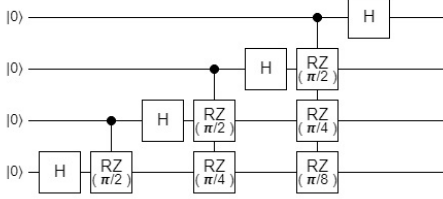


FIG. 26. Quantum Fourier Transform Sub-Circuit

Our second QFT benchmark, the QFT (2) benchmark, executes only the inverse QFT sub-circuit. In our benchmark, the inverse QFT operates on a quantum state in the Fourier basis. In this method, instead of creating that quantum state using the QFT (which converts any integer into the Fourier basis), we use a set of one-qubit gates — specifically, a series of Hadamard gates and Z rotations, to encode a specific integer value. Figure 27 demonstrates this circuit, for the case of encoding the integer value 6. The inverse QFT reverses that operation, yielding the expected integer value in its measurements.

The circuits of the QFT (2) benchmark are similar to the second half of circuits of the QFT (1) benchmark, but version (2) results in a circuit depth that is approximately half that of method 1. A useful consequence of this is that the benchmarks can be used to cross-validate each other: if both benchmarks are a good quantification of a quantum computer’s performance on the QFT and its inverse, and the QFT and its inverse have approximately the same error, then the result fidelity from QFT (1) should be approximately the square of the result fidelity from QFT (2).

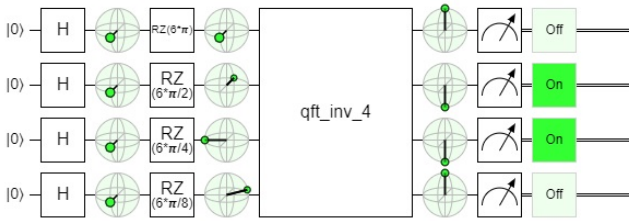


FIG. 27. Quantum Fourier Transform Benchmark Circuit (2)

Figure 28 shows the results obtained from executing these two variations of the QFT benchmark. We see the expected relationship between the fidelities of the two benchmarks (note that this simple relationship can be proven to hold for simple

depolarizing errors like those used in this simulation). Note that the circuit depth scale is significantly compressed compared to the Shallow circuits shown in Figure 24 in order to plot circuits with greater depth. Deeper circuits described later in the paper use an even more compressed scale.

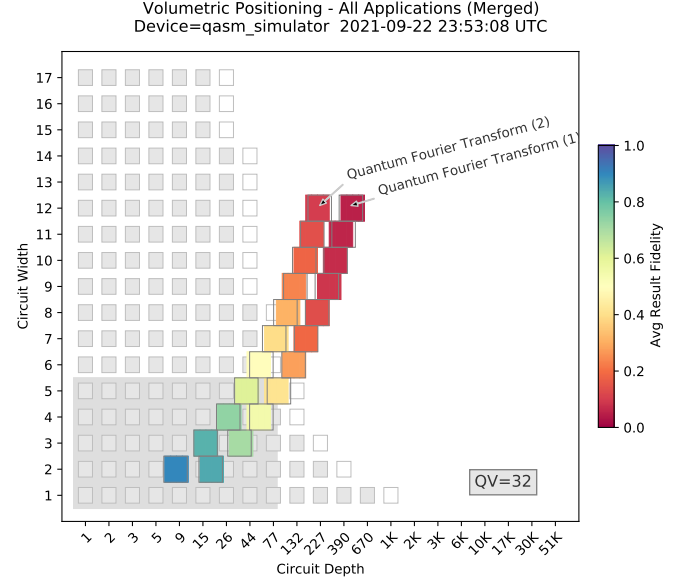


FIG. 28. Benchmark Results for Quantum Fourier Transform Algorithms

### 3. Grover’s Search Algorithm

Grover’s algorithm [64], also referred to as the quantum search algorithm, is one of the most well known quantum algorithms due to its quadratic run-time speedup from the best known classical algorithm. This algorithm aims to find the correct item from an un-ordered list. It achieves a quantum speedup by starting in a uniform superposition and using repeated calls to a quantum oracle to amplify the probability of measuring the correct state. The amplification used in Grover’s algorithm was then generalized into amplitude amplification [65], which is used in the Amplitude Estimation and Monte Carlo benchmarks.

Our Grover’s Search benchmark is designed so that the oracle marks a given bit string with a phase of -1. At each circuit width, multiple circuits are constructed by generating random bit strings of length equal to the width of the circuit. The fidelity of these circuits is then averaged to estimate the performance of this benchmark. Figure 29 shows the results of executing our Grover’s Search benchmark on the simulator. Figure 30 highlights the drastic increase in circuit depth with increasing circuit width for this algorithm — causing a rapid drop in the result fidelity as the circuit width grows. This is the result of Grover’s algorithm requiring many MCX gates, with  $n - 1$  controls for  $n$  qubits. These gates need to be decomposed into one- and two-qubit primitive gates, which substantially increases the depth. Note that decompositions of MCX gates

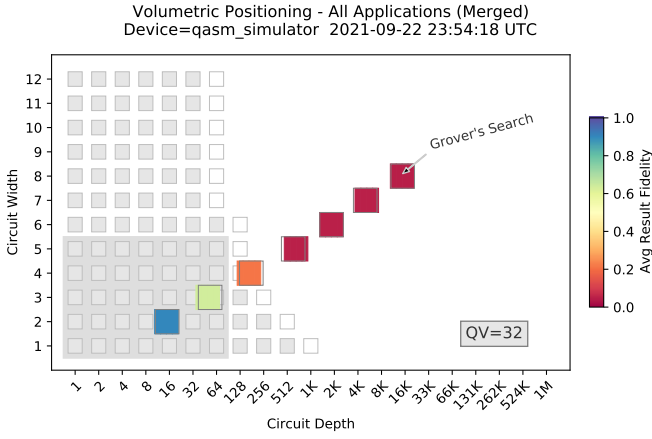


FIG. 29. Grover's Search Benchmark

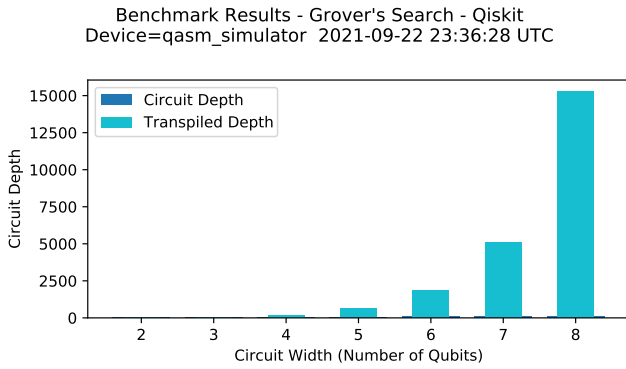


FIG. 30. Grover's Search Depth

can be made shallower by using ancilla qubits, which we do not do here.

#### 4. Phase and Amplitude Estimation

Similar to the Quantum Fourier Transform, Quantum Phase Estimation [66] is another important quantum subroutine fundamental to many quantum algorithms, including Shor's algorithm [67]. The goal of the algorithm is to estimate the eigenvalues of a unitary operator by calculating the corresponding phases. A generalization of phase estimation is amplitude estimation [65] where the goal is to estimate the amplitudes of a quantum state. Due to the importance of phase estimation and amplitude estimation as subroutines for more complicated quantum algorithms, benchmarking these routines will provide insight to the capability of a quantum computer.

For the phase estimation benchmark, we have to pick a unitary whose phase is to be measured. For this, we choose a phase gate with eigenvalues  $\exp(\pm i\theta)$ , where  $\theta$  is defined to a precision  $2^{-k}$ , where  $k$  is the number of qubits in the register that stores the measured phase. The benchmark runs over random values of  $\theta$  of the form  $\frac{n}{2^k}$  where  $k$  represents the total number of measured qubits and  $n$  is an integer between 0 and

$2^k - 1$ .

The amplitude estimation benchmark contains two quantum subroutines: quantum amplitude amplification and quantum phase estimation. The quantum amplitude amplification is used to encode the amplitudes of a target quantum state into a phase which can be extracted using phase estimation. This benchmark performs the simplest case of quantum amplitude estimation where, similar to phase estimation, we chose specific circuits such that the amplitudes are specified to a precision which can be exactly encoded in the available qubits.

Figure 31 shows the results obtained when executing our Phase Estimation and Amplitude Estimation benchmarks. Note the Phase Estimation benchmark circuits' depths scale similarly to the QFT benchmark's circuits — which is because the QFT is the subroutine of phase estimation that carries the most complexity. The Amplitude Estimation benchmark, on the other hand, involves deeper circuits due to the quantum amplitude amplification protocol. This figure shows that a quantum computer with a quantum volume of 32 is incapable of giving any meaningful result (high fidelity) for the Amplitude Estimation benchmark, even with tiny problem instances.

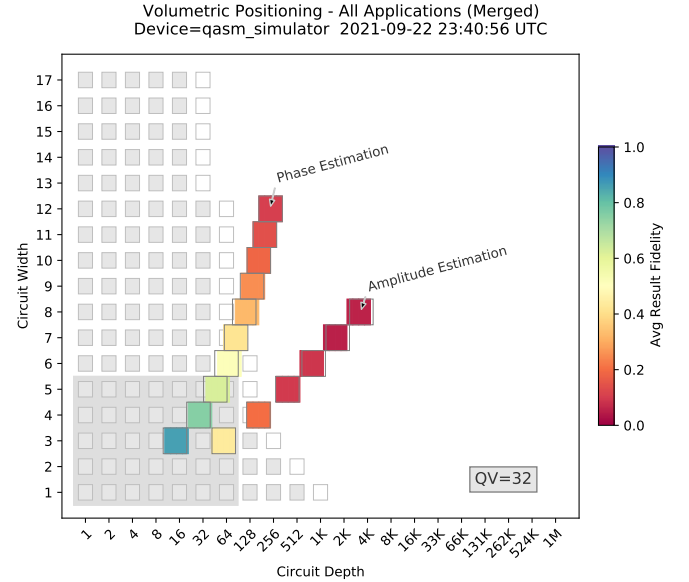


FIG. 31. Phase and Amplitude Estimation Benchmark

#### 5. Hamiltonian Simulation

One of the most promising near-term applications for quantum computers is in the simulation of quantum systems [68]. As the evolution of quantum systems can be described in terms of Hamiltonians, we choose to evolve a non-trivial Hamiltonian as our benchmark. The Hamiltonian we choose is the anti-ferromagnetic Heisenberg chain of 1-D spins with disordered x- and z-fields and an open boundary condition:

$$H = J \sum_{i=0}^{n-2} \vec{\sigma}_i \cdot \vec{\sigma}_{i+1} + w \sum_{i=0}^{n-1} (h_{x,i} \sigma_i^x + h_{z,i} \sigma_i^z), \quad (\text{A1})$$



where  $J$  and  $w$  are the strength of the interactions and the disordered fields respectively,  $h_{x,i}$  and  $h_{z,i}$  give the strength of the  $X$  and  $Z$  disordered fields at site  $i$ ,  $n$  is the total number of qubits, and  $\sigma^{(x,y,z)}$  are the usual Pauli operators. In this benchmark,  $J = 1$ ,  $w = 10$ , and  $h_{x,i}$  and  $h_{z,i}$  are randomly generated in the range  $(-1, 1)$ . The algorithm initializes the qubits in a product state and evolves them according to a Trotterized version of the Hamiltonian [69]. The Trotterization implements the term  $\exp(-iJ\vec{\sigma}_i \cdot \vec{\sigma}_{i+1})$  exactly using a construction from Vatan and Williams [70]. We use a fixed number of Trotter steps. To calculate the result fidelity for this benchmark, we simulate this evolution on an ideal simulator.

Figure 32 shows the results of simulating our Hamiltonian Simulation benchmark. Note that with increasing width, the depth does not increase. This is a result of the Hamiltonian only including local terms, and allows the Hamiltonian simulation benchmark to scale similarly to Hidden Shift.

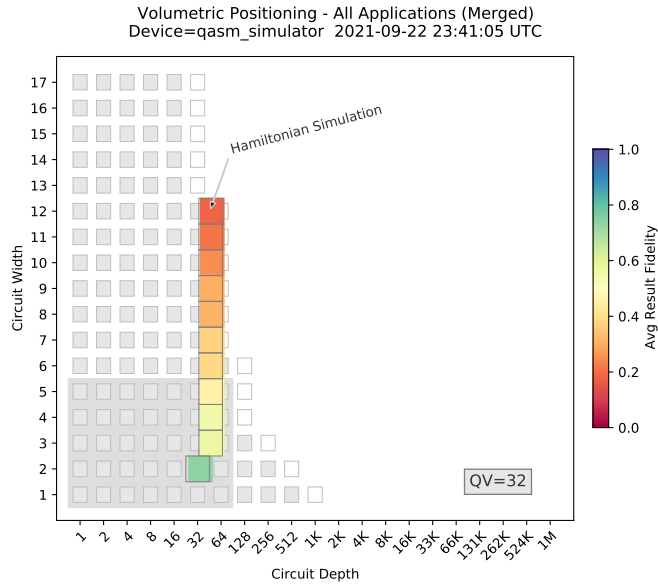


FIG. 32. Hamiltonian Simulation Benchmark

Quantifying the success of the simulation of a quantum computer starting in a state that is not an eigenstate is a difficult problem, as in order to exactly verify the quantum computer ran correctly, it must be compared against the computationally expensive classical simulation. This sort of reasoning was highlighted by Boixo et al. [9], where the fidelity metric they use is intractable in the regime of quantum supremacy. This benchmark does not currently solve this issue as our fidelity is calculated compared to error free evolution, which scales exponentially on the ideal simulator.

One suggested method to modify this benchmark is by estimating the fidelity of the simulation through a phenomenon called many-body localization [71, 72], where evolution under a random Hamiltonian such as the one in Eq. A1 preserves certain quantities. However, it is not realistic as a benchmark for current quantum computers because it requires evolution to long times to get into a regime where many body localization physics can be seen. This results in circuits that are thousands

of gates long. In addition, it requires averaging over several random circuits to be able to properly estimate the hardware performance. All of these factors make it such that running this enhanced benchmark would be exceedingly expensive and/or time-intensive to run on current hardware.

## 6. Monte Carlo Sampling

Classical Monte Carlo algorithms use random processes to solve problems numerically where other methods might be infeasible. A general problem description might involve a function of a random variable for which we would like to find the expected value given the probability distribution of the variable. A quantum algorithm which has a quadratic speed up over classical Monte Carlo algorithms was proposed by Woerner and Egger [73], which we have implemented here as the Monte Carlo Sampling benchmark.

Suppose a random variable  $X$  follows the probability distribution  $P(X)$ , and we are interested in a function  $f$  with respect to  $X$ . The goal is to compute the expectation value of  $f$

$$E[f(X)] = \sum_i p(X = i) f(i). \quad (A2)$$

Firstly, we define the oracle as

$$\mathcal{R} |0\rangle_n |0\rangle = \sum_i \sqrt{p(X = i)} |i\rangle_n |0\rangle \quad (A3)$$

in which  $\mathcal{R}$  encodes the desired probability distribution into the amplitudes of  $n$  qubit states.

We also define the operator  $\mathcal{F}$  as

$$\mathcal{F} |i\rangle_n |0\rangle = |i\rangle_n (\sqrt{1 - f(i)} |0\rangle + \sqrt{f(i)} |1\rangle) \quad (A4)$$

in which  $\mathcal{F}$  encodes the function value  $f(i)$  into the amplitude of a single auxiliary qubit.

By applying  $\mathcal{R}$  and  $\mathcal{F}$  on a  $n + 1$  qubit system initialized to the zero state, we have

$$\begin{aligned} \mathcal{F}\mathcal{R} |0\rangle_n |0\rangle = \\ \sum_i |i\rangle_n (\sqrt{p(X = i)} \sqrt{1 - f(i)} |0\rangle + \sqrt{p(X = i)} \sqrt{f(i)} |1\rangle). \end{aligned} \quad (A5)$$

Recall that for the amplitude estimation, given the operator  $\mathcal{A}$ ,

$$\mathcal{A} |0\rangle_{n+1} = \sqrt{1 - a} |\psi_0\rangle |0\rangle + \sqrt{a} |\psi_1\rangle |1\rangle, \quad (A6)$$

and the amplitude estimation algorithm yields  $a$ . For the Monte Carlo simulation, using

$$\mathcal{A} = \mathcal{F}\mathcal{R} \quad (A7)$$

we can perform amplitude estimation to produce

$$a = \sum_i p(X = i) f(i) = E[f(X)], \quad (A8)$$

which is the expectation value of the  $f$  function.

As all Monte Carlo methods output approximations which converge as the number of samples is increased, one figure of merit for success is the variance in the mean in number of samples. While classical methods scale with the number of samples  $M$  as  $O(M^{-1})$ , this quantum algorithm provides advantage by scaling as  $O(M^{-1/2})$  [73]. In the documentation of the benchmark repository (See Section VII), we discuss at a deeper level the intuition for the algorithm and the circuits used for the benchmark.

Figure 33 shows the result of executing the two methods that are implemented for the Quantum Monte Carlo Sampling benchmark. Method 1 allows the user to calculate the expected value of arbitrary distributions and functions. However, as this generally results in very large circuit depths, we also implement method 2. This method uses a fixed uniform distribution, easily created using Hadamard gates, and the function in which  $f(i) = 1$  if  $i$  has an odd number of 1's in its binary expansions and  $f(i) = 0$  otherwise, easily preparable with only CNOT gates.

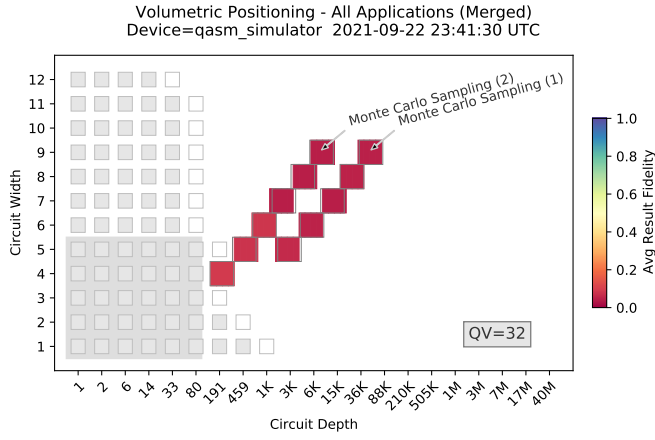


FIG. 33. Monte Carlo Sampling Algorithms

While many aspects of this algorithm's implementation are the same as in the Amplitude Estimation benchmark, the fidelity calculation is modified to account for our expectation values being real numbers of arbitrary precision. As with Amplitude Estimation, the qubits where the amplitude is encoded, referred to as 'counting qubits', can only approximate the amplitude to a binary approximation with significant figures equal to the number of counting qubits. This results in the algorithm returning the closest bit string with a high probability (relatively). However, since the bit string is an approximation, not an exact result, there will be other measurements returned that are also valid, but with lower probability.

To address this issue, separating algorithmic error from hardware error, the Hellinger fidelity is compared to the analytical probabilities of being in a particular state guaranteed by the phase estimation subroutine according to Nielsen and Chuang [34] as

$$|\alpha_b|^2 \equiv \left| \frac{1}{2^t} \left( \frac{1 - e^{2\pi i(2^t \varphi - b)}}{1 - e^{2\pi i(\varphi - b/2^t)}} \right) \right|^2, \quad (\text{A9})$$

with  $t$  as the number of counting qubits,  $b$  being the raw bit-

string, and  $\varphi$  being the exact phase corresponding to the amplitude we are searching for from the phase estimation subroutine.

As seen in the other deep algorithms, a quantum computer with a quantum volume of 32 is incapable of giving a meaningful result for the Monte Carlo algorithm. Figure 7 shows that it requires a machine with quantum volume of 512 or more to obtain useful results from this algorithm.

## 7. Variational Quantum Eigensolver

Solving the electronic structure problem is one of the most promising applications for quantum computers, with potential contributions to drug discovery [74], developing new materials and catalysts for more efficient light-harvesting and CO<sub>2</sub> reduction [75], understanding high-temperature super conductivity [74], and other important use cases [76]. In the NISQ era, one of the most promising approaches is the variational quantum eigensolver (VQE) [48, 49].

The VQE algorithm implements a parametrized circuit for a chosen wavefunction ansatz. It then measures the energy expectation value of the wave function and optimizes the circuit parameters towards a direction that lowers the energy expectation value. The VQE algorithm is based on the variational principle [77],

$$E = \frac{\langle \Psi(\vec{\theta}) | H | \Psi(\vec{\theta}) \rangle}{\langle \Psi(\vec{\theta}) | \Psi(\vec{\theta}) \rangle} \geq E_0, \quad (\text{A10})$$

in which the energy expectation value of an parameterized wave function  $\Psi(\vec{\theta})$  is lower bounded by the exact ground state energy  $E_0$ . In other words, the optimal parameter  $\vec{\theta}^*$  that minimizes the energy expectation value corresponds to the closest description to the exact ground state within the flexibility of the wave function ansatz.  $H$  is the electronic Hamiltonian defined as

$$H = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} g_{pqrs} a_p^\dagger a_q^\dagger a_s a_r \quad (\text{A11})$$

in which  $h_{pq}, g_{pqrs}$  are the one- and two-electron integrals, and  $a, a^\dagger$  are the Fermionic annihilation and creation operators

The VQE algorithm belongs to a wide range of quantum-classical hybrid algorithms. The energy expectation value is measured on a quantum computer, and the optimization task is performed with a classical computer. Comparing with other approaches such as quantum phase estimation (QPE), VQE requires shallower circuit depths [48], which makes it particularly suitable for near-term quantum computers. However, unlike QPE, which yields the exact energy prediction, the accuracy of the VQE algorithm depends on the accuracy of the wave function ansatz.

The most crucial part of the VQE algorithm is the choice of the wave function ansatz. Many choices exist, and in this benchmark we choose to use the unitary Coupled Cluster with singles and doubles ansatz (unitary-CCSD). This ansatz is defined as [78]

$$|\Psi\rangle = e^{T-T^\dagger} |\Phi\rangle \quad (\text{A12})$$

in which  $T$  is the cluster operator,  $|\Phi\rangle$  is the reference state, which is chosen to be the Hartree-Fock (HF) state. The cluster operator is defined as

$$T = \sum_{i,a} t_i^a a_i^\dagger a_a + \sum_{i>j, a>b} t_{ij}^{ab} a_i^\dagger a_j^\dagger a_b a_a \quad (\text{A13})$$

in which  $i, j$  indicate molecular orbitals that are occupied in the HF state, and  $a, b$  indicate molecular orbitals that are empty in the HF state.  $t_i^a$  and  $t_{ij}^{ab}$  are the wave function parameters associated with single and double excitations.

The unitary CCSD ansatz is a variant of the widely used traditional CCSD ansatz. Unlike the traditional CCSD ansatz, which fails to model strongly correlated systems, the unitary CCSD ansatz has been shown to produce highly accurate results in both weakly and strongly correlated systems. However, there is no known classical method that could evaluate the energy expectation value of the unitary CCSD ansatz with polynomial cost. To the contrary, its unitary nature provides a straightforward and efficient way to implement it on a quantum computer.

Both the electronic Hamiltonian and the unitary CCSD ansatz is defined with Fermionic operators. In order to implement them on a quantum computer, one needs to first transform them to Pauli operators. In our implementation we use the Jordan-Wigner transformation [79]:

$$\begin{aligned} a_p^\dagger &\rightarrow I^{\otimes p-1} \otimes \left[ \frac{1}{2}(\sigma_p^x - i\sigma_p^y) \right] \otimes \sigma^{\otimes N-p} \\ a_p &\rightarrow I^{\otimes p-1} \otimes \left[ \frac{1}{2}(\sigma_p^x + i\sigma_p^y) \right] \otimes \sigma^{\otimes N-p} \end{aligned} \quad (\text{A14})$$

in which  $I, X, Y$  and  $Z$  are Pauli matrices and  $N$  is the total number of qubits. Upon the Jordan-Wigner transformation, both the Hamiltonian and the cluster operator becomes a weighted sum of Pauli words

$$H(T) = \sum_i^M g_i(t_i) P^i \quad (\text{A15})$$

in which  $P^i$  is a tensor product of Pauli matrices,

$$P^i = \{I, \sigma^x, \sigma^y, \sigma^z\}^{\otimes N}. \quad (\text{A16})$$

The VQE algorithm proceeds by iteratively evaluating and optimizing the energy by varying the wave function parameters, until the minimum is found.

The full VQE algorithm includes both a quantum and a classical part. The former is used to implement the ansatz and measure the energy, and the latter is applied to optimize the ansatz parameters. For the sake of simplicity of our initial phase of benchmarking algorithms, we only implement and test the quantum part from two aspects, denoted by method 1 and 2. For method 1, we randomly sample the wave function parameters and compute the expectation value of only one non-trivial term in the Hamiltonian. For method 2, we use a fixed set of wave function parameters and compute the expectation value of the full Hamiltonian.

In our implementation, the Hamiltonian and the wave function is implemented for the NaH molecule with a bond length

of 1.9 Angstrom with the STO-6G basis[77]. By changing the number of orbitals to simulate, we could vary the number of qubits. The smallest case is to use 4 qubits, which corresponds to 2 electrons and 4 spin orbitals.

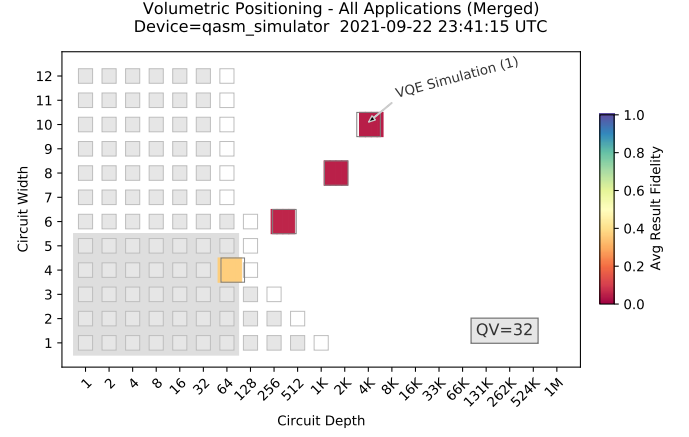


FIG. 34. Variational Quantum Eigensolver Benchmark.

Figure 34 shows the results of executing this benchmark on the quantum simulator up to 10 qubits with method 1. Below are a few observations about this algorithm and the results seen here:

- The benchmark requires a large number of measurements to ensure that the errors of the expectation value are small enough. As been shown before, the number of measurements in VQE scales as  $1/\epsilon^2$ , in which  $\epsilon$  is the desired accuracy.
- By nature of the problem's goal being calculating an expectation value, every term in the Hamiltonian cannot be measured after a single circuit. Multiple circuits need to be made to measure all of the required expectation values.
- Similar to the Hamiltonian simulation, the fidelity is computed by comparing the measured probability distribution against that of the noise-free simulator.
- We are only investigating the quantum aspect of the benchmark right now, and are not investigating choices of optimization routines and hyper-parameters.
- We have two methods: one which measures the expectation values of the entire Hamiltonian (and thus has many individual circuits to run), and one which calculates the expectation value of only a single Pauli operator from the JWT Hamiltonian (and only runs a single circuit.)
- This benchmark, similar to Hamiltonian simulation, requires comparing against the classical simulator to calculate fidelity currently. This is something that we would like to address in the future.

## 8. Shor's Order Finding

Shor's Factoring Algorithm [67] is one of the most well-known applications of quantum computing. Shor's algorithm performs integer factorization, a cryptosystem protocol used by the RSA, exponentially faster than any classical algorithm. However the resources required to factor even a small number are significant, making Shor's algorithm impractical with current quantum hardware. For this reason, it serves as a useful benchmark to test the capabilities of current and future quantum hardware [34].

Similar to the Variational Quantum Eigensolver, Shor's factoring algorithm includes both a quantum and classical subroutine. The quantum subroutine involves finding the smallest integer period, or the order, of a periodic function while the classical subroutine uses this order to determine the factors of a number in polynomial time. This process may need to be repeated a few times in case an invalid order is generated. Since the order finding routine is the bottleneck of integer factorization, the focus is on benchmarking the quantum subroutine of order finding.

The algorithm is based on functions that take the form  $f(x) = a^x \bmod N$  where  $a < N$ . The order  $r$  of this function occurs when  $a^r \bmod N = 1$ . If  $r$  is known, a classical computer can compute the prime factors of  $N$  in polynomial time. This benchmark generates random values of  $r$  and  $N$  from which a corresponding base value  $a$  can be analytically calculated. Since the expected measured distribution is solely based on the  $r$  and  $N$ , the measured distribution is compared to the expected distribution via the Hellinger calculation to determine the fidelity.

The quantum algorithm for order finding is actually a variation of quantum phase estimation where the chosen unitary operator performs modular exponentiation. This modular exponentiation encodes the order in the phases of a quantum state.

There are two variations of Shor's Order finding benchmark implemented for this benchmark, both introduced by Stephane Beaulieu [80]. The first method is considered the standard Shor's algorithm requiring  $4n + 2$  qubits. The second method is an improvement on method 1 that utilizes mid-circuit measurements to reduce the number of control qubits. This method requires  $2n + 3$  qubits. A deeper discussion about the limitations and implications of benchmarking with mid-circuit measurements can be found in Appendix B 1. Figure 35 shows the results obtained executing both implementations of Shor's order finding algorithm.

As seen in the other deep algorithms, a quantum computer with a quantum volume of 32 is incapable of giving a meaningful result for the Shor's Period Finding algorithm. A question naturally arises from the results seen here. What is the level of fidelity that would be required of the order finding algorithm before it will be suitable for use in the complete Shor's application, i.e. to find the factors of large numbers? To determine this, the benchmark would need to be extended to include the iterative classical attempts to use the order-finding algorithm to determine the factors.

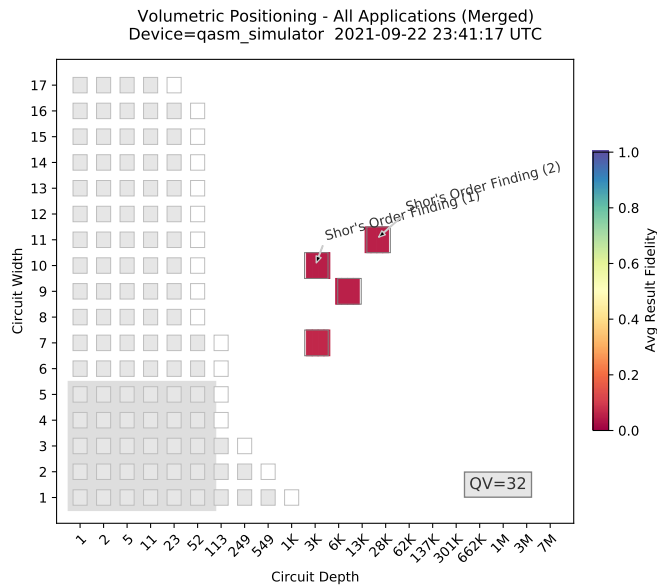


FIG. 35. Shor's Order Finding Algorithms



## Appendix B: Advances in Quantum Computers

Developing a suite of performance benchmarks early in a technology cycle establishes markers with which to measure advances in both hardware components and supporting software. In order to provide this benefit over multiple product cycles, they must be designed to anticipate advances in technology that may emerge in the next five years. Factoring such advances into benchmarks from the beginning helps to ensure that future comparisons are meaningful.

We considered specific enhancements that have been openly discussed within the quantum computing community and reviewed how these would impact the code structure of the application benchmark programs. Examples are new features such as mid-circuit measurement, parameterized gate operations, tighter integration of classical / quantum computation. Some of these are available now on a limited set of quantum computers while others remain on the drawing board.

With input from the community as well as the providers of quantum computing hardware, the suite can evolve to effectively gauge the impact of performance improvements that result from these continuing advances.

### 1. Mid-Circuit Measurements

In the first quantum computers introduced by commercial providers, classical measurement of qubit state was permitted only at the end of the circuit and qubits were not reusable. Recently, IBM [81, 82], Quantinuum [39, 83] and others have introduced ways to perform measurement at any stage of the quantum circuit. The measurements are recorded, qubits are reset to an initial state, and additional computation can be performed while qubit coherence is maintained by the quantum hardware. The measurements collected during execution may be retrieved by the user with an API call and processed as if they had been taken on individual qubits.

This feature provides an option for constructing algorithms with a smaller number of qubits. Circuits implemented with more than 5 qubits often suffer from loss of fidelity traceable to the large number of SWAP and CNOT gates generated in transpilation to account for system topology. Nation and Johnson [81] describe an implementation of a Bernstein-Vazirani circuit on IBM hardware using only 2 qubits to encode a 12-bit secret string. The fidelity in the measurement result is significantly greater than the equivalent 13-qubit circuit due to the use of only two qubits. Figure 36 shows a smaller version of that circuit that is equivalent to a 5 Qubit implementation.

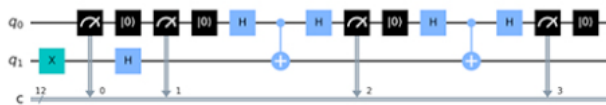


FIG. 36. Equivalent 5 Qubit Bernstein-Vazirani on 2 Qubits

The application benchmark suite is able to account for this

mid-circuit measurement advance by creating a circuit using a smaller number of qubits that represents the same problem as a circuit implemented with a larger number of qubits (a ‘qubit-equivalent’ problem size). In the normal Bernstein-Vazirani benchmark, a sweep is performed over a progressively larger number of qubits and encodes a larger secret integer each time. An optional method is implemented that performs the same sweep over the number of qubits representing the problem size, but uses only two qubits. The depth of the circuit grows instead of the width. Fidelity results are shown as a series of 2 qubit circuits but with multiple results at different depths. With this approach the results of the mid-circuit measurement circuit can be compared with the results from the multi-qubit circuit implementation.

Griffiths and Niu [84] have described a semi-classical approach to performing the Quantum Fourier Transform that requires only a single qubit with mid-circuit measurement. Using this algorithm, the Quantum Fourier Transform benchmark can be implemented using the same approach as the Bernstein-Vazirani, holding width constant while replicating depth components.

The Shor’s Period Finding benchmark includes a method (2) that uses a single qubit Quantum Phase Estimation circuit, implemented with mid-circuit measurement. Until the recent introduction of mid-circuit measurement in hardware systems, this algorithm could only be performed on a quantum simulator. In the Shor’s benchmark, this single qubit algorithm is implemented as part of the complete period finding algorithm and is not analyzed independently. The effect of the mid-circuit measurement can be seen in the reduction in circuit width and in the fidelity of the results in Figure 35.

### 2. Circuit Parameterization

Many of the common algorithms proposed for quantum computers are highly iterative. In the simple case, this involves the repeated execution of an identical circuit hundreds or thousands of times (shots) and the analysis of the probability distribution of measurement results to extract useful information (an “answer”).

Several more complex algorithms, such as the Variational Quantum Eigensolver (VQE) or the Quantum Approximate Optimization Algorithm (QAOA), introduce an additional iteration layer that involves modifying the circuit before executing another round of shots. The circuit is typically changed only by modifying the rotation angles (parameters) on a few of the gates in the circuit before executing it again. A familiar example of this is the parameterized ansatz circuit in the VQE algorithm, an example of which is shown in Figure 37 taken from [11].

Modification of rotation angles prior to execution can be done by creating the circuit from scratch in a high-level language (e.g. Qiskit), followed by compilation for the backend target. The time taken to create, compile and re-load it for execution can be significant, on the order of a few seconds to minutes for larger circuits. For an algorithm that requires



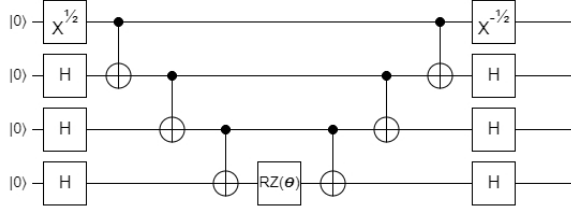


FIG. 37. Single Parameter UCC Circuit Ansatz

hundreds or thousands of iterations, this circuit preparation time dramatically impacts the total "time to solution".

The Qiskit and Cirq programming APIs both support a form of parameter definition within circuits using Python *sympy* objects that partially improves performance by reducing compilation, but still requires reloading of the circuit. It is possible that future advances in quantum computer control systems will be able to reduce the latency between parameterized executions to near 0, which would offer the optimal execution time for such applications.

The Variational Quantum Eigensolver benchmark includes the execution of a parameterized circuit. The ansatz circuit is executed repeatedly but with different angles applied to its gates. Using the default method, this is implemented by creating, compiling and executing the circuit each time as a completely new circuit. The result is that there is latency involved in executing the circuit repeatedly.

In the current version, the benchmarks in this suite are not written to take advantage of parameterized circuit representation and execution. In a future work, the benchmarks could be enhanced with the addition of a method to define the ansatz as a parameterized circuit and measure precisely the various stages involved in its repeated execution.

### 3. Multiply-Controlled Gates

Several of the benchmark circuits are implemented using various multiply-controlled gate operations. This is a type of gate in which the operation applied to the target qubit is controlled by more than just a single qubit. The multiply-controlled MCX gate is used in Grover's as well as Amplitude Estimation and Monte Carlo Sampling benchmarks. Several double controlled CCX gates are used in the modular exponentiation circuit within Shor's Period Finding benchmark.

While a simulator is able to execute these gates efficiently using matrix operations, none of today's physical hardware implementations of a quantum computer are able to implement these natively. Any multiply-controlled gate needs to be transpiled into many smaller primitive gate operations. Figure 38 shows the decomposition of a simple Toffoli CCX gate which has two controls.

As circuit width is increased, the number of controlling qubits applied to the MCX in some of the benchmarks is in-

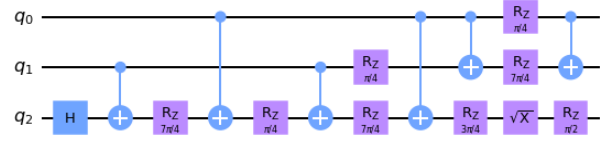


FIG. 38. Decomposition of Doubly Controlled CCX Gate

creased proportionally. The decomposition depth of the MCX gate increases dramatically with the number of controlling qubits. Thus the circuits grow in depth very rapidly as circuit width increases. This is readily seen in the volumetric positioning profile for the Grover's Search benchmark in Figure 29.

There is evidence that hardware advances will incrementally lead to native implementation of multiply-controlled gates. For example, Bækkegaard et al. [85] have described a possible native implementation of a Toffoli gate as a CCZ gate. This is an early indicator of potential advances in the development of multiply-controlled gate operations.

When multiply-controlled gates are implemented natively, the application benchmarks that use them will serve as good tests of the performance improvement that would be expected. Given the large difference in error rates for one and two qubit gates, there will be a tradeoff between the MCX error rate and the circuit depth it replaces. Within the benchmark program itself, there would need to be a way to enable or disable native implementations of these gates in order to measure the difference. As these hardware implementations emerge, we may need to revisit the approach to normalization of the circuit depth in order to properly reflect the depth using the native implementation. However, the basic framework of the benchmark should not have to change.

### 4. Close Classical/Quantum Integration

Increased attention is being given to the total execution time of quantum applications [44, 47]. Quantum computers are increasingly able to execute complex applications that include multiple iterations of quantum circuits as subroutines interleaved with classical computation. The total time taken by the application to arrive at a solution, aggregated over all stages of the execution, can be hours or even days. Architectural improvements in the implementation of the hardware and firmware of the quantum computer can result in significant reduction in these execution times.

In a recent IBM blog post, Johnson and Faro [44] report on enhancements to their quantum computing architecture that results in a reduction in the "time to solution" for a specific chemistry problem, from several days to nine hours. This is significant in that it represents a focus on the evolutionary reduction in the time taken to compute a solution, and not solely on the relative asymptotic scaling time of quantum computing as compared to classical. This trend is likely to continue, as evidenced by current research into quantum accelerators [45]

and other architectural optimizations [46, 47].

Karalekas et al. [86] described a quantum-classical-cloud architecture that supports a tight coupling between execution of the classical and quantum parts of a hybrid algorithm with adjacent hardware, to enable faster execution of such programs over the cloud. IBM’s announcement of public availability of the Qiskit Runtime in IBM Quantum [87] is another concrete step in this direction. In a solution of this type, a user may upload a quantum program and run it with different inputs and configurations each time it is executed without having to reload the program.

Precise measurement of the execution times for various stages of a combined quantum/classical application may have particular value in gauging the evolution of these hardware advances. As the machines evolve, hand-tuning of applications should become less relevant and application benchmarks may provide a level playing field on which to compare hardware and software solutions.

Currently, the benchmark suite collects and reports on the quantum-specific execution time reported by the target system, as well as the circuit creation, launch and queue times. The iterative nature of the benchmarks means that the total execution time for any one benchmark program is a rough indicator for overall total time to solution for that type of application (factoring out the highly variable queue times). An analysis of execution times aggregated over multiple executions is not reported systematically or compared in the current suite and is proposed for a future work.

As of this writing, the benchmarks have not been structured to take advantage of the Qiskit Runtime. The net impact of implementing the close coupling of quantum and classical computation is that the overall execution time will be dramatically reduced and should be quantifiable in these benchmarks.

Some have proposed taking the classical/quantum integration a step farther and push certain parts of the classical computation down into the quantum hardware itself. In a discussion about IBM’s roadmap for the future, Wehden et al. [88] introduces a feature in which both classical and quantum instructions can be run within the coherence time of the qubits.. Takita et al. [89] discuss specifically how such a feature would improve the implementation of a quantum phase estimation algorithm.

A feature such as this opens the door to a whole new class of algorithm. It is likely that application-oriented benchmarking of such a feature would require changes to identify the portions of the classical computation that could in fact be pushed to the quantum hardware. This has not been studied yet, but such a change could be made within the framework of these benchmarks.

## 5. Qubit Connectivity Improvements

Connectivity and topology considerations are important as a focus of hardware providers at this early stage of quantum computing. In discussing the new IBM heavy hex lattice architecture, Nation et al. [90] show that spectator errors (from qubits

not concurrently involved in the execution) matter severely when running circuits of more than one or two qubits. The rate of spectator errors is directly related to the system connectivity. To achieve the best possible performance, offering end users ways to mold application circuits to specific hardware architectures is essential for current research efforts.

In the longer term, a number of user applications will be written to be independent of the target hardware in many cases. There is a heavy cost to a provider of application software to maintain hardware-specific knowledge across a range of constantly improving applications.

Our suite of application benchmarks is deliberately structured to be independent of hardware topology. Each benchmark assumes access to  $N$  fully connected qubits. We propose an enhancement to the suite that would add a mapping option to automatically map qubits of a circuit so that the circuit is effectively run on a different set of physical qubits. Running the benchmark multiple times using different mappings would provide a general means to evaluate a target system’s sensitivity to topology or connectivity. This is a simple, yet powerful way to make the benchmark suite useful as a test of topology without requiring the code within the benchmark to have specific knowledge of connectivity.

- [1] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. URL <https://quantum-journal.org/papers/q-2018-08-06-79/>.
- [2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G S L Brandao, David A Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P Harrigan, Michael J Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S Humble, Sergei V Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C Platt, Chris Quintana, Eleanor G Rieffel, Pedram Roushan, Nicholas C Rubin, Daniel Sank, Kevin J Satzinger, Vadim Smelyanskiy, Kevin J Sung, Matthew D Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-019-1666-5. URL <http://dx.doi.org/10.1038/s41586-019-1666-5>.
- [3] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.abe8770. URL <http://dx.doi.org/10.1126/science.abe8770>.
- [4] 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, Jan 2008. doi:10.1103/PhysRevA.77.012307. URL <https://link.aps.org/doi/10.1103/PhysRevA.77.012307>.
- [5] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. Scalable and robust randomized benchmarking of quantum processes. *Phys. Rev. Lett.*, 106:180504, May 2011. doi: 10.1103/PhysRevLett.106.180504. URL <https://link.aps.org/doi/10.1103/PhysRevLett.106.180504>.
- [6] Robin Blume-Kohout, John King Gamble, Erik Nielsen, Kenneth Rudinger, Jonathan Mizrahi, Kevin Fortier, and Peter Maunz. Demonstration of qubit operations below a rigorous fault tolerance threshold with gate set tomography. *Nat. Commun.*, 8:14485, February 2017. doi:10.1038/ncomms14485. URL <https://www.nature.com/articles/ncomms14485>.
- [7] Daniel C Murphy and Kenneth R Brown. Controlling error orientation to improve quantum algorithm success rates. *Phys. Rev. A*, 99(3):032318, 2019. URL <https://journals.aps.org/pr/abstract/10.1103/PhysRevA.99.032318>.
- [8] Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. Measuring the capabilities of quantum computers, 2020.
- [9] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, Apr 2018. ISSN 1745-2481. doi:10.1038/s41567-018-0124-x. URL <http://dx.doi.org/10.1038/s41567-018-0124-x>.
- [10] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3), Sep 2019. ISSN 2469-9934. doi:10.1103/physrev.100.032328. URL <http://dx.doi.org/10.1103/PhysRevA.100.032328>.
- [11] Alexander J. McCaskey, Zachary P. Parks, Jacek Jakowski, Shirley V. Moore, T. Morris, Travis S. Humble, and Raphael C. Pooser. Quantum chemistry as a benchmark for near-term quantum computers, 2019.
- [12] Kristel Michielsen, Madita Nocon, Dennis Willsch, Fengping Jin, Thomas Lippert, and Hans De Raedt. Benchmarking gate-based quantum computers. *Computer Physics Communications*, 220:44–55, 2017. ISSN 0010-4655. doi: <https://doi.org/10.1016/j.cpc.2017.06.011>. URL <https://www.sciencedirect.com/science/article/pii/S0010465517301935>.
- [13] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. C. Pienti, M. Chmielewski, C. Collins, and et al. Benchmarking an 11-qubit quantum computer. *Nature Communications*, 10(1), Nov 2019. ISSN 2041-1723. doi:10.1038/s41467-019-13534-2. URL <http://dx.doi.org/10.1038/s41467-019-13534-2>.
- [14] Daniel Koch, Brett Martin, Saahil Patel, Laura Wessing, and Paul M. Alsing. Demonstrating nisq era challenges in algorithm design on ibm’s 20 qubit quantum computer. *AIP Advances*, 10(9):095101, Sep 2020. ISSN 2158-3226. doi:10.1063/5.0015526. URL <http://dx.doi.org/10.1063/5.0015526>.
- [15] Daniel Mills, Seyon Sivarajah, Travis L. Scholten, and Ross Duncan. Application-motivated, holistic benchmarking of a full quantum computing stack. *Quantum*, 5:415, Mar 2021. ISSN 2521-327X. doi:10.22331/q-2021-03-22-415. URL <http://dx.doi.org/10.22331/q-2021-03-22-415>.
- [16] Michele Amoretti. An Effective Framework for Full-Stack Benchmarking of Quantum Computers. *Quantum Views*, 5: 52, April 2021. doi:10.22331/qv-2021-04-26-52. URL <https://doi.org/10.22331/qv-2021-04-26-52>.
- [17] Arjan Cornelissen, Johannes Bausch, and András Gilyén. Scalable benchmarks for gate-based quantum computers, 2021.
- [18] Application-Oriented Performance Benchmarks for Quantum Computing. URL <https://github.com/SRI-International/QC-App-Oriented-Benchmarks>.
- [19] Qiskit Open Source Quantum Development. <https://qiskit.org>, 2021. Qiskit website.
- [20] Google Cirq Quantum AI. <https://quantumai.google/cirq>, 2021. Google Cirq website.
- [21] Amazon Braket. <https://aws.amazon.com/braket>, 2021. Amazon Braket website.
- [22] Microsoft Q# and the Quantum Development Kit. <https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing/>, 2021. Microsoft Q# website.
- [23] Robin Blume-Kohout and Kevin C. Young. A volumetric framework for quantum computer benchmarks. *Quantum*, 4:362, November 2020. ISSN 2521-327X. doi:10.22331/q-2020-11-15-362. URL <https://doi.org/10.22331/q-2020-11-15-362>.

- [24] Amazon Braket > Devices > IonQ. <https://us-west-1.console.aws.amazon.com/braket/home?region=us-west-1#/devices/arn:aws:braket:::device/qpu/ionq/ionq-device>, 2021. Amazon Braket website, accessed 2021-05-24.
- [25] Google Quantum AI Quantum Computer Datasheet. <https://quantumai.google/hardware/datasheet/weber.pdf>, 2021. Weber datasheet, accessed 2021-05-24.
- [26] IBM Quantum. <https://quantum-computing.ibm.com>, 2021. accessed 2021-05-15.
- [27] Mohan Sarovar, Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. Detecting crosstalk errors in quantum information processors. *Quantum*, 4:321, 2020. URL <https://quantum-journal.org/papers/q-2020-09-11-321/>.
- [28] Jay M Gambetta, AD Córcoles, Seth T Merkel, Blake R Johnson, John A Smolin, Jerry M Chow, Colm A Ryan, Chad Rigetti, S Poletto, Thomas A Ohki, et al. Characterization of addressability by simultaneous randomized benchmarking. *Phys. Rev. Lett.*, 109(24):240504, 2012. URL <https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.109.240504>.
- [29] Patty Lee. Quantum Volume: The Power of Quantum Computers. <https://www.honeywell.com>, 2020. Chief Scientist for Honeywell Quantum Solutions. Retrieved 2021-05-25.
- [30] Scott Aaronson. Turn down the quantum volume, Mar 2020. URL <https://www.scottaaronson.com/blog/?p=4649>.
- [31] Note1. In the quantum volume protocol, success is quantified in terms of the heavy output probability (HOP) rather than result fidelity. The success threshold used for the HOP in the quantum volume protocol is  $2/3$ , which approximately corresponds to a threshold for result fidelity of  $1/2$ .
- [32] Colin P. Williams. *Quantum Gates*, pages 51–122. Springer London, London, 2011. ISBN 978-1-84628-887-6. doi:10.1007/978-1-84628-887-6\_2. URL [https://doi.org/10.1007/978-1-84628-887-6\\_2](https://doi.org/10.1007/978-1-84628-887-6_2).
- [33] Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 1909(136):210–271, 1909.
- [34] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.
- [35] Amazon Braket > Devices > Rigetti. <https://us-west-1.console.aws.amazon.com/braket/home?region=us-west-1#/devices/arn:aws:braket:::device/qpu/rigetti/Aspen-9>, 2021. Amazon Braket website, accessed 2021-08-18.
- [36] Achieving quantum volume 128 on the honeywell quantum computer, Sep 2020. URL <https://www.honeywell.com/us/en/news/2020/09/achieving-quantum-volume-128-on-the-honeywell-quantum-computer>.
- [37] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, and et al. Demonstration of the trapped-ion quantum ccd computer architecture, Apr 2021. URL <https://www.nature.com/articles/s41586-021-03318-4>.
- [38] Honeywell. Honeywell sets new record for quantum computing performance, 2021. URL <https://www.honeywell.com/us/en/news/2021/03/honeywell-sets-new-record-for-quantum-computing-performance>.
- [39] J. P. Gaebler, C. H. Baldwin, S. A. Moses, J. M. Dreiling, C. Figgatt, M. Foss-Feig, D. Hayes, and J. M. Pino. Suppression of mid-circuit measurement crosstalk errors with micromotion, 2021.
- [40] Honeywell. Honeywell sets another record for quantum computing performance, 2021. URL <https://www.honeywell.com/us/en/news/2021/07/honeywell-sets-another-record-for-quantum-computing-performance>.
- [41] IonQ Quantum Cloud. <https://console.cloud.google.com/marketplace/product/ionq-public/ionq?pli=1>, 2021. Amazon Braket website, accessed 2021-08-18.
- [42] Elton Yechao Zhu, Sonika Johri, Dave Bacon, Mert Esencan, Jungsang Kim, Mark Muir, Nikhil Murgai, Jason Nguyen, Neal Pisenti, Adam Schouela, Ksenia Sosnova, and Ken Wright. Generative quantum learning of joint probability distribution functions, 2021.
- [43] Timothy Proctor, Stefan Serita, Erik Nielsen, Robin Blume-Kohout, and Kevin Young. Scalable randomized benchmarking of quantum computers using mirror circuits. *In preparation*.
- [44] Blake Johnson and Ismael Faro. Ibm quantum delivers 120x speedup of quantum workloads with qiskit runtime, May 2021. URL <https://research.ibm.com/blog/120x-quantum-speedup?lnk=ushpv18re2>.
- [45] K. Bertels, A. Sarkar, T. Hubregtsen., M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, and I. Ashraf. Quantum computer architecture: Towards full-stack quantum accelerators. *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar 2020. doi:10.23919/date48585.2020.9116502. URL <http://dx.doi.org/10.23919/DATE48585.2020.9116502>.
- [46] Matthias Möller and Cornelis Vuk. On the impact of quantum computing technology on future developments in high-performance scientific computing. *Ethics and Information Technology*, 19(4):253–269, Aug 2017. ISSN 1572-8439. doi:10.1007/s10676-017-9438-0. URL <http://dx.doi.org/10.1007/s10676-017-9438-0>.
- [47] Yudong Cao and Timothy Hirzel. Quantum acceleration in 2020, Dec 2020. URL <https://www.infoq.com/articles/quantum-acceleration-2020/>.
- [48] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213, Jul 2014. ISSN 2041-1723. doi:10.1038/ncomms5213. URL <http://dx.doi.org/10.1038/ncomms5213>.
- [49] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, feb 2016. doi:10.1088/1367-2630/18/2/023023. URL <https://doi.org/10.1088/1367-2630/18/2/023023>.
- [50] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
- [51] Note2. The Quantum Fourier Transform acting on standard input states can be efficiently simulated on a classical computer using a special-purpose simulation algorithm.
- [52] Note3. Here we have used  $F_s$  rather than the normalized fidelity  $F$  for simplicity; the result for  $F$  differs only by a small  $n$ -dependent factor.
- [53] Samuele Ferracin, Theodoros Kapourniotis, and Animesh Datta. Accrediting outputs of noisy intermediate-scale quantum computing devices. *New J. Phys.*, 21(11):113038, 2019. ISSN 1367-2630. doi:10.1088/1367-2630/ab4fd6. URL <https://iopscience.iop.org/article/10.1088/1367-2630/ab4fd6>.
- [54] Noah Linden and Ronald de Wolf. Average-case verification of the quantum fourier transform enables worst-case phase estimation, 2021.
- [55] Standard Performance Evaluation Corporation. URL <https://spec.org/>. SPEC Benchmark Suite, accessed 2021-05-28.
- [56] John L. Hennessy and Patterson. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann, 2019.
- [57] The Qiskit Team. Measuring quantum volume, Aug 2021. URL <https://qiskit.org/textbook/ch-quantum-hardware/>

- measuring-quantum-volume.html.
- [58] Sandia Quantum Performance Lab. pyGSTi. <https://github.com/pyGSTio/pyGSTi>, 2021. Github repository.
  - [59] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439 (1907):553–558, 1992. doi:10.1098/rspa.1992.0167. URL <http://royalsocietypublishing.org/doi/abs/10.1098/rspa.1992.0167>.
  - [60] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. doi:10.1137/S0097539796300921. URL <https://doi.org/10.1137/S0097539796300921>.
  - [61] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, Mar 2017. ISSN 1091-6490. doi:10.1073/pnas.1618020114. URL <http://dx.doi.org/10.1073/pnas.1618020114>.
  - [62] Martin Rötteler. *Quantum algorithms for highly non-linear Boolean functions*, pages 448–457. doi:10.1137/1.9781611973075.37. URL <https://epubs.siam.org/doi/abs/10.1137/1.9781611973075.37>.
  - [63] D. Coppersmith. An approximate fourier transform useful in quantum factoring", ibm research report rc19642, ; r. cle. 1994.
  - [64] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi:10.1145/237814.237866. URL <https://doi.org/10.1145/237814.237866>.
  - [65] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002. ISSN 0271-4132. doi:10.1090/conm/305/05215. URL <http://dx.doi.org/10.1090/conm/305/05215>.
  - [66] A. Kitaev. Quantum measurements and the abelian stabilizer problem. *Electron. Colloquium Comput. Complex.*, 3, 1996.
  - [67] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172. URL <https://doi.org/10.1137/S0097539795293172>.
  - [68] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982. ISSN 1572-9575. doi:10.1007/BF02650179. URL <https://doi.org/10.1007/BF02650179>.
  - [69] Masuo Suzuki. Generalized trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems. *Communications in Mathematical Physics*, 51(2):183–190, Jun 1976. ISSN 1432-0916. doi:10.1007/BF01609348. URL <https://doi.org/10.1007/BF01609348>.
  - [70] Farrokh Vatan and Colin Williams. Optimal quantum circuits for general two-qubit gates. *Phys. Rev. A*, 69:032315, Mar 2004. doi:10.1103/PhysRevA.69.032315. URL <https://link.aps.org/doi/10.1103/PhysRevA.69.032315>.
  - [71] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018. ISSN 0027-8424. doi:10.1073/pnas.1801723115. URL <https://www.pnas.org/content/115/38/9456>.
  - [72] D. Zhu, S. Johri, N. H. Nguyen, C. Huerta Alderete, K. A. Landsman, N. M. Linke, C. Monroe, and A. Y. Matsuura. Probing many-body localization on a noisy quantum computer. *Phys. Rev. A*, 103:032606, Mar 2021. doi:10.1103/PhysRevA.103.032606. URL <https://link.aps.org/doi/10.1103/PhysRevA.103.032606>.
  - [73] Stefan Woerner and Daniel J. Egger. Quantum risk analysis. *npj Quantum Information*, 5(1):15, Feb 2019. ISSN 2056-6387. doi:10.1038/s41534-019-0130-6. URL <https://doi.org/10.1038/s41534-019-0130-6>.
  - [74] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan. Quantum computational chemistry. *Rev. Mod. Phys.*, 92:015003, 2020.
  - [75] V. V. Burg, G. H. Low, T. Häner, D. S. Steiger, M. Reiher, M. Roetteler, and M. Troyer. Quantum computing enhanced computational catalysis. *Phys. Rev. Research*, 3:033055, 2021.
  - [76] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer. Elucidating reactions mechanisms on quantum computers. *Proc. Natl. Acad. Sci.*, 114(29):7555–7560, 2017.
  - [77] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Publications, Mineola, N.Y., 1996.
  - [78] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, and et al. Quantum chemistry in the age of quantum computing. *Chemical Reviews*, 119(19):10856–10915, Aug 2019. ISSN 1520-6890. doi:10.1021/acs.chemrev.8b00803. URL <http://dx.doi.org/10.1021/acs.chemrev.8b00803>.
  - [79] P. Jordan and E. Wigner. Über das paulische äquivalenzverbot. *Zeitschrift für Physik*, 47(9):631–651, Sep 1928. ISSN 0044-3328. doi:10.1007/BF01331938. URL <https://doi.org/10.1007/BF01331938>.
  - [80] Stephane Beauregard. Circuit for shor's algorithm using 2n+3 qubits. *Quantum Info. Comput.*, 3(2):175–185, March 2003. ISSN 1533-7146.
  - [81] Nation and Johnson. How to measure and reset a qubit in the middle of a circuit execution, Feb 2021. URL <https://www.ibm.com/blogs/research/2021/02/quantum-mid-circuit-measurement>.
  - [82] Mid-Circuit Measurements Tutorial. <https://quantum-computing.ibm.com/lab/docs/iql/manage/systems/midcircuit-measurement/>, 2021. IBM Quantum Lab.
  - [83] Samuel Moore. Honeywell's ion trap quantum computer makes big leap, Mar 2020. URL <https://spectrum.ieee.org/searchContent?q=quantum%2Bsupremacy>.
  - [84] Robert B. Griffiths and Chi-Sheng Niu. Semiclassical fourier transform for quantum computation. *Physical Review Letters*, 76(17):3228–3231, Apr 1996. ISSN 1079-7114. doi:10.1103/physrevlett.76.3228. URL <http://dx.doi.org/10.1103/PhysRevLett.76.3228>.
  - [85] T. Bækkegaard, L. Kristensen, N. Loft, C. Andersen, David Petrosyan, and N. Zinner. Realization of efficient quantum gates with a superconducting qubit-qutrit circuit. *Scientific Reports*, 9, 09 2019. doi:10.1038/s41598-019-49657-1.
  - [86] Peter J Karalekas, Nikolas A Tezak, Eric C Peterson, Colm A Ryan, Marcus P da Silva, and Robert S Smith. A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology*, 5(2):024003, Apr 2020. ISSN 2058-9565. doi:10.1088/2058-9565/ab7559. URL <http://dx.doi.org/10.1088/2058-9565/ab7559>.
  - [87] Qiskit Runtime. <https://quantum-computing.ibm.com/lab/docs/iql/runtime/>, 2021. IBM Quantum Lab.
  - [88] Wehden, Gambetta, and Faro. Ibm's roadmap for building an



- open quantum software ecosystem, Feb 2021. URL <https://www.ibm.com/blogs/research/2021/02/quantum-development-roadmap>.
- [89] Takita, Lekuch, and Corcoles. Quantum circuits get a dynamic upgrade with the help of concurrent classical computation, Feb 2021. URL <https://www.ibm.com/blogs/research/2021/02/quantum-phase-estimation>.
- [90] Nation, Paik, Cross, and Nazario. The ibm quantum heavy hex lattice, Jul 2021. URL <https://www.research.ibm.com/blog/heavy-hex-lattice>.